Tree decompositions and routing problems

Bi Li

▶ **To cite this version:**

Bi Li. Tree decompositions and routing problems. Other [cs.OH]. Université Nice Sophia Antipolis, 2014. English. NNT : 2014NICE4088 . tel-01127108

HAL Id: tel-01127108

https://theses.hal.science/tel-01127108v1

Submitted on 7 Mar 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS
ÉCOLE DOCTORALE DES SCIENCES ET TECHNOLOGIES DE L'INFORMATION
ET DE LA COMMUNICATION

P H D T H E S I S

to obtain the title of

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis
Mention : INFORMATIQUE

Defended by

Bi LI

Tree Decompositions and Routing Problems

COATI Project
(INRIA, I3S (CNRS/UNS))
Advisors:
David COUDERT
Nicolas NISSE

Defended on November 12, 2014

Jury:

Reviewers:	Pascal BERTHOMÉ	-	LIFO (Bourges, France)
	Nicolas HANUSSE	-	CNRS (Talence, France)
		-	
Examinators:	David COUDERT	-	INRIA (Sophia Antipolis, France)
	Nicolas NISSE	-	INRIA (Sophia Antipolis, France)
	Hao LI	-	CNRS (Orsay, France)
	Igor LITOVSKY	-	UNSA (Sophia Antipolis, France)
	Galtier JÉRÔME	-	Orange (Sophia Antipolis, France)

Acknowledgements

First, I am deeply indebted to my advisers David Coudert and Nicolas Nisse. Without their help, this work would not have been done. Specially, Nicolas Nisse is very patient and give me methodical and systematic guidance for doing research. I would like also to thank my coauthors Jean-Claude Bermond, Xujin Chen, Xiaodong Hu, Hervé Rivano, Karol Suchan, Joseph Yu, etc.

I dedicate this thesis to all my family, specially my grandfather, who encourages me all the time to conquer my laziness and negativeness. In particular, my mother always comforts me when I feel depressed.

I'll always appreciate that Patricia Lachaume has helped me with all administrative problems and even personal ones for these four years. I would like to also thank so much to my dear friends Frédéric Giroire, Julio Araújo, Yaning Liu, Ana Karolinna Maia de Oliveira, Fatima Zahra Moataz and other Chinese friends in residence OXFORD. It is hard to imagine my life in France without their help and companion.

I could not mention everyone, but I would like to thank everyone in MASCOTTE/COATI, with whom I feel so lucky to have been.

Sophia Antipolis, France
November 1st, 2014

Tree Decompositions and Routing Problems

Abstract: A tree decomposition of a graph is a way to represent it as a tree by preserving some connectivity properties of the initial graph. Tree decompositions have been widely studied for their algorithmic applications, in particular using dynamic programming approach. In this thesis, we study tree decompositions satisfying various constraints and design algorithms to compute them in some graph classes. We then use tree decompositions or specific graph properties to solve several problems related to routing. The thesis is divided into two parts.

In the first part, we study tree decompositions satisfying some properties. In Chapter 2, we investigate minimum size tree decompositions, i.e., with minimum number of bags. Given a fixed $k \geq 4$, we prove it is NP-hard to compute a minimum size decomposition with width at most k in the class of graphs with treewidth at least 4. We design polynomial time algorithms to compute minimum size tree decompositions in some classes of graphs with treewidth at most 3 (including trees). Part of these results will be presented in ICGT 2014.

In Chapter 3, we study the chordality (longest induced cycle) of graphs and introduce the notion of good tree decomposition (where each bag must satisfy some particular structure). Precisely, we study the Cops and Robber games in graphs with no long induced cycles. Our main result is the design of a polynomial-time algorithm that either returns an induced cycle of length at least $k+1$ of a graph G or compute a k-good tree decomposition of G. These results have been published in ICALP 2012 and Algorithmica.

In the second part of the thesis, we focus on routing problems. In Chapter 4, we design a compact routing scheme that achieves good performance in the class of graphs admitting k-good tree decompositions.

In Chapter 5, we consider the prize collecting Steiner tree problem (a generalization of the Steiner-tree problem with weighted nodes and edges). We design two risk models of the problem when the weights are given as intervals. In these models, we design polynomial-time algorithms for graphs with small treewidth. These results have been published in AAIM 2010 and the journal Acta Mathematicae Applicatae Sinica.

Finally, in Chapter 6, we consider the gathering problem in grids and in presence of interferences. We design approximation algorithms (up to small additive constants depending on the interferences) for solving this problem. This work is in revision for TCS.

Keywords: Tree Decomposition, Compact Routing Scheme, Prize Collecting Steiner Tree, Gathering.

Décomposition Arborescente et Problème de Routage

Résumé :

Une décomposition arborescente d'un graphe est une manière de le représenter sous forme d'un arbre (dont les sommets sont appelés 'sac'), en préservant des propriétés de connexité. Les décompositions arborescentes ont été beaucoup étudiées pour leurs applications algorithmiques qui utilisent, en particulier, la programmation dynamique. Dans cette thèse, nous étudions les décompositions arborescentes qui satisfont certaines contraintes supplémentaires et nous proposons des algorithmes pour les calculer dans certaines classes de graphes. Finalement, nous résolvons des problèmes liés au routage en utilisant ces décompositions ainsi que des propriétés structurelles des graphes. Cette thèse est divisée en deux parties.

Dans la première partie, nous étudions les décompositions arborescentes satisfaisant des propriétés spécifiques. Dans le Chapitre 2, nous étudions les décompositions de taille minimum, c'est-à-dire avec un nombre minimum de sacs. Etant donné une entier $k \geq 4$ fixé, nous prouvons que le problème de calculer une décomposition arborescente de largeur au plus k et de taille minimum est NP-complet dans les graphes de largeur arborescente au plus 4. Nous décrivons ensuite des algorithmes qui calculent des décompositions de taille minimum dans certaines classes de graphes de largeur arborescente au plus 3 (en particulier dans les arbres). Ces résultats ont été présentés au workshop international ICGT 2014.

Dans le Chapitre 3, nous étudions la cordalité (plus long cycle induit) des graphes et nous introduisons la notion de k-good décomposition arborescente (où chaque sac doit admettre une structure particulière). Nous étudions tout d'abord les jeux de Gendarmes et Voleur dans les graphes sans long cycle induit. Notre résultat principal est un algorithme polynomial qui, étant donné un graphe G, soit trouve un cycle induit de longueur au moins k+1, ou calcule une k-good décomposition de G. Ces résultats ont été publiés à la conférence internationale ICALP'12 et dans la revue internationale Algorithmica.

Dans la seconde partie de la thèse, nous nous concentrons sur des problèmes de routage. Dans le Chapitre 4, nous concevons un algorithme de routage compact qui a de très bonnes performances dans les graphes qui admettent une k-good décomposition.

Dans le Chapitre 5, nous considérons le problème du "Price Collecting Steiner Tree". Nous proposons 2 modèles de risque pour ce problème (lorsque les poids des sommets et coûts des arêtes sont donnés sous forme d'intervalles). Pour ces deux modèles, nous proposons des algorithmes polynomiaux qui résolvent ces problèmes dans les graphes de petite largeur arborescente. Ces résultats ont été publiés dans AAIM 2010 et dans la revue Acta Mathematicae Applicatae Sinica.

Finalement, dans le Chapitre 6, nous considérons le problème de collecte d'information dans les réseaux en grille et en présence d'interférences. Nous proposons des algorithmes d'approximation (à une petite constante additive près) pour résoudre ce problème. Ce travail est en révision pour la revue internationale TCS.

Contents

CHAPTER 1

Introduction

How do people keep up to date with the happenings around the world? How do people keep in touch with friends all over the world? How do people find the information they need... Most people will answer by the Internet. It goes without saying that Internet is very important in our life.

One of the main roles of the Internet, also all kinds of communication network, e.g. transportation network, social network, is to transmit information. Then a fundamental task is to *route* the information between any pairs of its *clients*, which can be computers, stations or people, that is to find paths in the network.

How to route the information efficiently, i.e., to find a desired path from the origin to the destination quickly? Everyone knows how to go home after work everyday, because they know the way from home to work. But imagine that someone is going to visit a new city. Without a GPS or a map in their hand, they will spend a long time to find their way. We see that the map, which shows the topology of the network (city), is important to do efficient routing. It is not so difficult to get a map of a city, because we can record all the buildings and all the roads in the map. However, what about getting a map of the Internet network, Facebook, Twitter, etc.? So far, there are more than 65,000 Autonomous Systems (AS) in the Internet [fIDAC]. Furthermore, according to [Cis14], over half a billion (526 million) mobile devices and connections were added and global mobile data traffic grew by 81% in 2013. There were nearly 22 million *wearable devices*[1] in 2013 generating 1.7 petabytes of monthly traffic. The telecommunication networks are not only becoming larger and larger but also more and more dynamic. All this brings difficulties in getting a map, i.e. in knowing the global topology of the network. It is almost impossible to get it because the Internet is changing all the time with many new devices connecting to the Internet and at the same time many devices disconnecting from the Internet.

How does the Internet route nowadays? Routing in the Internet is mainly guided by Border Gateway Protocol (BGP). Simply speaking, in BGP, every router stores a path to any other router in the Internet. Since the Internet is changing all the time, the paths stored in each router is also changing accordingly. Moreover, every router needs a big memory to store the paths since there are huge numbers of routers in the Internet. Because of the growth of the Internet, BGP may not work any more in the future. It is important to improve BGP and also explore new protocols for routing in the Internet.

[1] The so called *wearable devices* are devices that can be worn on a person, which have the capability to connect and communicate to the network either directly through embedded cellular connectivity or through another device (primarily a smartphone) using Wi-Fi, Bluetooth or another technology, such as smart watches, smart glasses, health and fitness trackers, health monitors, wearable scanners, navigation devices, smart clothing, and so forth [Cis14].

A lot of research is dedicated to implement experiments to measure some properties of large-scale networks, e.g. the Internet network. There are already many well known structural properties of the large-scale networks, such as *logarithmic diameter* (the diameter of the network is much smaller than the number of clients), *power-law degree distribution* (there are few clients connected with many other clients) and *high clustering coefficient* (two clients connected with another one common client has big probability to be connected) [BA99, AJB99, SFFF03, FFF99]. Traditionally, a network can be modeled by a graph with the vertex set of all the clients and the edge set of possible connections between two clients. Using some *graph libraries* or *network simulators*, e.g. Grph, DRMsim, NS simulator, researchers and engineers are able to generate graph models satisfying some of the above structural properties of the large-scale networks. Moreover, many efficient routing algorithms are designed for these graph models [KFY04, CSTW12]. There is big probability that these algorithms work also well in the real life large-scale networks, because these graph models satisfy the same properties. So it is interesting to design efficient algorithms for graph models of large-scale networks. To design efficient algorithms for any graph, it is helpful to take advantage of some particular structural properties of the graph. The main motivation of my thesis is to explore the structural properties of graphs, which may be used for algorithmic purposes.

One of the simplest classes of (undirected) graphs is the class of *trees*, connected acyclic graphs. Many difficult (*NP-complete*) problems become easy (*polynomially solvable*) in the class of trees, see e.g. [DN66, CGH75]. It is conceivable that many problems should be also easy in a class of graphs 'close' to trees. But how to define that a graph is 'close' to a tree? This question leads to the main topic of my PhD thesis, *tree decomposition*, which is a way to study tree-likeness of a graph. Roughly speaking, a *tree-decomposition* of a graph maps each vertex of the graph to a subtree of the *decomposition tree* in a way that the subtrees assigned to the adjacent vertices intersect [RS86b, Bod98]. Such decompositions play an important role in design of efficient algorithms. Let us see some definitions before presenting its algorithmic applications.

Definitions and Notations

In this section, we give some basic definitions used in this thesis. All the definitions in this section can be found in [BM08].

For any two sets A and B, if any element in B in contained in A, we say that B is a subset of A, denoted as $B \subseteq A$; if $B \subseteq A$ and $B \neq A$ then we say B is a proper subset of A, denoted as $B \subset A$. Let $G = (V,E)$ be any undirected simple graph. For any two adjacent vertices $u, v \in V$, we denote the edge as uv. Any graph $G' = (V',E')$ with $V' \subseteq V$ and $E' \subseteq \{uv \in E : u,v \in V'\}$ is a *subgraph* of G; moreover G' is an *induced subgraph* of G, or *subgraph induced by* V', denoted as $G[V']$, if $E' = \{uv \in E : u,v \in V'\}$. For a set $M \subseteq V \cup E$, we denote by $G \setminus M$ the subgraph $(V \setminus M, (E \setminus M) \setminus \{uv : u \in M \text{ or } v \in M\})$ of G. For any subgraph H of G, denoted as $H \subseteq G$, we use $V(H)$ and $E(H)$ to denote the vertex and edge set of H, respectively. A graph is *connected* if there is a path between any two vertices in it. A graph is *disconnected* if it is not connected. A *connected component* of a graph G is

a maximal connected subgraph of G.

A graph is *complete* if any of its two vertices are adjacent. We denote by K_n a complete n-vertex-graph. Given a graph G, a subgraph K of G is a *clique* if K is complete. A graph $G = (V,E)$ is *bipartite* if there exist two disjoint subsets $A,B \subseteq V$ such that $V = A \cup B$ and each edge in E is incident to a vertex in A and the other one in B; moreover G is a *complete bipartite* graph if each vertex in A and each vertex in B are adjacent. We denote the complete bipartite graph G by $K_{|A|,|B|}$. A graph is *planar* if it can be embedded in the plane, i.e., it can be drawn on the plane in such a way that its edges intersect only at their endpoints. In other words, it can be drawn in such a way that no edges cross each other. Such a drawing is called a *planar embedding* of the graph. A planar graph is *outerplanar* if it has a planar embedding such that all its vertices belong to the unbounded face.

Given a graph $G = (V,E)$, the *distance* between two vertices $u,v \in V$ in G is the *length*, i.e., the number of edges, of a shortest path between them in G. For any vertex $v \in V$, the *open neighborhood* of v is defined as $N_G(v) = \{u : uv \in E\}$; the *closed neighborhood* of v is defined as $N_G[v] = N_G(v) \cup \{v\}$. When the graph is clear from the context, without confusion we use $N(v)$ or $N[v]$ simply. The *degree* of a vertex v in G is $|N_G(v)|$. We denote by Δ the maximum degree over all vertices in G. A subset $D \subseteq V(G)$ is a *dominating set* of the graph G if any vertex not in D is adjacent to some vertices in D.

Tree Decomposition

Definition 1. *Given a graph $G = (V,E)$, a tree decomposition $(T,\mathcal{X})$ of G consists of a tree T and a family $\mathcal{X} = \{X_t : t \in V(T)\}$ of subsets of V, called bags, such that:*

- $\cup_{t \in V(T)} X_t = V$;
- *for any edge $e = uv \in E$, there exists a node $t \in V(T)$ such that the bag X_t contains both u and v;*
- *for any vertex $v \in V$, the nodes in $\{t \in V(T) : v \in X_t\}$ induces a subtree of T.*

The *width* of $(T,\mathcal{X})$ is $\max_{t \in V(T)} |X_t| - 1$, i.e. the maximum size of a bag minus one. The *treewidth*[2] of G, denote by $tw(G)$, is the minimum width among all the tree decompositions of G [RS86b]. For each bag X_t, the *diameter* of X_t, denoted by $diam(X_t)$, is the maximum distance in G between any two vertices in X_t. The *length* of $(T,\mathcal{X})$ is $\max_{t \in V(T)} diam(X_t)$, i.e. the maximum diameter of all bags in $\mathcal{X}$. The *treelength* of G, denote by $tl(G)$, is the minimum length among all the tree decompositions of G [DG07].

Moreover, $(T,\mathcal{X})$ is a *path decomposition* of G if T is a path. The *pathwidth* of G, denote by $pw(G)$, is the minimum width among all the path decompositions of G [RS83].

Given a tree decomposition $(T,\mathcal{X} = \{X_t : t \in V\})$ of G, without confusion, we identify the bag $X_t \in \mathcal{X}$ and the node $t \in V(T)$.

It is well-known that forests, union of disjoint trees, are exactly the class of graphs of treewidth at most 1.

[2]Note that, while the term treewidth has been defined by Robertson and Seymour in their work on Graph Minors, similar notions already appeared in previous work, e.g., [Ros74].

Tree decomposition has been first introduced by Robertson and Seymour in the *Graph Minor theory* [RS86b]. Let us give some basic definitions about graph minor.

Given an edge $uv \in E(G)$, the *edge contraction* of uv in G, is to identify vertices u and v in G and removing all loops and duplicate edges. The obtained graph is denoted by $G/\{uv\}$. A graph H is a *contraction* of a graph G if H can be obtained by a sequence of edge contractions from G. Particularly, we say that G is also a contraction of G. A *minor* of a graph G is a contraction of a subgraph of G. (Note that G is also a subgraph of G.) Given an n-vertex-graph G, for every fixed graph H there exists an $O(n^3)$ algorithm testing whether H is a minor of G, where the big O notation hides a constant that depends superexponentially on $|V(H)|$; while, if G is given with a tree decomposition of bounded treewidth, then it can be tested in linear (but exponential in $|V(H)|$) time whether H is a minor of G [RS95]. Kawarabayashi et. al improved their time complexity to $O(n^2)$ (for general graph G) in [iKKR12].

The famous Wagner's theorem says that a graph is planar if and only if it does not contain K_5 and $K_{3,3}$ as a minor. It is well-known that a graph is outerplanar if and only if it does not contain K_4 and $K_{2,3}$ as a minor. A graph is a *partial* 2*-tree* if it has treewidth at most 2. Wald and Colbourn proved that:

Theorem 1. *[WC83] A graph is a partial 2-tree if and only if it does not contain K_4 as a minor.*

In Fig. 1.1, we see the clear hierarchy of the classes of forest, outerplanar graphs, partial 2-trees and planar graphs.

A class of graphs $\mathcal{C}$ is *minor-closed* if any minor of any graph in $\mathcal{C}$ is also a member of $\mathcal{C}$. Given a minor-closed class of graphs $\mathcal{C}$, denote its complementary as $\bar{\mathcal{C}}$, i.e. the set of graphs not in $\mathcal{C}$. A graph $F \in \bar{\mathcal{C}}$ is a *forbidden minor* of $\mathcal{C}$ if F is *minimal* in $\bar{\mathcal{C}}$, i.e. no other graphs in $\bar{\mathcal{C}}$ is a minor of F, equivalently any minor of F is in $\mathcal{C}$. So Theorem 1 says K_4 is the only forbidden minor of the class of partial 2-trees. Robertson and Seymour proved that:

Theorem 2. [Graph Minor Theorem] *Every minor-closed class of graphs has finite number of forbidden minors [RS04].*

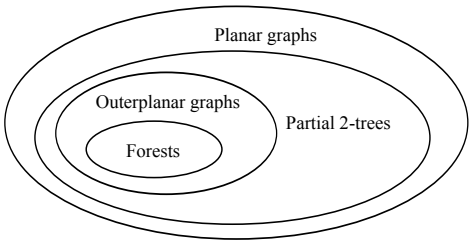


Figure 1.1: The hierarchy of the classes of forest, outerplanar graphs, partial 2-trees and planar graphs.

Tree decompositions are also used in parameterized complexity theory. A *parameter* is a function mapping graphs to nonnegative integers, e.g. chordality, treewidth. A *parameterized problem*, taking as parameter a fixed nonnegative integer k and as input an

instance I, is *Fixed Parameter Tractable (FPT)* if there is an algorithm that solves it in time $f(k)n_I^{O(1)}$, where n_I is the size of the instance and f is a function depending only on k. We are going to see later that many parameterized problems are FPT, which takes as input a graph G and as parameter the treewidth of G.

Plenty of work is devoted to compute the treewidth of a given graph. One of the well-known exact algorithms for treewidth computation is given by Fomin et al. [FKTV08], which is base on *minimal triangulation* of a graph. We present the related definitions in the following.

A graph G is *k-chordal* for an integer k if there is no *induced cycle*, cycle without chords, longer than k. The *chordality* of a graph G is the minimum k such that G is k-chordal. In particular, a 3-chordal graph G is also called *chordal graph*. Given a graph $G = (V,E)$, a graph $G^+ = (V,E \cup F)$ is a *triangulation* of G if G^+ is chordal; moreover G^+ is a *minimal triangulation* of G if for any proper subset $F' \subset F$, the graph $H = (V,E \cup F')$ is not chordal. That is to say that a minimal triangulation of G is obtained by adding a minimal set of edges to G to make it chordal. There is a $n^{2.376}$ time algorithm for finding a minimal triangulation of an n-graph in [Mez11].

Chordal graphs have many different characterizations, see e.g. a survey [Heg06]. A *clique tree* of a graph G is a tree decomposition of G such that the vertices in each bag of the tree decomposition induces a clique in G. A graph G is chordal if and only if it has a clique tree [Gav74, Bun74].

Given a graph G and a tree decomposition $(T,\mathcal{X})$ of width $tw(G)$, construct a graph G^+ by adding an edge uv in G if there exists a bag in $(T,\mathcal{X})$ containing both u and v. Then $(T,\mathcal{X})$ is a clique tree of G^+. So G^+ is a chordal graph. This gives hints to the following well known fact: *G has treewidth at most k if and only if G has a minimal triangulation with maximum clique of size at most $k+1$.*

So the treewidth of a graph is closely related to the cliques in its minimal triangulations. Given a graph $G = (V,E)$, a *potential maximal clique* is a set of vertices in G which induces a maximal clique in some minimal triangulation of G.

Given a graph $G = (V,E)$, two vertices $u,v \in V$ in a same connected component of G and a subset $S \subseteq V \setminus \{u,v\}$, we say that S *separates* u and v, if u and v are in different connected component of the subgraph $G \setminus S$; S is a *separator* of G if there exist $u,v \in V$ such that S separates u and v. Moreover $S \subset V$ is a *minimal separator* if there exist $u,v \in V$ such that S separates u and v but none of its proper subset $S' \subset S$ separates u and v. In general, for any three subsets $U,W,S \subseteq V$, we say that S *separates* U and W if each path between any vertex $u \in U$ and any vertex $w \in W$ contains a vertex in S. Bouchitté and Todinca proved that treewidth can be computed in polynomial time in the class of graphs, which have polynomial number of minimal separators [BT02].

Treewidth Computations

In this section, we present some previous work focusing on the treewidth computation.

Many *NP-hard* problems in general graphs have been shown to be polynomially solvable in the class of graphs with bounded treewidth. Particularly, the famous Courcelle's

theorem states that all problems expressible in *monadic second order logic*[3] can be solved in linear time in the class of graphs with bounded treewidth [Cou90]. It means that the algorithm is linear time in the size of a given graph but maybe exponential in its treewidth. Due to the important role of treewidth, the problem 'Given an n-vertex-graph $G = (V,E)$ and an integer k, is the treewidth of G at most k?' received much attention.

This problem has been proved to be NP-complete by Arnborg et al. in [ACP87]. Additionally, they showed that the problem is solvable in $O(n^{k+2})$ time for a fixed k. Later, Robertson and Seymour proved that there exists an $O(3^{3k}n^2)$ time algorithm for this problem [RS95]. Their algorithm consists of two steps: first, apply an $O(3^{3k}n^2)$ algorithm that either decides that the treewidth of G is bigger than k, or computes a tree decomposition of G with width at most $4k+3$; if the first step outputs a tree decomposition of G with width at most $4k+3$, then implement the second step: Since the class of graphs with treewidth at most k is minor-closed (see the reason in the subsection Graph Minor), it has a finite number of forbidden minors. Given G with a tree decomposition of width at most $4k+3$, they check in linear time whether G has treewidth at most k by testing whether G has some of those forbidden minors as minors. Note that the second step is non-constructive because the forbidden minors are known to be finite but not listed. The time complexity is dominated by the first step. (Actually, the results in [RS95] are presented in terms of *branchwidth*, which is equivalent to treewidth up to a constant factor [RS91]. So this is an unimportant technical difference.)

Plenty of work is devoted to improve the time complexity of the two steps in this algorithm. Based on the notion of *'balanced separators'*, Lagergren [Lag96] and Reed [Ree92] can implement the first step in $k^{O(k)}n\log^2 n$ and $k^{O(k)}n\log n$ time respectively. Bodlaender made it be linear time and the total time complexity was $k^{O(k^3)}n$, by reducing the problem for G in linear time to a problem on a smaller graph, which is a minor of G [Bod96]. On the other hand, the second step can be improved without using graph minors; and there exist linear time (single exponential in k) constructive algorithms for deciding whether G has treewidth at most k if G is given with a tree decomposition of width $O(k)$(see e.g. [BK91, LA91]).

Note that given a graph G of treewidth k, the first step of the algorithm outputs a tree decomposition of G with width at most $4k+3$. Then it is a 4-approximation for the treewidth. All the algorithms above have time complexities either not linear in n or not single exponential in k. Recently, Bodlaender et al. has given the first algorithm of time complexity linear in n and single-exponential in k [BDD$^+$13]. More precisely, given a graph G and an integer k, their algorithm, in time $2^{O(k)}n$, either outputs that the treewidth of G is bigger

[3] *Monadic second-order formulas* in the language of graphs are built up from:

- atomic formulas $E(x,y)$, $x = y$ and $X(x)$ (for set variable X and individual variables x,y) by using the usual Boolean connectives $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication), and $\leftrightarrow$ (equivalence) and
- existential quantification $\exists x$, $\exists X$ and universal quantification $\forall x$, $\forall X$ over individual variables and set variables.

Individual variables range over vertices of a graph and set variables are interpreted by sets of vertices. The atomic formula $E(x,y)$ expresses adjacency, the formula $x = y$ expresses quality and $X(x)$ means that the vertex x is contained in the set X.

that k, or constructs a tree decomposition of G of width at most $5k+4$. So for a graph with treewidth at most k, there is a 5-approximation algorithm for finding its treewidth in time $2^{O(k)}n$. Polynomial time approximation (but not constant ratio) algorithms for treewidth can be found in [Ami10, FHL08].

In [BB05, BK07, BK10, BK11], several heuristic algorithms are presented for computing lower or upper bounds of the treewidth of a given graph.

There are also exact (exponential) algorithms for finding the treewidth of a given graph. In [BT01], Bouchitté and Todinca gave an algorithm for computing the treewidth of a given graph in polynomial time in the number of potential maximal cliques. In [BT02], they listed all potential maximal cliques of a graph in polynomial time in the number of minimal separators of the graph. So if we list all the minimal separators or all the potential maximal cliques of a given graph, then the treewidth can be computed. Based on this idea, Fomin et al. gave an $O(1.8899^n)$ time algorithm to find the treewidth of an n-vertex-graph in [FKTV08]. Fomin and Villanger presented an $O(1.7549^n)$ time algorithm in [FV12].

Algorithmic Applications of Tree Decompositions

Tree decompositions have been used in many areas (see a survey e.g. [Bod93]). We present in this section some of their algorithmic applications in graph problems.

Graph Minors

In this subsection, we briefly introduce the role of tree decomposition in proving the *Graph Minor Theorem* [RS86b].

First let us show that the class of graphs with bounded treewidth are minor-closed. Given a graph $G=(V,E)$, let $(T,\mathcal{X})$ be a tree decomposition of width k of G. Then $(T,\mathcal{X})$ is also a tree decomposition of any subgraph obtained by deleting any edges of G; for any vertex $v\in V$, deleting v in all bags of $(T,\mathcal{X})$, $(T,\mathcal{X})$ becomes a tree decomposition of width at most k of the subgraph $G\setminus\{v\}$ of G; for any edge $uv\in E$, identifying u and v in all bags of $(T,\mathcal{X})$, $(T,\mathcal{X})$ becomes a tree decomposition of width at most k of the contraction graph $G/\{uv\}$ of G. So the treewidth of a minor of the graph G is at most the treewidth of G.

The *Graph Minor Theorem*, see in Theorem 2, is proved by Robertson and Seymour in a series of twenty papers spanning over 500 pages from 1983 to 2004. In particular, they proved that the set of graphs with bounded treewidth has a finite set of forbidden minors [RS90]; and that the class of graphs having a fixed planar graph as a forbidden minor has bounded treewidth [RS84, RS86a]. These two facts play an important role in the proof of the Graph Minor theorem. Particularly, the *grid minor theorem* is involved in the proofs, which says that any graph either has bounded treewidth or has a large grid as minor [RST94]. More precisely, we display the related results on *grid minor theorem* in the following.

Theorem 3. [grid minor theorem] *Let G be a graph with treewidth at least k, then G contains $r\times r$-grid as a minor, where $r\geq f(k)$ for some function f:*

- $f(k)=\Omega(\sqrt{\log k/\log\log k})$ *[iKK12];*

- *$f(k) = \Omega(k^\delta)$ for a fixed constant $\delta > 0$ [CC13];*
- *if G is H-minor free, then $f(k) = \Omega(k)$ [DH08b, iKK12].*

There exists graph with treewidth at least k but no $O(\sqrt{k/\log k}) \times O(\sqrt{k/\log k})$ grid as a minor [RST94].

We are going to see later that the *grid minor theorem* is one of the bases of the *bidimensionality theory*.

Dynamic programming

In this subsection, we present *dynamic programming*, the main technique for efficiently solving problems in bounded treewidth graphs.

As mentioned before, many NP-hard problems in general graphs are polynomially solvable in the class of graphs with bounded treewidth, e.g. *maximum independent set problem, minimum dominating set problem, coloring problem, etc*. Given an n-vertex-graph G these problems can be solved in $c^{tw(G)}n^{O(1)}$ time for a constant c, see e.g. [AP89, TaP97]. Such algorithms generally proceed by *dynamic programming*.

Given an n-vertex-graph G and a tree decomposition $(T,\mathcal{X})$ of width $tw(G)$, let bag X_r be the root of T. Consider e.g. the *maximum independent set problem*, in which it is required to find a maximum set of vertices in G that are pairwise non-adjacent. Each leaf bag X_f induces a subgraph of G with at most $tw(G)+1$ vertices. Let X_p be the parent bag of X_f in T. Consider the intersection set $S = X_f \cap X_p$. The set S separates $X_f \setminus S$ and $V(G) \setminus X_f$. For each independent set $I \subseteq S$ in the induced subgraph $G[X_f]$, find a 'partial optimal solution' M_I, which is a maximum independent set in $G[X_f]$ and satisfies $M_I \cap S = I$. This can be done in $O(2^{tw(G)})$ time because $|X_f| \leq tw(G)+1$. For each non-leaf-bag X_i in $(T,\mathcal{X})$, let G_i be the subgraph of G induced by vertices in X_i and its descendants in T. For all children X_js of X_i in T, we combine the 'partial optimal solutions' of the subgraphs G_js to get 'partial optimal solutions' for the subgraph G_i, until we get an optimal solution for the graph $G_r = G$. More precisely, consider each independent set $Y \subseteq X_i$ in the subgraph G_i. Let X_j be a child of X_i in T. Denote $S_{ij} = X_i \cap X_j$. Then a maximum independent set M_Y in G_i s.t. $M_Y \cap X_i = Y$ can be obtained by the union of Y and the 'partial optimal solution' containing $S_{ij} \cap Y$ for each child X_j of X_i in the subgarph G_j. Let S_p be the intersection of X_i and its parent bag. Among all M_Y with the same $Y \cap S_p$, record one M_Y with maximum size. This can be done in $O(2^{tw(G)}n)$ because there are at most $2^{tw(G)}$ independent sets in X_i and X_i has at most $n-1$ children. So it takes $O(2^{tw(G)}n^2)$ time totally.

The key point of the above dynamic programming algorithm is that the intersection of two adjacent bags in the tree decomposition is a separator of the graph. Moreover, for the locally checkable problem, e.g. maximum independent set problem, it is sufficient to record an optimal partial solution respective to each subset of this intersection. On the other hand, the problems involving global constraints, e.g. the connectivity problem, may require to record an optimal partial solution respective to each ordered subset of this intersection. Generally it leads to running time $tw(G)^{tw(G)}n^{O(1)}$.

It was a long-standing challenge to find algorithms for the 'global constraints' problem with time complexity single exponential on the treewidth. Fortunately, combining the dynamic programming with other techniques, such as *sphere cut decomposition, Catalan number, matrix rank and determinants*, the connectivity problem can also be solved in time single exponential in the treewidth, i.e. $2^{O(tw(G))}n^{O(1)}$. Please see details in [DPBF10, ST10, DFT12, BCKN13].

From the time complexity of the above algorithms ($2^{O(tw(G))}n^{O(1)}$ or $tw(G)^{tw(G)}n^{O(1)}$), we get that all the problems above are FPT with parameter treewidth of a graph. We are going to see in the next subsection that some parameterized problems with a parameter k are FPT sub-exponential (i.e. $2^{o(k)}n^{O(1)}$) solvable by using *bidimensionality theory*.

Bidimensionality Theory

In this subsection, we present some previous work on *bidimensionality theory*, which combines the grid minor theorem and dynamic programming technique in the previous subsections.

The *Bidimensionality theory* has been introduced and developed in a series of papers, see in a survey [DH08a] and references in it. A parameter P is *(minor) bidimensional* if

(i) there exists a function g such that the parameter P is at least $g(r)$ in an $r \times r$-grid $G_{r\times r}$; and

(ii) it does not increase when taking minors, i.e. for any graph G and its minor H, $P(H) \leq P(G)$.

Consider a *decision problem associated with a parameter P*, for a given graph G and a nonnegative integer p, asking whether $P(G) \leq p$. Let P be a ($g(r)$-) bidimensional parameter. Let f be the same function as in Theorem 3, if the treewidth of G is $\Omega(f^{-1}g^{-1}(p))$, then G has a grid $G_{g^{-1}(p)\times g^{-1}(p)}$ as a minor. So $P(G) \geq P(G_{g^{-1}(p)\times g^{-1}(p)}) > p$; otherwise, the treewidth of G is bounded by a function of p. If we can find a tree decomposition of bounded width of G, then the problem can be solved by using dynamic programming presented in Section 1. So to solve this problem it is sufficient to answer : given a graph G and an integer $k > 0$ ($\Omega(f^{-1}g^{-1}(p))$), is the treewidth of G at most k? And if yes, find a tree decomposition of width $O(k)$ of G. We have seen that this can be done in time $(2^{O(k)}n)$ in the previous section [BDD+13]. So when $f^{-1}g^{-1}(p) = o(p)$, we get an FPT sub-exponential time algorithm.

Moreover, this technique has been extended for parameters with less constrained properties, called *contraction-bidimensionality* [DFHT05, DH08a]. The main differences are that *contraction-bidimensionality* parameter does not increasing when taking contraction instead of minor, and it is *'large'* (i.e. at least $g(r)$) in a *'grid like'* graph instead of grid. But the *grid like* graphs are only defined for the classes of *planar graphs, bounded genus graphs and apex-minor-free graphs*. For instance, a *planar grid like graph* is an $r \times r$ grid partially triangulated by additional edges that preserve planarity. For more details please see in [DFHT05, DH08a].

In particular, the parameter chordality of a graph is contraction-bidimensional. The decision problem associated with the chordality, for a given graph G and a nonnegative

integer p, asks whether the chordality of G is at most p. We are going to show that this problem is FPT (and subexponential on the parameter chordality) in the class of planar graphs in Chapter 3.

Other parameters related with tree decompositions

In this section, we present some limitations of the application of tree decompositions of bounded width. Then we introduce some other parameters related to tree decompositions, which are used for efficiently solving some graph problems.

We have seen that tree decompositions of bounded width are the corner-stone for solving efficiently many graph problems. However, the algorithms presented above for computing a tree decomposition of bounded width of a given graph are mainly of theoretical importance due to the hidden huge constants or multiple-exponential functions of the parameters. Only for the class of graphs of treewidth at most 4, there are practical linear algorithms for finding an optimal tree decomposition by a finite set of *reduction rules*, which reduces a graph to an empty graph if and only if the graph has treewidth at most 4 [AP86, MT91, San96]. Indeed, for any nonnegative integer k, there is a finite set of reduction rules for the class of graphs of treewidth at most k [ACPS93, BF96]. But for $k > 4$, the algorithms based on reduction rules may require big (exponential on k) memories [Bod07]. The lack of practical algorithm for treewidth computation is still a bottleneck of its applications in practice. Moreover, unless $P = NP$ there is no polynomial time algorithm that given a graph G computes a tree decomposition of width within an additive constant of $tw(G)$ [BGHK91]. Assuming the *Small Set Expansion Conjecture*[4], the treewidth, also the pathwidth, are NP-hard to approximate within a constant factor [APW12]. In addition, the complexity of the computing the treewidth of planar graphs is still a longstanding open problem.

On the other hand, there also exist problems, which remain hard for the class of graphs with constant bounded treewidth, but become polynomially solvable on some class of graphs with unbounded treewidth. For instance, the *bandwidth problem*[5] is NP-complete even for trees [GGJK78, Mon86]. But this problem is polynomially solvable in the class of *interval graphs* [KV90]. A graph is an *interval graph* if every vertex in the graph can be associated with an interval in the real line so that two vertices are adjacent in the graph if and only if the two corresponding intervals intersect.

[4]Let $G = (V,E)$ be an undirected *d-regular graph*, i.e. every vertex has degree d in G. For a set $S \subseteq V$ of vertices, the *normalized edge expansion* of S is $\Phi_G(S) \equiv \frac{|E(S,V\setminus S)|}{d|S|}$, where $E(S,V \setminus S) = \{uv \in E : u \in S, v \in V \setminus S\}$. The *Small Set Expansion Problem* with parameter η and δ, denoted by $SSE(\eta,\delta)$, distinguish two cases:

Yes there exists an $S \subseteq V$ with $|S| = \delta|V|$ and $\Phi_G(S) \leq \eta$;

No For any subset $S \subseteq V$ with $|S| = \delta|V|$ it holds that $\Phi_G(S) \geq 1 - eta$.

The *Small Set Expansion Conjecture* says that For any $\eta > 0$, there is a $\delta > 0$ such that $SSE(\eta,\delta)$ is NP-hard.

[5]Given an n-vertex graph, a linear arrangement is a numbering of the vertices from 1 to n (which can be viewed as a layout of the graph vertices on a line) and its bandwidth is the maximum difference in numbers given to the endpoints of an edge (the maximum stretch of an edge on the line). The (minimum) bandwidth problem asks for a linear arrangement of minimum bandwidth.

A *clique path* of a graph G is a path decomposition of G such that each of its bag induces a clique in G. A graph G is interval if and only if it has a clique path. So the class of interval graphs has unbounded treewidth. The algorithm in [KV90] for solving the bandwidth problem in interval graphs used the clique paths of interval graphs, which is actually a special path decomposition, and also a tree decomposition satisfying some structure properties. The complexity of computing the pathwidth is NP-complete even in the class of chordal graphs [Gus93]. We do not know any good approximation algorithms for computing pathwidth in chordal graphs. It is an on-going work not included in this thesis.

In [DG07] Dourisboure and Gavoille introduce the treelength of a graph, which studies the structures inside each bag of the tree decomposition, the maximum diameter of each bag. Note that chordal graphs are exactly graphs of treelength 1. It is NP-complete to decide wether a given graph has treelength at most any fixed $k \geq 2$ [Lok10], but it has an easy linear time 3-approximation algorithm [DG07]. The bounded treelength graphs are proved to admit *compact routing schemes* [Dou05] and *sparse additive spanners* [DG04]. Recently, Dragan and Köhler introduced in [DK11] another parameter related with treelength, named *treebreadth*, which is the minimum of the maximum *radius*[6] of all the bags among all the tree decompositions. Treebreadth is proved to be useful for *tree spanner problem* [DK11, DAA13].

Motivated by application in parallel and dynamic graph algorithm, Bodlaender and Hagerup investigated the tradeoff between the width and the diameter of tree decompositions [BH98]. Note that here the diameter is not the diameter of the bags in the tree decompositions, but the diameter of the tree.

We are going to study the tradeoff between the width and the size (number of bags) of tree decompositions in Chapter 2. Continuing the idea of exploiting the structural properties of the tree decompositions, we are going to study the dominating set of each bag of the tree decompositions in Chapter 3. In the rest of the chapter, we describe the contributions and organization of the thesis.

Contributions and Organization of the Thesis

This thesis studies mainly two topics: *tree decompositions*, such as minimum size tree decompositions, k-good tree decompositions; and their applications to some routing problems, such as the compact routing schemes, prize collecting Steiner trees and gathering problems. In the remainder of this section, we summarize the main contributions and organizations of the thesis.

Part I Tree Decomposition. In this part, we study two new parameters concerning tree decompositions. It consists of two chapters:

[6]Given a graph G and a tree decomposition $(T, \mathcal{X})$ of G, the radius of a bag X is the minimum of the maximum distance in G from a vertex in the bag X to all other vertices in X.

Chapter 2: Minimum Size Tree Decomposition. This chapter is dedicated to study the *minimum size tree decomposition (MSTD) problem*, in which the *size* of a tree decomposition is the number of its bags. Given a graph G and an integer $k \geq 1$, the objective of the problem is to find a tree decomposition of G of width at most k with minimum size among all its tree decompositions. For a given graph G and a nonnegative integer s, we prove that it is NP-complete to decide whether there is a tree decomposition of G of size at most s and width at most k, for any fixed integer $k \geq 4$. Moreover, it is also NP-complete when $k \geq 5$ and G is connected. Given a fixed integer $k > 0$ and a graph $G = (V,E)$ of width at most k, a general approach is proposed for computing a minimum size tree decomposition of width at most k. A *k-potential-leaf* of G is a set $S \subseteq V$ such that there is a minimum size tree decomposition $(T,\mathcal{X})$ of G satisfying that $S \in \mathcal{X}$ is a leaf bag in T. We show that if we can find an algorithm for computing a k-potential-leaf in the class of graphs $\mathcal{C}$ in time $g(|V(G)|)$, then we can construct a minimum tree decomposition in time $O(g(|V(G)|) \cdot |V(G)|)$ in the class of graphs $\mathcal{C}$. For $k = 2$, we present a linear ($O(n+m)$) time algorithm to find a 2-potential-leaf of the class of partial 2-trees with n vertices and m edges. For $k = 3$, we give linear (number of vertices) time algorithm for computing 3-potential-leaf only for the class of trees and 2-connected outer planar graphs. So we can construct a minimum size tree decomposition of width at most 2 or 3 in the corresponding classes of graphs. This result has been presented in the conference ICGT 2014 [c-LMN14].

Chapter 3: k-Good Tree Decomposition. This chapter focuses on a new structural decomposition, called *k-good tree decomposition*. A graph is called *k-super-caterpillar* for $k \geq 2$ if it has a *dominating path*, i.e., a dominating set of size at most $k-1$ inducing a path in it. A k-good tree decomposition of a graph is a tree decomposition such that each of its bags X induces a k-super-caterpillar in the graph. The main result is inspired by the study of *Cops and Rober games* in k-chordal graphs. In this game, a player starts by placing $c \geq 1$ *cops* on some vertices of a graph, then a visible *robber* is placed on one vertex of the graph. Alternately, the cop-player may move each cop along one edge, and then the robber can move to an adjacent vertex. The robber is captured if, at some step, a cop occupies the same vertex. The *cop-number* of a graph is the minimum number of cops such that there exists a strategy for the cop-player that assures to capture the robber whatever he does. We show that the cop-number of any k-chordal graph is at most $k-1$. Particularly, we prove that 2 cops are sufficient to catch a robber in any 4-chordal graph. The proofs of these facts lead us to the main result of this chapter, the k-good tree decompositions.

We prove that there is a polynomial time algorithm, which given a graph G and an integer $k \geq 3$, either returns an induced cycle of length at least $k+1$ in G or computes a k-good tree decomposition of G. For graphs admitting such a decomposition, we give upper bounds for its treewidth ($\leq (k-1)(\Delta-1)+2$), treelength ($\leq k$), hyperbolicity ($\leq \lfloor \frac{3}{2}k \rfloor$). Particularly, any k-chordal graph admits a k-good tree decomposition and then it has treewidth at most $O(k\Delta)$ improving the exponential bound of [BT97]. This implies that our tree decomposition may be used efficiently for solving problems using dynamic programming in graphs of small chordality and small maximum degree. This result has appeared in the proceedings of the conference ICALP 2012 [c-KLNS12] and in the journal

Algorithmica [j-KLNS14].

It is well-known that it is NP-hard to compute the chordality in planar graphs. Based on above result, given a graph G and an integer $k \geq 3$, to decide whether the chordality of G is at least k, it is sufficient to find an algorithm for computing the chordality of any graph admitting a k-good tree decomposition for some integer $k \geq 3$. We start from computing the chordality of the k-super-caterpillar, the subgraph induced by each bag of a k-good tree decomposition. In a small class of graphs, which are planar and k-super-caterpillars, we conjecture that it is NP-complete to compute its chordality when k is part of the input. We only present a dynamic programming algorithm for computing the chordality in a special subclass of planar k-super-caterpillar, where the vertices except the dominating path induce a cycle in the graph. Moreover, we also show that it is NP-complete to decide whether a given planar graph is a k-super-caterpillar when k is part of the input.

Part II Routing Problems. In this part, we study some routing problems. Some of them are solved by using tree decompositions. It consists of three chapters:

Chapter 4: Compact Routing Scheme for k-Good Tree Decomposable Graphs. In this chapter, we present a *compact rouging scheme* for graphs admitting a k-good tree decomposition. The objective of *compact routing problem* is to provide an algorithm for finding a path from a sender vertex to a known destination. *Compact routing scheme* takes routing decisions for a packet at every step using only very limited information stored at each vertex. In our routing model, each vertex is assigned a *name*, which distinguishes the vertex from others; also, each vertex has a *routing table* which stores the information for routing. Every message has a *header*, which contains its destinations and some informations may be modified during the transmission. Our scheme has routing tables, names and headers of $O(k\log\Delta+\log n)$ bit. When k is small it improves the previous ones with routing tables $O(\log^2 n)$ [Dou05] and $O(\Delta\log n)$ [NRS12]. Roughly, the scheme consists of two steps: first following the paths in a BFS-tree of the graph according to the scheme in [FG01]; and secondly using one bag of the tree-decomposition as a short-cut between two branches of the BFS-tree. Finally, we prove that our scheme has *additive stretch* at most $O(k\log\Delta)$. That is to say that the maximum difference of the length of the path found by our scheme between any two vertices and their distance in the graph is at most $O(k\log\Delta)$. This result has appeared in the proceedings of the conference ICALP 2012 [c-KLNS12] and in the journal Algorithmica [j-KLNS14].

Chapter 5: Prize Collecting Steiner Tree Problem on Serial Parallel Graphs. In this chapter, we study the *prize collecting Steiner tree problem (PCST)*. In this problem, given a graph $G=(V,E)$ and a *target set* $V'\subseteq V$, each edge has a nonnegative cost and each vertex has a nonnegative prize. The objective is to find a tree T in G containing all the target vertices in V' such that the *cost of* T (the sum of costs of all edges in T) minus the *prize of* T (the sum of the prizes of all vertices in T) is minimum. PCST problem is a kind of routing problem in the sense that it finds paths connecting all the target vertices. It is a generalization of the classical Steiner tree problem, in which each vertex has prize

0. The Steiner tree is NP-complete in general graphs [Kar72], so is the PCST problem. In [WC83], there is a linear time algorithm for Steiner tree problem in the class of graphs of treewidth at most 2. Generalizing their algorithm, we give a linear-time algorithm for the PCST problem in the class of graphs of treewidth at most 2 using dynamic programming.

In real life, there is always some uncertainty, e.g., in the costs of some connections and the prizes from some clients, but generally we can estimate its *range*, an interval from a lower bound to an upper bound. So we consider the PCST problem, in which costs and prizes are not only one number but belong to given intervals. Extending the two risk models in [CHH09, Hu10] for this problem, we establish the *min-max risk model* and *min-sum risk model* for the PCST problem with interval data, and propose two polynomial-time algorithms for this problem in the class of graphs of treewidth at most 2 under these two models. Finally, we describe an example in which the optimal values of these two risk models are very different. This result has appeared in the proceedings of the conference AAIM 2010 [c-AMCC$^+$10] and in the journal Acta Mathematicae Applicatae Sinica [j-AMCC$^+$14].

Chapter 6: Data Gathering and Personalized Broadcasting in Radio Grids. This chapter focuses on a type of routing problem, in which all messages has the same destination, the so called *data gathering problem* or simply *gathering problem*. In the *gathering* problem, a particular vertex in a graph, the *base station* (BS), aims at receiving messages from some vertices in the graph. At each step, a vertex can send one message to one of its neighbor (such an action is called a *call*). However, a vertex cannot send and receive a message during the same step. Moreover, the communication is subject to interference constraints. More precisely, two calls interfere in a step, if one sender is at distance at most $d_I \geq 0$ from the other caller. Given a graph with a base station and a set of vertices having some messages, the goal of the gathering problem is to compute a schedule of calls for the base station to receive all messages as fast as possible, i.e., minimizing the number of steps (called *makespan*). The gathering problem is equivalent to the *personalized broadcasting* problem where the base station has to send messages to some vertices in the graph, with same transmission constraints. We focus on the personalized broadcasting problem (and so the equivalent gathering problem) in grid networks, which model well many real life networks [KLNP09]. We presented linear (in the number of messages) time algorithms that compute schedules for the problem with $d_I \in \{0,1,2\}$.

We first study the basic instance consisting of an *open grid* where no messages have destination on an axis, with BS in the corner of the grid and with $d_I = 0$. We give a simple lower bound LB. Then we design for this basic instance a linear time algorithm with a makespan at most $LB+2$ steps, so obtaining a $+2$-approximation algorithm for the open grid, which improves the multiplicative 1.5 approximation algorithm of [RS07]. Moreover, we refine this algorithm to obtain for the basic instance a $+1$-approximation algorithm. Then we prove that the $+2$-approximation algorithm works also for a general grid where messages can have destinations on the axis again with BS in the corner and $d_I = 0$. For the cases $d_I = 1$ and 2, we give lower bounds $LB_c(1)$ (when BS is in the corner) and $LB(2)$ and use the $+1$-approximation algorithm to design algorithms with a makespan at most

$LB_c(1)+3$ when $d_I = 1$ and BS is in the corner , and at most $LB(2)+4$ when $d_I = 2$. Finally, we extend our results to the case where BS is in a general position in the grid. This work is submitted to the journal Theoretical Computer Science [s-BLN+13].

Conclusion and Perspective. Finally, we conclude the thesis and give some perspectives in the last chapter.

Part I

Tree Decomposition

CHAPTER 2

Minimum Size Tree Decomposition

Contents

In this chapter, we consider the problem of computing a tree-decomposition of a graph with width at most k and minimum *size*, i.e. the number of bags in the tree decomposition. More precisely, we focus on the following problem: given a fixed $k \geq 1$, what is the complexity of computing a tree-decomposition of width at most k with minimum size in the class of graphs with treewidth at most k? The results of this chapter is a collaboration with N. Nisse and F. Moataz. They were presented in the conference ICGT 2014 [c-LMN14].

2.1 Introduction

Sanders showed that there are practical algorithms for computing tree-decompositions of graphs with treewidth at most 4 in [San96]. Based on these tree decompositions of small widths, many NP-hard problem can be solved efficiently by dynamic programming in graphs of treewidth at most 4. The time-complexity of the dynamic programming algorithms is linear in the size of the tree-decompositions. Therefore, it is interesting to minimize it. Obviously, if the width is not constrained, then the problem is trivial since there always exists a tree-decomposition of a graph with one bag (the full vertex-set). Hence, given a graph G and an integer $k \geq tw(G)$, we consider the problem of minimizing the size of a tree-decomposition of G with width at most k.

Let k be any positive integer and G be any graph. If $tw(G) > k$, let us set $s_k(G) = \infty$. Otherwise, let $s_k(G)$ denote the minimum size of a tree-decomposition of G with width at most k. See a simple example in Fig. 2.1. We first prove in Section 2.2 that, for any (fixed) $k \geq 4$, the problem of computing s_k is NP-hard in the class of graphs with treewidth at most

k. Moreover, the computation of s_k for $k \geq 5$ is NP-hard in the class of connected graphs with treewidth at most k. In Section 2.3, we present a general approach for computing s_k for any $k \geq 1$. In the rest of the chapter, we prove that computing s_2 can be solved in polynomial-time. Finally, we prove that s_3 can be computed in polynomial time in the class of trees and 2-connected outerplanar graphs.

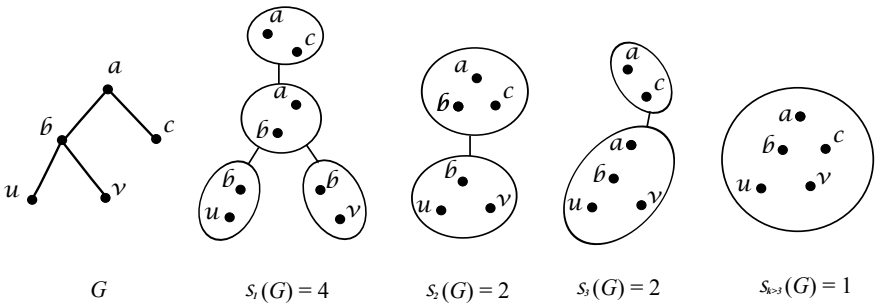


Figure 2.1: Given a tree G with five vertices, for any $k \geq 1$, a minimum size tree decomposition of width at most k is shown. So we see that $s_1(G) = 4$, $s_2(G) = s_3(G) = 2$, $s_{k>3}(G) = 1$.

Related Work. In [DKZ13], Dereniowski *et al.* consider the problem of size-constrained path-decompositions. Given any positive integer k and any graph G with pathwidth at most k. Let $l_k(G)$ denote the smallest size of a path-decomposition of G with width at most k. For any fixed $k \geq 4$, computing l_k is NP-complete in the class of general graphs and it is NP-complete, for any fixed $k \geq 5$, in the class of connected graphs [DKZ13]. Moreover, computing l_k can be solved in polynomial-time in the class of graphs with pathwidth at most k for any $k \leq 3$. Finally, the "dual" problem is also hard: for any fixed $s \geq 2$, it is NP-complete in general graphs to compute the minimum width of a tree-decomposition with size at most s. Note that this result was proved in [DKZ13] in terms of path-decomposition but it is straightforward to extend it to tree-decomposition.

2.2 NP-hardness in the class of bounded treewidth graphs

In this section, we prove that:

Theorem 4. *For any fixed integer $k \geq 4$ (resp., $k \geq 5$), the problem of computing s_k is NP-complete in the class of graphs (resp., of connected graphs) with treewidth at most k.*

Note that the corresponding decision problem is clearly in NP. Hence, we only need to prove it is NP-hard.

Our proof mainly follows the one of [DKZ13] for size-constrained path-decompositions. Hence, we recall here the two steps of the proof in [DKZ13]. First, it is proved that, if computing l_k is NP-hard for any $k \geq 1$ in general graphs, then the computation of l_{k+1} is NP-hard in the class of connected graphs. Second, it is shown that computing l_4 is NP-hard in general graphs with pathwidth 4. In particular, it implies that computing l_5 is NP-hard in the class of connected graphs with pathwidth 5. The second

step consists of a reduction from the 3-PARTITION problem [GJ79] to the one of computing l_4. Precisely, for any instance $\mathcal{I}$ of 3-PARTITION, a graph $G_{\mathcal{I}}$ is built such that $\mathcal{I}$ is a YES instance if and only if $l_4(G_{\mathcal{I}})$ equals some defined value $\ell_{\mathcal{I}}$.

Our contribution consists first in showing that the first step of [DKZ13] directly extends to the case of tree-decompositions. That is, it directly implies that, if computing s_k is NP-hard for some $k \geq 4$ in general graphs, then so is the computation of s_{k+1} in the class of connected graphs. Our main contribution of this section is to show that, for the graphs $G_{\mathcal{I}}$ built in the reduction proposed in [DKZ13], any tree-decomposition of $G_{\mathcal{I}}$ with width at most 4 and minimum size is a path decomposition. Hence, in this class of graphs, $l_4 = s_4$ and, for any instance $\mathcal{I}$ of 3-PARTITION, $\mathcal{I}$ is a YES instance if and only if $s_4(G_{\mathcal{I}})$ equals some defined value $\ell_{\mathcal{I}}$. We describe the details as follows.

Lemma 1. *If the problem of computing s_k for an integer $k \geq 1$ is NP-complete in general graphs, then the computation of s_{k+1} is NP-complete in the class of connected graphs.*

Proof. Let G be any graph. We construct an auxiliary connected graph G' from G by adding a vertex a adjacent to all vertices in $V(G)$. Given two integers $k, s \geq 1$, in the following, we prove that there is a tree decomposition of G with width at most k and size at most s if and only if there is a tree decomposition of G' with width at most $k+1$ and size at most s.

First, assume that $(T, \mathcal{X})$ is a tree decomposition of G with width at most k and size at most s. Add a in each bag of $\mathcal{X}$. Then we get a tree decomposition of G' with width at most $k+1$ and size at most s.

Now let $(T', \mathcal{X}')$ be a tree decomposition of G' with width at most $k+1$ and size at most s. We are going to find a tree decomposition of G with width at most k and size at most s.

Let $\mathcal{X}_a$ be the set of all bags in $\mathcal{X}'$ containing a. Let T_a be the subtree of T' induced by the bags in $\mathcal{X}_a$. Every vertex $v \in V(G)$ is contained in a bag in $\mathcal{X}_a$ because $va \in E(G')$. For any edge $uv \in E(G)$, there is a bag $X \supseteq \{a, u, v\}$ in $\mathcal{X}'$ since $\{a, u, v\}$ induces a clique in G'. So $X \in \mathcal{X}_a$. Delete a in each bag of $\mathcal{X}_a$ and denote $\mathcal{X}^-$ as the obtained set of bags. So $(T_a, \mathcal{X}^-)$ is a tree decomposition of G with width at most k and size at most s. □

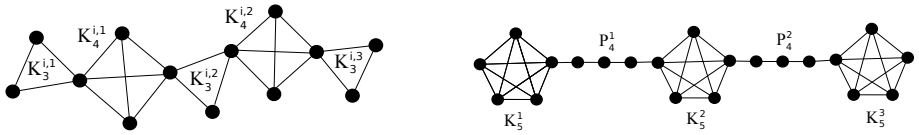
Before doing the reduction from the 3-PARTITION problem to the problem of computing s_4, let us first recall its definition.

Definition 2. [3-PARTITION]
Instance: A multiset S of $3m$ positive integers $S = (w_1, \ldots, w_{3m})$ and an integer b.
Question: Is there a partition of the set $\{1, \ldots, 3m\}$ into m sets $S_1, \ldots, S_m$ such that $\sum_{i \in S_j} w_i = b$ for each $j = 1, \ldots, m$?

This problem is NP-complete even if $|S_j| = 3$ for all $j = 1, \ldots, m$ [GJ79].

Given an instance of 3-PARTITION, in the following, we construct a disconnected graph $G(S, b)$ as in [DKZ13].

First, for each $i \in \{1, \ldots, 3m\}$, we construct a connected graph H_i as follows. Take w_i copies of K_3, denoted by $K_3^{i,q}$, $q = 1, \ldots, w_i$, and $w_i - 1$ copies of K_4, denoted by $K_4^{i,q}$, $q = 1, \ldots, w_i - 1$ (the copies are mutually disjoint). Then for each $q = 1, \ldots, w_i - 1$, we identify two different vertices of $K_4^{i,q}$ with a vertex of $K_3^{i,q}$ and with a vertex of $K_3^{i,q+1}$, respectively. This is done in such a way that each vertex of each $K_3^{i,q}$ is identified with



(a) H_i, where $w_i = 3$. (b) $H_{2,4}$

Figure 2.2: Examples of gadgets in graph $G(S,b)$.

at most one vertex from other cliques. Informally the cliques form a 'chain' in which the cliques of size 3 and 4 alternate. See Figure 2.2(a) for an example of H_i where $w_i = 3$.

Second, we construct a graph $H_{m,b}$ as follows. Take $m+1$ copies of K_5, denoted by $K_5^1,\ldots,K_5^{m+1}$, and m copies of the path graph P_b of length b (P_b has b edges and $b+1$ vertices), denoted by $P_b^1,\ldots,P_b^m$. (Again, the copies are taken to be mutually disjoint.) Now, for each $j = 1,\ldots,m$, identify one of the endpoints of P_b^j with a vertex of K_5^j, and identify the other endpoint with a vertex of K_5^{j+1}. Moreover, do this in a way that ensures that, for each j, no vertex of K_5^j is identified with the endpoints of two different paths. See Figure 2.2(b) for an example of $H_{2,4}$. This implies that the computation of s_4 is NP-hard.

Let $G(S,b)$ be the graph obtained by taking the disjoint union of the graphs $H_1,\ldots,H_{3m}$ and the graph $H_{m,b}$. In the following, we prove that there is a tree decomposition of $G(S,b)$ of width 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$ if and only if there is a partition of the set $\{1,\ldots,3m\}$ into m sets $S_1,\ldots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\ldots,m$ in the instance of 3-PARTITION.

In Lemma 2.2 of [DKZ13], they constructed a path decomposition of $G(S,b)$ of width 4 and length $1-2m+2\sum_{i=1}^{3m} w_i$ if there is a partition of the set $\{1,\ldots,3m\}$ into m sets $S_1,\ldots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\ldots,m$ in the instance of 3-PARTITION. Obviously, this path decomposition is also a tree decomposition of $G(S,b)$ of width 4 and size s. So we have the following lemma.

Lemma 2. *Given a multiset S of $3m$ positive integers $S = (w_1,\ldots,w_{3m})$ and an integer b, if there is a partition of the set $\{1,\ldots,3m\}$ into m sets $S_1,\ldots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\ldots,m$, then $G(S,b)$ has a tree decomposition of width at most 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$.*

Now we prove the other direction.

Lemma 3. *If $G(S,b)$ has a tree decomposition $(T,\mathcal{X})$ of width at most 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$, then there is a partition of the set $\{1,\ldots,3m\}$ into m sets $S_1,\ldots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\ldots,m$.*

Proof. Lemma 2.6 in [DKZ13] proved that if $G(S,b)$ has a path decomposition $(T,\mathcal{X})$ of width at most 4 and length at most $1-2m+2\sum_{i=1}^{3m} w_i$, then there is a partition of the set $\{1,\ldots,3m\}$ into m sets $S_1,\ldots,S_m$ such that $\sum_{i\in S_j} w_i = b$ for each $j = 1,\ldots,m$. So it is enough to prove that any tree decomposition (T,X) of $G(S,b)$ of width at most 4 and size at most $s = 1-2m+2\sum_{i=1}^{3m} w_i$ is a path decomposition of $G(S,b)$.

As proved in Lemma 2.3 of [DKZ13], each bag in (T,X) contains exactly one of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$. Indeed, each of these cliques has size at least 3. Moreover, any two

of them share at most one vertex, and no two cliques of size 3 ($K_3^{i,q}$) share a vertex. So each bag of $(T,\mathcal{X})$ contains at most one of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$. However, for any clique, there is a bag in $(T,\mathcal{X})$ containing its vertices. Since s equals the number of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$, each bag of $(T,\mathcal{X})$ contains exactly one of them.

Moreover let us prove that any edge in $K_4^{i,q}, K_5^j, P_b^j$ (i.e. both the two endpoints of the edge) are contained in exactly one bag. Since each bag in (T,X) contains exactly one of the cliques $K_3^{i,q}, K_4^{i,q}, K_5^j$, the two endpoints of any edge in the paths $P_b^1,\ldots,P_b^m$ are contained in a bag containing some $K_3^{i,q}$. (The bags containing a $K_4^{i,q}$ (resp. K_5^j) cannot add another two vertices (one vertex) since $(T,\mathcal{X})$ is a tree decomposition of width at most 4.) Every bag containing some $K_3^{i,q}$ contains at most one edge in the paths $P_b^1,\ldots,P_b^m$, because the bag can add at most another two vertices and any $K_3^{i,q}$ and P_b^j are disjoint. There are mb edges in the paths $P_b^1,\ldots,P_b^m$ and there are mb bags containing some $K_3^{i,q}$, so every bag containing a $K_3^{i,q}$ contains exactly one edge in the paths $P_b^1,\ldots,P_b^m$. So any edge in the paths $P_b^1,\ldots,P_b^m$ is contained in exactly one bag. Also each bag containing some $K_3^{i,q}$ contains 5 vertices, so it does not contains any edge (i.e. both its two endpoints) in $K_4^{i,q}$ or K_5^j. Therefore, any edge on $K_4^{i,q}, K_5^j$ are contained in exactly one bag.

Now we prove that there are only two leaves in T and so T is a path. If a bag containing some $K_3^{i,q}$ and an edge uv on some path P_b^j is a leaf bag in T, then its neighbor bag also contains u,v because both u and v are incident to other edges in $G(S,b)$. This is a contradiction with any edge (its two endpoints) on P_b^j are contained only in one bag. So any bag containing some $K_3^{i,q}$ is not a leaf bag in T. Similarly, we can prove that any bag containing any $K_4^{i,q}$ or K_5^j for $1 < j < m+1$ is not a leaf bag in T. Thus there are only two bags containing K_5^1 and K_5^{m+1} are leaves in T. □

Then we get the following corollary.

Corollary 1. *It is NP-complete to compute s_4 in the class of graphs of treewidth at most 4.*

Theorem 4 follows from Lemma 1 and Corollary 1.

2.3 Notations and preliminaries

In this section, we present the definitions and notations used throughout the chapter and some well known facts about tree-decompositions.

2.3.1 Notations

Given a graph $G=(V,E)$, for any $S \subseteq V$, For an integer $c \geq 0$, a graph $G=(V,E)$ is *c-connected* if $|V| > c$ and no subset $V' \subseteq V$ with $|V'| < c$ is a separator in G. A *2-connected component* of G is a maximal 2-connected subgraph.

Let $(T,\mathcal{X})$ be any tree-decomposition of G. Abusing the notations, we will identify a node $t \in V(T)$ and its corresponding bag $X_t \in \mathcal{X}$. This means that, e.g., instead of saying $t \in V(T)$ is adjacent to $t' \in V(T)$ in T, we can also say that $X_t \in \mathcal{X}$ is adjacent to $X_{t'} \in \mathcal{X}$ in T. A bag $B \in \mathcal{X}$ is called a *leaf-bag* if B has degree one in T. Let $k \geq 1$ and G be a graph with $tw(G) \leq k$. A subset $B \subseteq V(G)$ is a *k-potential-leaf* if there is a tree-decomposition $(T,\mathcal{X})$

with width at most k and size $s_k(G)$ such that B is a leaf bag of $(T,\mathcal{X})$. A subgraph $H \subseteq V$ is a *k-potential-leaf* of G if $V(H)$ is a k-potential-leaf of G. Note that a k-potential-leaf has size at most $k+1$.

A tree-decomposition is *reduced* if no bag is contained in another one. It is straightforward that, in any leaf-bag B of reduced tree-decomposition, there is $v \in V$ such that v appears only in B and so $N[v] \subseteq B$. Note that it implies that any reduced tree-decomposition has at most $n-1$ bags.

In the following we define two *transformation rules*, that takes a tree-decomposition $(T,\mathcal{X})$ of a graph G, and computes another one without increasing the width nor the size.

Leaf. Let $X \in \mathcal{X}$ and $N_T(X) = \{X_1, \cdots, X_d\}$. Assume that, for any $1 < i \leq d$, $X_i \cap X \subseteq X_1$. Let $(T^*,\mathcal{X}^*) = Leaf(X, X_1, (T,\mathcal{X}))$ denote the tree-decomposition of G obtained by replacing each edge $X_iX \in E(T)$ by an edge X_iX_1 for any $1 < i \leq d$. Note that X becomes a leaf-bag after the operation. See in Fig. 2.3.

Reduce. Let $XX' \in E(T)$ with $X \subseteq X'$. Let $(T^*,\mathcal{X}^*) = Reduce(X, X', (T,\mathcal{X}))$ denote the tree-decomposition of G obtained by deleting the bag X from the tree-decomposition $Leaf(X, X', (T,\mathcal{X}))$. Note that the size of the tree decomposition is decreased by one after the operation.

From any tree-decomposition of G with width k and size s, it is easy to obtain a reduced tree-decomposition of G with width at most k and size at most $s-1$ by applying the Reduce operation while it is possible (i.e., while a bag is contained in another one). In particular, any minimum size tree decomposition is reduced.

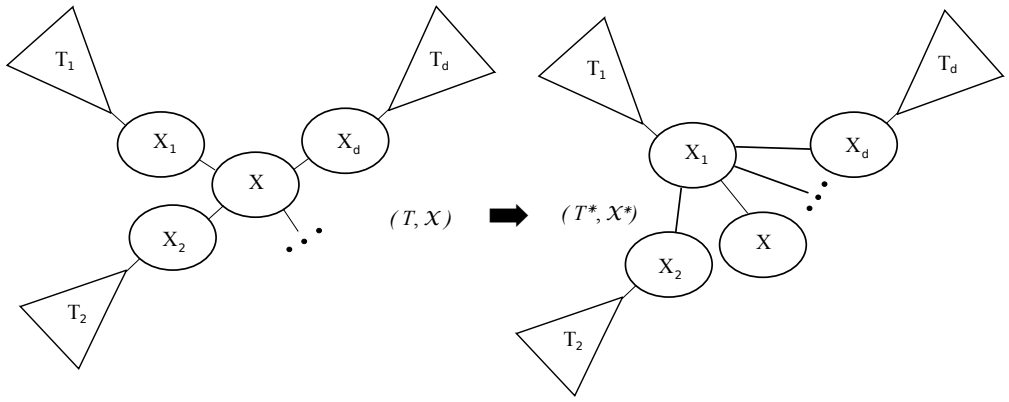


Figure 2.3: In a tree decomposition $(T,\mathcal{X})$, $N_T(X) = \{X_1, \cdots, X_d\}$ and for any $1 < i \leq d$, $X_i \cap X \subseteq X_1$. For $1 \leq i \leq d$, $T_i \cup X_i$ induces the subtree containing X_i in $T \setminus \{XX_i\}$. Replace each edge $X_iX \in E(T)$ by an edge X_iX_1 for any $1 < i \leq d$. This gives a tree decomposition $(T^*,\mathcal{X}^*) = Leaf(X, X_1, (T,\mathcal{X}))$. X is a leaf-bag in $(T^*,\mathcal{X}^*)$.

We conclude this section by a general lemma on tree-decompositions. This lemma is known as folklore, we recall it for completness.

Lemma 4. *Let $(T,\mathcal{X})$ be a tree decomposition of a graph G. Let $X \in \mathcal{X}$ and $v,w \in X$. If there exists a connected component in $G \setminus X$ containing a neighbor of v and a neighbor of w, then there is a neighbor bag of X in $(T,\mathcal{X})$ containing v and w.*

Proof. First, let us note that, for any connected subgraph H of G, the set of bags of T that contain a vertex of H induces a subtree of T (the proof is easy by induction on $|V(H)|$).

Let C be a connected component in $G \setminus X$ containing a neighbor of v and a neighbor of w. By above remark, let T'_C be the subtree of T induced by the bags that contain some vertex of C. Moreover, because no vertices of C are contained in the bag X, then T'_C is a subtree of $T \setminus X$. Let T_C be the connected component of $T \setminus X$ that contains T'_C. Let $Y \in V(T_C)$ be the bag of T_C which is a neighbor of X in T. Let $x \in N(v) \cap C$ be a neighbor of v in C. Then there exists a bag $Z \in \mathcal{X}$ in T_C containing both x and v. So both X and Z contain vertex v. Then the bag Y, which is on the path between X and Z in T, also contains v. Similarly, we can prove that $w \in Y$. □

Corollary 2. *Let $(T,\mathcal{X})$ be a tree decomposition of a 2-connected graph G. Let $X \in \mathcal{X}$ and $|X| \leq 2$. Then there is a neighbor bag Y of X in $(T,\mathcal{X})$ such that $X \subseteq Y$.*

Proof. Since G is 2-connected, $|V(G)| \geq 3$. So there exist at least another bag except X in $\mathcal{X}$.

If $|X| = 1$, let $X = \{v\}$. Then there is a neighbor bag Y of X containing v, since G is 2-connected and v is adjacent to some vertices in G. So $X \subseteq Y$.

Otherwise $X = 2$ and let $X = \{v,w\}$. Let G_1 be any connected component in $G \setminus X$. If v is not adjacent to any vertex in G_1, then $\{w\}$ separates $V(G_1)$ from $\{v\}$. It contradicts with the assumption that G is 2-connected. So any connected component in $G \setminus X$ containing a neighbor of v and a neighbor of w. From Lemma 4, there is a neighbor bag Y of X containing v,w, i.e. $X \subseteq Y$. □

2.3.2 General approach

In what follows, we propose polynomial-time algorithms to compute minimum-size tree-decompositions of graphs with small treewidth. Our algorithms mainly use the notion of potential leaf.

Let $k \geq 1$ and $G = (V,E)$ be a graph with $tw(G) \leq k$. The key idea of our algorithms is to identify a finite number of subgraphs that may be potential-leaves (abusing the notations, we will identify a subgraph and its vertex-set). Then, given a graph G and a k-potential-leaf H, we will be able to compute a minimum-size tree-decomposition of G by adding H to a minimum-size tree-decomposition of a smaller graph. So, our algorithms proceed by induction on the number of nodes.

The next lemmas formalize the above paragraph. Given a graph $G = (V,E)$ and a set $S \subseteq V$, let $G_S = G \cup \{uv : u,v \in S\}$.

Lemma 5. *Let $k \geq 1$ and $G = (V,E)$ be a graph with $tw(G) \leq k$. Let $B \subseteq V$ be a k-potential-leaf of G. Let $S \subset B$ be the set of vertices of B that have a neighbor in $V \setminus B$. Then $s_k(G) = s_k(G_S \setminus (B \setminus S)) + 1$.*

Proof. Let us first prove $s_k(G) \leq s_k(G_S \setminus (B \setminus S)) + 1$. Suppose that $(T_S, \mathcal{X}_S)$ is a minimum size tree decomposition of width at most k of the graph $G_S \setminus (B \setminus S)$. Then there exists a bag $X \in \mathcal{X}_S$ containing S because S induces a clique in the graph $G_S \setminus (B \setminus S)$. So add the bag B and make it adjacent to X in the tree decomposition $(T_S, \mathcal{X}_S)$. Then we obtain a tree decomposition of width at most k for graph G of size $s_k(G_S \setminus (B \setminus S)) + 1$.

Now we prove that $s_k(G) \geq s_k(G_S \setminus (B \setminus S)) + 1$. Let $(T, \mathcal{X})$ be a minimum size tree decomposition of G of width at most k such that B is a leaf bag in it. Note that, if $B = V$ then $G_S \setminus (B \setminus S)$ is the empty graph. Let us assume that $B \subset V$. Then $(T, \mathcal{X})$ is also a tree decomposition of G_S. Let B be adjacent to the bag Y in $(T, \mathcal{X})$. Then $S \subset Y$ since each vertex in S is contained in another bag in $(T, \mathcal{X})$. Let $(T', \mathcal{X}')$ be the tree decomposition obtained by deleting the vertices in $B \setminus S$ in all the bags of $(T, \mathcal{X})$. Then B is changed to $B' = S \in \mathcal{X}'$ and let Y be changed to $Y' \in \mathcal{X}'$. So $B' \subseteq Y'$. Then the tree decomposition $Reduce(B', Y', (T', \mathcal{X}'))$ is a tree decomposition of $G_S \setminus (B \setminus S)$ of size $s_k(G) - 1$. So $s_k(G) - 1 \geq s_k(G_S \setminus (B \setminus S))$. □

This lemma implies the following corollary:

Corollary 3. *Let $k \in \mathbb{N}^*$ and $\mathcal{C}$ be the class of graphs with treewidth at most k. If there is a $g(n)$-time algorithm $\mathcal{A}_k$ that, for any n-vertex-graph $G \in \mathcal{C}$, computes a k-potential leaf of G. Then s_k can be computed in $O(g(n) \cdot n)$ time in the class of n-vertex graphs in $\mathcal{C}$. Moreover, a minimum size tree decomposition of width at most k can be constructed in the same time.*

Proof. Let $G \in \mathcal{C}$ be a n-vertex-graph. Let us apply Algorithm $\mathcal{A}_k$ to find a subgraph H of G in $g(n)$ time, which is a k-potential-leaf of G. Let $S \subset V(H)$ be the set of vertices having a neighbor in $G \setminus H$ and $G' = G_S \setminus (V(H) \setminus S)$. Then, by Lemma 5, $s_k(G) = s_k(G') + 1$. Finally, $|V(G')| \leq n - 1$ and G' has treewidth at most k. We then proceed recursively. So the total time complexity is $O(g(n) \cdot n)$. Moreover, for any minimum size $(s_k(G'))$ tree decomposition $(T', \mathcal{X}')$ of G' of width k, there is a bag X containing S since S induces a clique in G'. Add a new bag $N = V(H)$ adjacent to X in $(T', \mathcal{X}')$. The obtained tree decomposition is a minimum size $(s_k(G) = s_k(G') + 1)$ tree decomposition of G of width at most k. □

2.4 Graphs with treewidth at most 2

In this section, we describe algorithm $\mathcal{A}_2$ computes a 2-potential leaf of a given graph. In particular, all graphs considered in this section have treewidth at most 2, i.e. partial 2-trees. Please see all the 2-potential leaf of graphs of treewidth at most 2 in Fig. 2.4.

Lemma 6. *Let G be a graph with treewidth at most 2 and $p \in V(G)$ such that $N(p) = \{f, q\}$ and f has degree one (see in Fig. 2.4(a)). Then $\{f, p, q\}$ is a 2-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree-decomposition of G with width at most 2 and size at most $s \geq 1$. We show how to modify it to obtain a tree-decomposition with width at most 2 and size at most s and in which $\{f, p, q\}$ is a leaf bag.

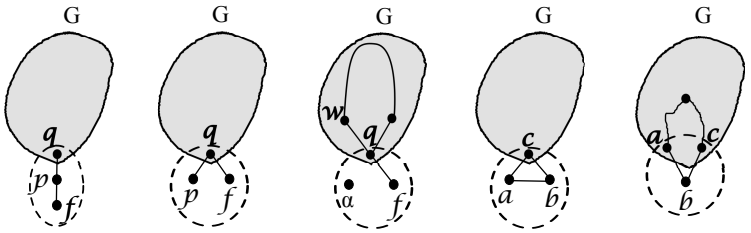


Figure 2.4: All the 2-potential leaves of graphs of treewidth at most 2.

Since $fp \in E(G)$, there is a bag B in $(T,\mathcal{X})$ containing both f and p. We may assume that B is the single bag containing f (otherwise, delete f from any other bag). Similarly, since $pq \in E(G)$, let X be a bag in $(T,\mathcal{X})$ containing both p and q.

First, let us assume that $X = B = \{f,p,q\}$. In that case, we may assume that X is the single bag containing p (otherwise, delete p from any other bag). If X is a leaf bag, then the lemma is proved. Otherwise, let $X_1,\cdots,X_d$ be the neighbors of X in T. Since f and p appear only in X, then $X \cap X_i \subseteq \{q\}$ for any $1 \leq i \leq d$. If there is $1 \leq i \leq d$ such that $q \in X_i$, let us assume w.l.o.g., that $q \in X_1$. By definition of the operation $Leaf$, the tree-decomposition $Leaf(X,X_1,(T,\mathcal{X}))$ has width at most 2, same size as $(T,\mathcal{X})$, and X is a leaf.

Second, consider the case when $X \neq B$. There are two cases to be considered. Either $B = \{f,p\}$ or $B = \{f,p,x\}$ with $x \neq q$. In the latter case, note that there is another bag B', neighbor of B, that contains x unless x is an isolated vertex of G. In the former case or if x appears only in B (in which case, x is an isolated vertex), let B' be any neighbor of B. Let $(T',\mathcal{X}')$ be obtained by deleting f,p in all bags of $(T,\mathcal{X})$. Then, contract the edge BB' in T', i.e., remove B and make any neighbor of B adjacent to B'. Note that, in the resulting tree-decomposition of $G \setminus \{f,p\}$, there is a bag X' containing q and with $|X'| \leq 2$ (the bag that results from X). Finally, add a bag $\{f,p,q\}$ adjacent to X' and, if node x was only in B, then add x to X'. The result is the desired tree-decomposition. □

Lemma 7. *Let G be a graph with treewidth at most 2 and $q \in V(G)$ such that q has at least two one-degree neighbors f and p (see in Fig. 2.4(b)). Then $\{f,p,q\}$ is a 2-potential-leaf of G.*

Proof. Let $(T,\mathcal{X})$ be any tree-decomposition of G with width at most 2 and size at most $s \geq 1$. We show how to modify it to obtain a tree-decomposition with width at most 2 and size at most s and in which $\{f,p,q\}$ is a leaf bag.

Since $fq \in E(G)$, there is a bag B in $(T,\mathcal{X})$ containing both f and q. We may assume that B is the single bag containing f (otherwise, delete f from any other bag). Similarly, since $pq \in E(G)$, let X be a bag in $(T,\mathcal{X})$ containing both p and q. Again, we may assume that X is the single bag containing p (otherwise, delete p from any other bag).

First, let us assume that $X = B = \{f,p,q\}$. If X is a leaf bag, then the lemma is proved. Otherwise, let $X_1,\cdots,X_d$ be the neighbors of X in T. Since f and p appear only in X, then $X \cap X_i \subseteq \{q\}$ for any $1 \leq i \leq d$. If there is $1 \leq i \leq d$ such that $q \in X_i$, let us

assume w.l.o.g., that $q \in X_1$. By definition of the operation $Leaf$, the tree-decomposition $Leaf(X, X_1, (T, \mathcal{X}))$ has width at most 2, same size as $(T, \mathcal{X})$, and X is a leaf.

Second, let us assume that $X = \{f, q\}$ or $B = \{p, q\}$. In the former case, remove p from any bag and add p to X. In the latter case, remove f from any bag and add f to B. In both cases, we get a bag $\{f, p, q\}$ as in the first case.

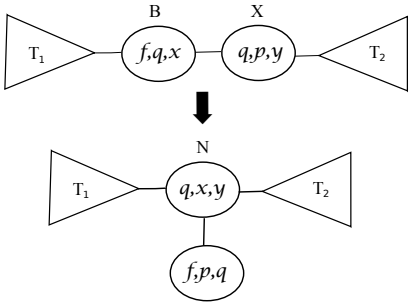
Otherwise, let $B = \{f, q, x\}$, $x \neq p$, and $X = \{p, q, y\}$, $y \neq f$.

- If B and X are adjacent in T, then add a new bag $N = \{q, x, y\}$; remove B and X and make each of their neighbors adjacent to the new bag N and, finally, add a leaf-bag $\{f, p, q\}$ adjacent to N. See in Fig. 2.5(a). The obtained tree-decomposition has the desired properties.

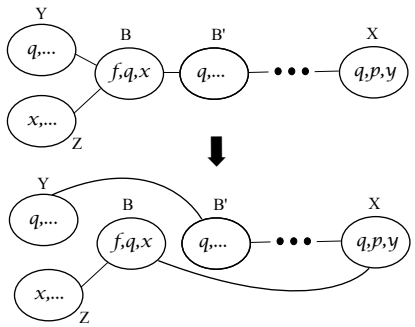
- Otherwise, if there is a neighbor B' of B with $q, x \in B'$, then remove B, make all neighbors of B adjacent to B' and finally add a leaf-bag $\{f, p, q\}$ adjacent to X. The obtained tree-decomposition has the desired properties.

- Otherwise, let B' be the neighbor of B on the path between B and X. In this case, $q \in B'$ and $x \notin B'$. Moreover, q does not belong to any neighbor of B that contains x and the other way around: x does not belong to any neighbor of B that contains q. For any neighbor Y of B with $q \in Y$ (and hence $x \notin Y$), replace the edge $YB \in E(T)$ with the edge YB'. Finally, replace the edge $BB' \in E(T)$ by the edge BX. See in Fig. 2.5(b). In the resulting tree-decomposition of G, B and X are adjacent and we are back to the first item.

□



(a) In the tree decomposition $(T, \mathcal{X})$, let $T_1 \cup B$ (resp. $T_1 \cup B$) induce the subtree containing B (resp. X) in $T \setminus \{BX\}$. Delete B and X and make each of their neighbors adjacent to the new bag $N = \{q, x, y\}$; and add a leaf-bag $\{f, p, q\}$ adjacent to N.



(b) To be simple and clear, we show only the induced path from B to X in T and two neighbors $Y, Z \neq B'$ of B. Y contains q and Z contains x. Then we just change Y to be adjacent to B' instead of B and change B to be adjacent to X instead of B'.

Figure 2.5: Explanation of proof of Lemma 7.

Lemma 8. *Let G be a graph with treewidth at most 2 and $q \in V(G)$ such that q has one neighbor f with degree 1 and for any vertex $w \in N(q) \setminus \{f\}$, $\{w, q\}$ belongs to a 2-connected component of G.*

If G has an isolated vertex α, then $\{q,f,\alpha\}$ is a 2-potential-leaf; otherwise $\{q,f\}$ is a 2-potential-leaf (see in Fig. 2.4(c)).

Proof. Let $(T,\mathcal{X})$ be any tree-decomposition of G with width at most 2 and size at most $s \geq 1$. We show how to modify it to obtain a tree-decomposition with width at most 2 and size at most s and in which $\{f,q,\alpha\}$ is a leaf bag if G has an isolated vertex α; and otherwise $\{f,q\}$ is a leaf bag.

Since $fq \in E(G)$, there is a bag B in $(T,\mathcal{X})$ containing both f and q. We may assume that B is the single bag containing f (otherwise, delete f from any other bag).

1. If $B = \{f,q\}$, then the intersection of B and any of its neighbor in T is empty or $\{q\}$. If there is a neighbor of B containing q, then let X be such a neighbor; otherwise let X be any neighbor of B. By definition of the operation *Leaf*, the tree-decomposition $Leaf(B,X,(T,\mathcal{X}))$ has width at most 2, same size as $(T,\mathcal{X})$, and B is a leaf. If there are no isolated vertices, we are done. Otherwise, if there is an isolated vertex α in G, then delete α in all bags of the tree-decomposition $Leaf(B,X,(T,\mathcal{X}))$ and add α to bag B, i.e. make $B = \{f,p,\alpha\}$. The result is the desired tree decomposition.

2. Otherwise let $B = \{f,q,x\}$.

   (a) If x is a neighbor of q, then x and q are in a 2-connected component of G. So there exists a connected component in $G \setminus B$ containing a vertex adjacent to x and a vertex adjacent to q. From Lemma 4, there is a neighbor X of B in $(T,\mathcal{X})$ containing both x and q. Then by definition of the operation *Leaf*, the tree-decomposition $Leaf(B,X,(T,\mathcal{X}))$ has width at most 2, same size as $(T,\mathcal{X})$, and B is a leaf. Then delete x in B, i.e. $B = \{f,q\}$. Finally, if α is an isolated vertex of G, remove it to any other bag and add it to B. The result is the desired tree decomposition.

   (b) Otherwise x is not adjacent to q. If there is a neighbor X of B in $(T,\mathcal{X})$ containing both x and q, then $(T,\mathcal{X})$ is modified as in case 2a. Otherwise, any neighbor of B in $(T,\mathcal{X})$ contains at most one of q and x.

   If there is a neighbor of B in T containing q, then let Y be such a neighbor of B; otherwise let Y be any neighbor of B. Delete the edges between B and all its neighbors not containing x except Y in $(T,\mathcal{X})$ and make them adjacent to Y.

   If there is no neighbor of B containing x, then x is an isolated vertex and we get a tree decomposition of the same size and width as $(T,\mathcal{X})$, in which there is a leaf bag $B = \{f,q,x\}$. It is a required tree decomposition.

   Otherwise let Z be a neighbor of B in $(T,\mathcal{X})$ containing x, then delete the edges between B and all its neighbors containing x except Z in $(T,\mathcal{X})$ and make them adjacent to Z. Now B has only two neighbors Y and Z and $B \cap Y \subseteq \{q\}$, $B \cap Z = \{x\}$ and $Y \cap Z = \emptyset$. Delete the edge between B and Z and make Z adjacent to Y. Delete x in B, i.e. make $B = \{f,q\}$. See the transformations in Fig. 2.6. Then we get a tree decomposition of the same size and width as $(T,\mathcal{X})$, in which $B = \{f,q\}$ is a leaf bag. Again, if α is an isolated vertex of G, remove it to any other bag and add it to B. The result is the desired tree decomposition.

□

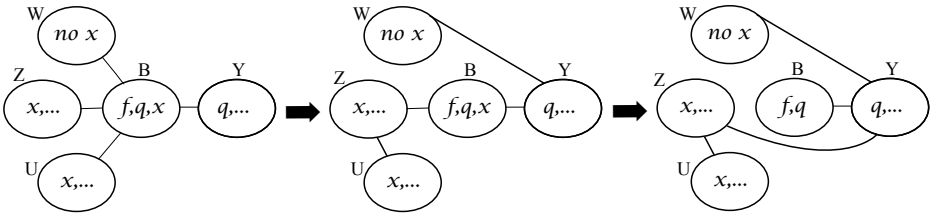


Figure 2.6: To be simple and clear, we show only the subtree induced by B,Y and another three neighbors Z,W,U of B. Y contains q; Z,U both contain x and W does not contain x. First we make the bag not containing x, e.g. W adjacent to Y instead of B; and make the bag containing x except Z, e.g. U adjacent to Z instead of B. Second, make Z adjacent to Y instead of B and delete x in B. Then $B = \{f,q\}$ is a leaf-bag.

Lemma 9. *Let G be a graph of treewidth at most 2. Let $b \in V(G)$ with $N(b) = \{a,c\}$. If $N(a) = \{b,c\}$ (see in Fig. 2.4(d)) or if there is a path, with at least one internal vertex, between a and c in $G \setminus \{b\}$ (see in Fig. 2.4(e)), then $\{a,b,c\}$ is a 2-potential-leaf of G.*

Proof. Let $G = (V,E)$ be a graph of treewidth at most 2. Let $b \in V$ with exactly 2 neighbors $a,c \in V$ satisfy the hypotheses of the lemma. If $V = \{a,b,c\}$, the result holds trivially, so let us assume that $|V| \geq 4$.

Let $(T,\mathcal{X})$ be a reduced tree decomposition of width at most 2 of G. From $(T,\mathcal{X})$, we will compute a tree decomposition $(T^*,\mathcal{X}^*)$ of G without increasing the width or the size and such that $\{a,b,c\}$ is a leaf-bag of $(T^*,\mathcal{X}^*)$.

Let X be any bag of $(T,\mathcal{X})$ containing $\{a,b\}$ and Y be any bag containing $\{b,c\}$. The bags X,Y exist because $ab,bc \in E$. If $X = \{a,b\}$, then there exists a connected component in $G \setminus X$ containing a neighbor of a and a neighbor of b. By Lemma 4, there is a neighbor of X in $(T,\mathcal{X})$ that contains both a and b, contradicting the fact that $(T,\mathcal{X})$ is reduced. So $|X| = 3$ and, similarly, $|Y| = 3$.

- Let us first assume that $X = Y = \{a,b,c\}$. In particular, it is the case when $N(a) = \{b,c\}$ since $\{a,b,c\}$ induces a clique. We may assume that b only belongs to bag X (otherwise, remove b from any other bag).

  If $N(a) = \{b,c\}$, then we can also assume that a only belongs to X. Let Z be any neighbor of B containing c if exists; otherwise let Z be any neighbor of B (Z exists since $|V| \geq 4$). Otherwise, there exists a path P between a and c in $G \setminus \{b\}$ with at least one internal vertex. In this latter case, there exists a connected component in $G \setminus X$ containing a neighbor of a and a neighbor of c. So by Lemma 4, there is a neighbor bag Z of X in $(T,\mathcal{X})$ containing both a and c. In both cases, $Leaf(X,Z,(T,\mathcal{X}))$ is the desired tree-decomposition.

- Otherwise, $X = \{a,b,x\}$ and $Y = \{b,c,y\}$ with $x \neq c$ and $y \neq a$; and there exists a path P between a and c in $G \setminus \{b\}$ with at least one internal vertex. Let Q be the path

between X and Y in $(T,\mathcal{X})$. We may assume that b only belongs to the bags in Q, because otherwise b can be removed from any other bag.

- If X is adjacent to Y, then by properties of tree-decomposition, $X \cap Y$ separates a and c. Since $\{b\}$ does not separate a and c, $X \cap Y = \{b,x\}$, i.e. $x = y$. In this case, $(T^*,\mathcal{X}^*)$ is obtained by making $X = \{a,c,x\}$ and removing Y from $(T,\mathcal{X})$, then making all neighbors of Y adjacent to X and finally, adding a bag $\{a,b,c\}$ adjacent to X.
- Otherwise, let X' be the bag in the path Q containing a, which is closest to Y. Similarly, let Y' be the bag in the path Q containing c, which is closest to X. Finally, let Q' be the path from X' to Y' in T and note that b belongs to each bag in Q' and a and c do not belong to any internal bag in Q'. Then we may assume that b only belongs to the bags in Q', because otherwise b can be removed from any other bag.

  If X' and Y' are adjacent in T, the proof is similar to the one in previous item. Otherwise, let Z be the neighbor of X' in Q'. By properties of tree-decompositions, $X' \cap Z$ separates a and c. Since $\{b\}$ does not separate a and c, let $X' \cap Z = \{b,x'\}$. Since $Z \neq \{b,x'\}$ because $(T,\mathcal{X})$ is reduced, then $Z = \{b,x',z\}$ for some $z \in V$. Replace b with a in all the bags. By doing this $(T,\mathcal{X})$ is changed to a tree decomposition $(T^c,\mathcal{X}^c)$ of the graph G/ab obtained by contracting the edge ab in G. In $(T^c,\mathcal{X}^c)$, the bag X' has become $X^c = \{a,x'\}$ and Z is changed to be $Z^c = \{a,x',z\}$. So X^c can be reduced in $(T^c,\mathcal{X}^c)$. Moreover Y is changed to $Y^c = \{a,c,y\}$. To conclude, let us add the bag $\{a,b,c\}$ adjacent to Y^c in the tree decomposition $Reduce(X^c,Z^c,(T^c,\mathcal{X}^c))$. See in Fig. 2.7. The result is the desired tree-decomposition $(T^*,\mathcal{X}^*)$ of G.

□

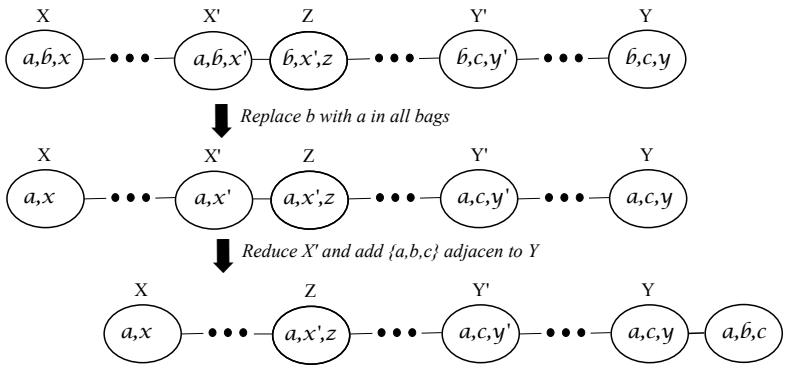


Figure 2.7: To be simple and clear, we show only the path from X to Y. After the two transformations, $\{a,b,c\}$ is a leaf-bag.

Before going further, let us introduce some notations. A *bridge* in a graph $G = (V,E)$ is any subgraph induced by two adjacent vertices u and v of G (i.e., $uv \in E$) such that the number of connected components strictly increases when deleting the edge uv, but not the

two vertices u, v in G, i.e., $G' = (V, E \setminus \{uv\})$ has strictly more connected components than G. A vertex $v \in V$ is a *cut vertex* if $\{v\}$ is a separator in G. A maximal connected subgraph without a cut vertex is called a *block*. Thus, every block of a graph $G = (V,E)$ is either a 2-connected component of G or a bridge or an isolated vertex. Conversely, every such subgraph is a block. Different blocks of G intersect in at most one vertex, which is a cut vertex of G. Hence, every edge of G lies in a unique block, and G is the union of its blocks.

Let $G = (V,E)$ be a connected graph and let $r \in V$. A spanning tree T of G is a BFS-tree of G if for any $v \in V(G)$, the distance from r to v in G is the same as the one in T. Let $\mathcal{B} = \{C : C \text{ is a block of } G\}$. The *block graph* of G is the graph $B(G)$ whose vertices are the blocks of G and two block-vertices of $B(G)$ are adjacent if the corresponding blocks intersect, that is, $B(G) = (\mathcal{B}, \{C_1C_2 : C_1, C_2 \in \mathcal{B} \text{ and } C_1 \cap C_2 \neq \emptyset\})$. Note that $B(G)$ is connected. Finally, a *block-tree* of G is any BFS-tree F (with any arbitrary root) of $B(G)$. See an example in Fig. 2.8.

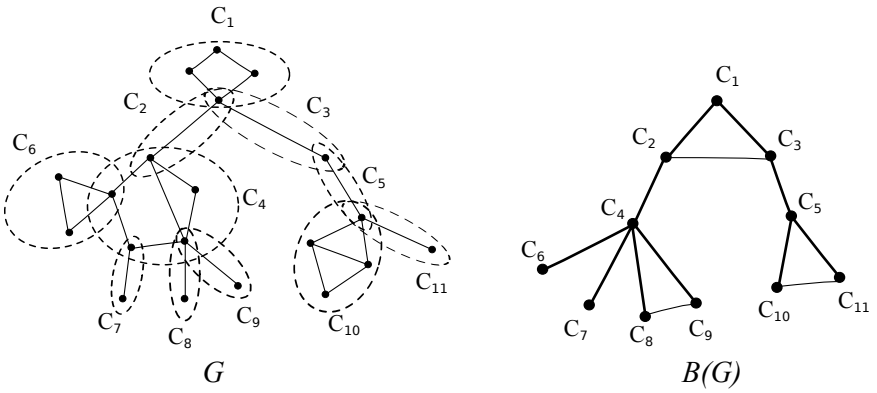


Figure 2.8: Graph G is connected. For $i = 1, \ldots, 11$, each C_i is a block of G. $B(G)$ is the block graph of G. The BFS tree of $B(G)$ with bold edges is a block tree of G with root C_1.

There is a linear (in the number of edges) algorithm for computing all blocks in a given graph [HT73]. Also a BFS-tree can be found in linear (in the number of vertices plus the number of edges) time. So given a graph $G = (V,E)$, we can compute a block tree F of G in $O(|V| + |E|)$ time.

Now we are ready to prove next theorem by using the Lemmas 6-9.

Theorem 5. *There is an algorithm that, for any n-vertex-m-edge-graph G with treewidth at most* 2*, computes* 2*-potential leaf of G in time* $O(n+m)$.

Proof. If $n \leq 3$, then $V(G)$ is a 2-potential-leaf of G. Let us assume that $n \geq 4$. First, let us compute the set of isolated vertices in G, which can be done in $O(n)$ time. If G has only isolated vertices, then any three vertices induce a 2-potential-leaf of G. Otherwise, there is at least one edge in G.

Let G_1 be any connected component of G containing at least one edge. If $|V(G_1)| = 2$, then from Lemma 9, either G has an isolated vertex α and $\{\alpha, u, v\}$ is a 2-potential-leaf or $\{u, v\}$ is a 2-potential leaf.

Otherwise, $|V(G_1)| \geq 3$. Compute a block tree F of G_1 rooted in an arbitrary block R. This can be done in time $O(n+m)$. Note that any node in F corresponds to either a

2-connected component of G or a bridge $uv \in E(G)$. Let C be a leaf block in F, which is furthest from R and $|V(C)|$ is maximum. There are several cases to be considered.

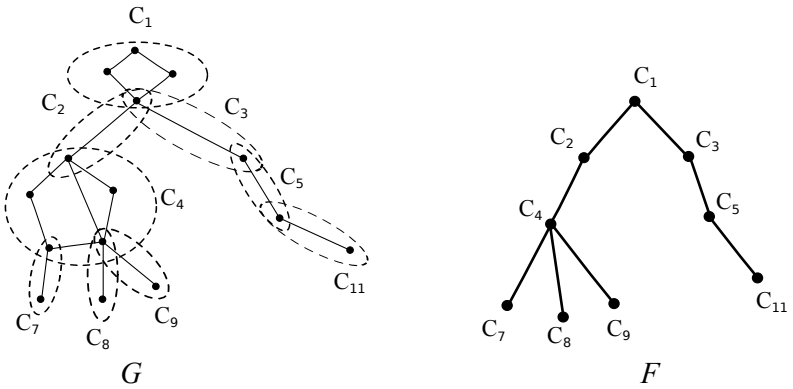


Figure 2.9: This graph G is an induced subgraph of the graph in Fig. 2.8. Its block tree F, with root C_1, has two blocks less than the one in Fig. 2.8 (the blocks C_6 and C_{10}). All leaf blocks, C_7, C_8, C_9, C_{11}, in F contains two vertices of G.

- let us first assume that C is a bridge in G, i.e. C consists of one edge $fp \in E(G)$ and p is a cut vertex. Then f has degree one in G because C is a leaf block in F. Let P be the parent block of C in F. Then any child block A of P in F consists of one edge because C has the maximum number of vertices among all the children of P; and A is a leaf block in F because C is a furthest leaf from the root block R.

  If P has another child block except C in F containing the cut vertex p, then this child block also consists of one edge $f'p \in E(G)$, where f' has degree one in G because this child is also a leaf block in F. (For example, in Fig. 2.9, take C as C_8, which intersects C_9 with a cut vertex.) From Lemma 7, $\{f, p, f'\}$ is a 2-potential-leaf.

  Otherwise P has only one child block C in F containing the cut vertex p. Then any vertex in $N_G(p) \setminus \{f\}$ belongs to P. If P is also a bridge in G, i.e., P consists of one edge $pq \in E(G)$, then p has degree 2 in G. (For example, in Fig. 2.9, take C as C_{11}, whose parent C_5 is also a bridge in G.) From Lemma 6, $\{f, p, q\}$ is a 2-potential-leaf of G. Otherwise, P is a 2-connected component of G and $p \in V(G)$ satisfies the hypothesis of Lemma 8. (For example, in Fig. 2.9, take C as C_7, whose parent C_4 is a 2-connected component of G.) Hence, either G has an isolated vertex α and $\{\alpha, f, p\}$ is a 2-potential-leaf or $\{f, p\}$ is a 2-potential leaf.

- Finally, let us assume that C is a 2-connected component of G. It is known that any graph with at least two vertices of treewidth k contains at least two vertices of degrees at most k [BK11]. There is no degree one vertex in C because C is 2-connected. So there exists two vertices with degree 2 in C. Since C is a leaf in F, there is only one cut vertex of G in C. So there exists a vertex b in C which has degree two in G. If $|V(C)| \geq 4$, then there exists a path between two neighbors a, c of b in $G \setminus \{b\}$ containing at least one internal vertex. (For example, in Fig. 2.8, take C as C_{10}.) From Lemma 9, $\{a, b, c\}$ is a 2-potential-leaf. Otherwise C is a triangle $\{a, b, c\}$

with at least two vertices with degree 2 in G. Again from Lemma 9, $\{a,b,c\}$ is a 2-potential-leaf.

So the total time complexity is $O(n+m)$. □

Corollary 4. *s_2 can be computed in polynomial-time in general graphs. Moreover, a minimum size tree decomposition can be constructed in polynomial-time in the class of partial 2-trees.*

Proof. Let G be any graph. It can be checked in polynomial-time whether $tw(G) \leq 2$ (e.g. see [WC83]). If $tw(G) > 2$, then $s_2 = \infty$. Otherwise $tw(G) \leq 2$, then the result follows from Theorem 5 and Corollary 3. □

2.5 Minimum-size tree-decompositions of width at most 3

In this section, we study the computation of s_3 in the class of trees and 2-connected outerplanar graphs.

2.5.1 computation of s_3 in trees

In this subsection, given a tree G, we show how to find a 3-potential-leaf in G. Please see all the 3-potential leaf of trees in Fig. 2.10.

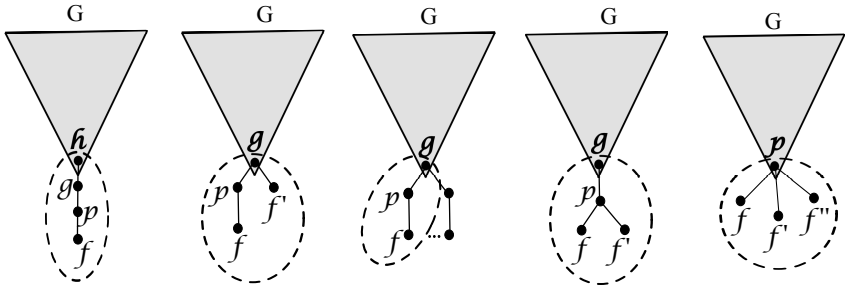


Figure 2.10: All the 3-potential leaves of trees.

Lemma 10. *Let $(T,\mathcal{X})$ be a tree decomposition of a tree G. Let $X \in \mathcal{X}$ and $N_T(X) = \{X_1,\ldots,X_d\}$ for $d \geq 1$. Suppose that for any $1 \leq i \leq d$, $X_i \cap X \subseteq \{x\}$. Then there is a tree decomposition $(T',\mathcal{X}')$ of G of the same width and size as $(T,\mathcal{X})$ such that X is a leaf bag.*

Proof. If there is a bag X_i for $1 \leq i \leq d$ containing x, then let B be X_i. Otherwise let B be any neighbor of X. By definition of the operation $Leaf$, the tree-decomposition $Leaf(X,B,(T,\mathcal{X}))$ is a desired tree decomposition. □

Lemma 11. *Let G be a tree rooted at $r \in V(G)$. Let f be a leaf in G. Let p be the parent of f and let g be the parent of p in G. Let p have degree 2 in G. Let $(T,\mathcal{X})$ be a tree decomposition of G of width at most 3 and size at most $s \geq 1$. If there is no bag in $(T,\mathcal{X})$*

containing all of f,p,g, then there is a tree decomposition $(T',\mathcal{X}')$ of G of width at most 3 and size at most s such that $\{f,p,g\} \in \mathcal{X}'$ is a leaf bag.

Proof. Since $fp \in E(G)$, there is a bag B in $(T,\mathcal{X})$ containing both f and p. We may assume that B is the single bag containing f (otherwise, delete f from any other bag). Similarly, since $pg \in E(G)$, let X be a bag in $(T,\mathcal{X})$ containing both p and g. Let P be the path in T from B to X. Then p is contained in all bags on P and we may assume that p is not contained in any other bags (otherwise, delete p from any other bag). Let B' be the neighbor of B on P. Then $B \cap B' \supseteq \{p\}$. Note that it is possible that $B' = X$.

If $B = \{f,p\}$, then make all other neighbors of B adjacent to B' and delete B. Add a bag $\{f,p,g\}$ adjacent to X. The result is a desired tree decomposition $(T',\mathcal{X}')$.

Otherwise, B contains at least one vertex not in $\{f,p\}$. If $B \cap B' = \{p\}$, then $\{p\}$ separates g from any vertex in $B \setminus \{p\}$. So $B \setminus \{p\} = \{f\}$, i.e., $B = \{f,p\}$. It contradicts with the assumption.

So $|B \cap B'| \geq 2$ and let $\{p,x\} \subseteq B \cap B'$. Then create a bag $Z = (B \setminus \{f,p\}) \cup (B' \setminus \{p,x\})$. (Note that $x \in Z$ since $x \in B$.) So $|Z| \leq 4$. Make Z adjacent to all neighbors of B and all neighbors of B'; and delete the two bags B and B'; and delete f,p from all bags. Finally add another new bag $N = \{f,p,g\}$ adjacent to some bag containing g. The obtained tree decomposition has width at most 3, same size as $(T,\mathcal{X})$, and a bag $N = \{f,p,g\}$. □

Lemma 12. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let f be a leaf in G. Let p be the parent of f and let g be the parent of p in G. Suppose that both p and g have degree 2. Let h be the parent of g (see in Fig. 2.10(a)), then $H = G[\{f,p,g,h\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T,\mathcal{X})$ be any reduced tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f,p,g,h\}$ is a leaf bag.

From Lemma 11, we can assume that there is a bag B in $(T,\mathcal{X})$ containing all f,p,g. We may assume that B is the single bag containing f,p (otherwise, delete f,p from any other bag). Since $gh \in E(G)$, let Y be a bag in $(T,\mathcal{X})$ containing both h and g.

1. If $B = Y = \{f,p,g,h\}$, then the intersection of B and any of its neighbor in T is contained in $\{h\}$. A desired tree decomposition can be obtained from Lemma 10.

2. If $B = \{f,p,g\}$, then the intersection of B and any of its neighbors in T is contained in $\{g\}$. From Lemma 10, there is a tree-decomposition $(T',\mathcal{X}'))$ of the same width and size as the ones of $(T,\mathcal{X})$ such that $B = \{f,p,g\}$ is a leaf. Then delete B in the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$ and add a new bag $N = \{f,p,g,h\}$ adjacent to Y. The obtained tree decomposition has the desired properties.

3. Otherwise, $B = \{f,p,g,x\}$ where $x \neq h$. Then the intersection of B and any of its neighbor in T is contained in $\{g,x\}$. Let P be the path in T from B to Y. Then g is contained in all bags on P. Let B' be the neighbor of B on P. Note that it is possible that $B' = Y$. If $B \cap B' = \{g\}$, then $\{g\}$ separates h from x. So $x \in \{f,p\}$ i.e. $B = \{f,p,g\}$, a contradiction with the assumption. So we have $B \cap B' = \{g,x\}$.

By definition of the operation $Leaf$, the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$ has width at most 3, same size as $(T,\mathcal{X})$, and $B=\{f,p,g,x\}$ is a leaf. Then delete B in the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$ and add a new bag $N=\{f,p,g,h\}$ adjacent to Y. The obtained tree decomposition has the desired properties since $\{g,x\}\subseteq B'$ and $\{g,h\}\subseteq Y$.

□

Lemma 13. *Let G be a tree rooted at $r\in V(G)$ and $|V(G)|\geq 4$. Let f a leaf in G. Let p be the parent of f and let g be the parent of p in G. If p has degree 2 and g has a child f', which is a leaf in G (see in Fig. 2.10(b)), then $H=G[\{f,p,g,f'\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T,\mathcal{X})$ be any reduced tree decomposition of width at most 3 and size at most $s\geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f,p,g,f'\}$ is a leaf bag.

From Lemma 11, we can assume that there is a bag B in $(T,\mathcal{X})$ containing all f,p,g. We may assume that B is the single bag containing f,p (otherwise, delete f,p from any other bag). Since $gf'\in E(G)$, let Y be a bag in $(T,\mathcal{X})$ containing both f and g. We may assume that Y is the single bag containing f' (otherwise, delete f' from any other bag).

- If $B=Y=\{f,p,g,f'\}$, then the intersection of B and any of its neighbor in T is contained in $\{g\}$. A desired tree decomposition can be obtained from Lemma 10.
- If $B=\{f,p,g\}$, then delete f' in Y and add f' in B, then we are in the previous case.
- Otherwise, $B=\{f,p,g,x\}$ where $x\neq f'$. Then the intersection of B and any of its neighbor in T is contained in $\{g,x\}$. Let P be the path in T from B to Y. Then g is contained in all bags on P. Let B' be the neighbor of B on P. If $B\cap B'=\{g,x\}$, then by definition of the operation $Leaf$, the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$ has width at most 3, same size as $(T,\mathcal{X})$, and $B=\{f,p,g,x\}$ is a leaf. Then in the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$, delete f' in Y and remove x from B and add f' to B, i.e. make $B=\{f,p,g,f'\}$. The obtained tree decomposition has the desired properties since $\{g,x\}\subseteq B'$.

  Otherwise $B\cap B'=\{g\}$. Delete f' from the bag Y and add x in Y; delete x from B and add f' in B, i.e., make $B=\{f,p,g,f'\}$; finally make all neighbors of B except B' adjacent to Y since now $\{g,x\}\subseteq Y$. The result is the desired tree decomposition.

□

Lemma 14. *Let G be a tree rooted at $r\in V(G)$ and $|V(G)|\geq 3$. Let f be one of the furthest leaves from r. Let p be the parent of f and let g be the parent of p in G. If g has degree at least 3 and any child of g has degree 2 in G (see in Fig. 2.10(c)), then $H=G[\{f,p,g\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T,\mathcal{X})$ be any reduced tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f,p,g,f'\}$ is a leaf bag.

From Lemma 11, we can assume that there is a bag B in $(T,\mathcal{X})$ containing all f,p,g. We may assume that B is the single bag containing f,p (otherwise, delete f,p from any other bag).

1. If $B=\{f,p,g\}$, then the intersection of B and any of its neighbor in T is contained in $\{g\}$. A desired tree decomposition can be obtained from Lemma 10.

2. Otherwise, $B=\{f,p,g,x\}$. Then the intersection of B and any of its neighbor in T is contained in $\{g,x\}$.

   (a) If there is a neighbor B' of B such that $B\cap B'=\{g,x\}$, then by definition of the operation $Leaf$, the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$ has width at most 3, same size as $(T,\mathcal{X})$, and $B=\{f,p,g,x\}$ is a leaf. Then delete x in B in the tree-decomposition $Leaf(B,B',(T,\mathcal{X}))$ since $\{g,x\}\subseteq B'$. The obtained tree decomposition has the desired properties.

   (b) Otherwise any neighbor of B contains at most one of g and x. If x is not adjacent to g, then there is a connected component in $G\setminus B$ containing a neighbor of g and a neighbor of x. From Lemma 4, there exists a neighbor bag of B in $(T,\mathcal{X})$ containing g and x. It is a contradiction. So we have x is adjacent to g in this case.

      i. x is a child of g. Then x has exactly one child y, which is a leaf in G since f is one of the furthest leaves from r. Since $yx\in E(G)$, there is a bag Y in $(T,\mathcal{X})$ containing both y and x. We may assume that Y is the single bag containing y (otherwise, delete y from any other bag). Since $\{g,x\}\subset B$ and any neighbor of B contains at most one of g and x, any bag except B contains at most one of g and x. Then $g\notin Y$ because $x\in Y$. So y,x,g are not contained in one bag. From Lemma 11, we can modify $(T,\mathcal{X})$ to obtain a tree-decomposition $(T',\mathcal{X}')$ of width at most 3 and size at most s having a leaf bag $X=\{y,x,g\}$. Note that x (resp. y) plays the same role as p (resp. f) in G, i.e., g,p,f and g,x,y are symmetric in G. Hence, the result is a desired tree decomposition.

      ii. x is the parent of g. Let p' be another child of g and let f' be the child of p', which is a leaf in G. Let B' be the bag in $(T,\mathcal{X})$ containing both f' and p'. We may assume that B' is the single bag containing f' (otherwise, delete f' from any other bag). Let X' be a bag containing both p' and g. Then we have $X'\neq B$ (because $p'\notin B$). Since $g\in X'$ any bag except B contains at most one of g and x, we have $x\notin X'$. In the following, we modify $(T,\mathcal{X})$ to obtain a tree-decomposition $(T',\mathcal{X}')$ with width at most 3 and size at most s having a bag $\{f',p',g\}$. Then we are in case 1, since g,p,f and g,p',f' are symmetric in G.

If $B' = X' = \{f', p', g\}$ then, we are done. If $B' = X' = \{f', p', g, x'\}$. Then x' is not x, which is the parent of g, since $x \notin X'$. So we can do as in case 2a or case 2(b)i.
Otherwise, $B' \neq X'$. From Lemma 11, we can modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s having a leaf bag $\{f', p', g\}$.

□

Lemma 15. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let f a leaf in G. Let p be the parent of f. If p has exactly two children f, f' in G and let g be the parent of p in G (see in Fig. 2.10(d)), then $H = G[\{f, f', p, g\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{f, f', p, g\}$ is a leaf bag.

Since $fp \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and p. We may assume that B is the single bag containing f (otherwise, delete f from any other bag). Similarly, let B' be the single bag in $(T, \mathcal{X})$ containing both f' and p. Let X be a bag containing both p and g.

1. If $B = B' = X = \{f, f', p, g\}$, then we can assume that B is the single bag containing p (otherwise, delete p from any other bag). So the intersection of B and any of its neighbor in T is contained in $\{g\}$. Then a desired tree decomposition can be obtained from Lemma 10.

2. If $B = B' = \{f, f', p\}$, then the intersection of B and any of its neighbor in T is contained in $\{p\}$. Let Y be a neighbor of B in T containing p. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p\}$ is a leaf. Then delete B and add a new bag $N = \{f, f', p, g\}$ adjacent to X. The result is a desired tree decomposition.

3. If $B = B' = \{f, f', p, x\}$ and $x \neq g$, then the intersection of B and any of its neighbor in T is contained in $\{p, x\}$. Since $x \notin \{f, f', g\}$, p is not adjacent to x. There is a connected component in $G \setminus B$ containing a neighbor of p and a neighbor of x. From Lemma 4, there exists a neighbor bag of B in $(T, \mathcal{X})$ containing p and x. Let Y be such a neighbor of B in T. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p, x\}$ is a leaf. Then delete x from B and we get a tree decomposition having a bag $\{f, f', p\}$. So we are in case 2.

4. If $B \neq B'$ and $|B| \leq 3$, then delete f' in B and add f' in B. Then we are in case 2 or 3. It is proved similarly if $B \neq B'$ and $|B'| \leq 3$.

5. Otherwise $B \neq B'$ and $|B| = |B'| = 4$. Let $B = \{f, p, x, y\}$ and $B' = \{f', p, x', y'\}$. Let P be the path in T from B to B'. Then p is contained in all bags on P. Let Y be

the neighbor of B on P. If $B \cap Y = \{p\}$, then $\{p\}$ separates x from x'. But p is not a separator between any two vertices in $V(G) \setminus \{f, f'\}$. It is a contradiction. So w.l.o.g. we can assume that $B \cap Y \supseteq \{p, x\}$. Deleting f, f', p in all bags of $(T, \mathcal{X})$. Add a new bag $Z = \{x, y\} \cup Y \setminus \{p, x\}$ adjacent to all neighbors of the two bags B, Y and delete B and Y. Finally add another new bag $N = \{f, f', p, g\}$ adjacent to a bag containing g. The obtained tree decomposition has the desired properties.

□

Lemma 16. *Let G be a tree rooted at $r \in V(G)$ and $|V(G)| \geq 4$. Let all children of p be leaves in G and p have at least three children f, f', f'' (see in Fig. 2.10(e)). Then $H = G[\{p, f, f', f''\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any reduced tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{p, f, f', f''\}$ is a leaf bag.

Since $fp \in E(G)$, there is a bag B in $(T, \mathcal{X})$ containing both f and p. We may assume that B is the single bag containing f (otherwise, delete f from any other bag). Similarly, let B' (resp. B'') be the single bag in $(T, \mathcal{X})$ containing both f' (resp. f') and p.

1. If $B = B' = B'' = \{f, f', f'', p\}$, then the intersection of B and any of its neighbor in T is contained in $\{p\}$. A desired tree decomposition can be obtained from Lemma 10.

2. If $B = B' = \{f, f', p\}$, then delete f'' in B'' and add f'' in B. Then we are in case 1. It can be proved similarly if $B = B'' = \{f, f'', p\}$ or $B' = B'' = \{f', f'', p\}$.

3. If $B = B' = \{f, f', p, x\}$ and $x \neq f''$, then the intersection of B and any of its neighbor in T is contained in $\{p, x\}$.

   If x is a child of p, then x is also a leaf in G and x play the same role as f''. Then we are in case 1. So in the following we assume that x is not a child of p.

   If x is not the parent of p, then p is not adjacent to x. So there is a connected component in $G \setminus B$ containing a neighbor of p and a neighbor of x. From Lemma 4, there exists a neighbor bag of B in $(T, \mathcal{X})$ containing p and x. Let Y be such a neighbor of B in T. By definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p, x\}$ is a leaf. Then delete x from B and we get a tree decomposition having a bag $\{f, f', p\}$. So we are in case 2.

   Otherwise x is the parent of p. Let P be the path in T from B to B''. Then p is contained in all bags on P. Let Y be the neighbor of B on P. If $B \cap Y = \{p, x\}$, then by definition of the operation $Leaf$, the tree-decomposition $Leaf(B, Y, (T, \mathcal{X}))$ has width at most 3, same size as $(T, \mathcal{X})$, and $B = \{f, f', p, x\}$ is a leaf. Then deleting x from B we are in case 2. Otherwise, $B \cap Y = \{p\}$. So $\{p\}$ separates x from all vertices in $B'' \setminus \{p\}$. Then all vertices in $B'' \setminus \{p\}$ are children of p and so they are leaves in G. So we can assume that any vertex in $B'' \setminus \{p\}$ are contained only in

B'' (otherwise we can delete it in any other bags). Then delete f, f' from B and add vertices of $B'' \setminus \{f'', p\}$ in B; and make $B'' = \{f, f', f'', p\}$. Then we are in case 1.

The cases $B = B'' = \{f, f'', p, x\}$ and $x \neq f'$ or $B' = B'' = \{f', f'', p, x\}$ and $x \neq f$ can be proved similarly.

4. Otherwise, none two of f, f', f'' are contained in a same bag.

   If $|B| \leq 3$, then delete f' in B' and add f' in B. Then we are in case 2 or 3. It is proved similarly if and $|B'| \leq 3$ or $|B''| \leq 3$.

   Otherwise $|B| = |B'| = |B''| = 4$. In the following, we are going to modify $(T, \mathcal{X})$ to obtain a tree-decomposition with width at most 3 and size at most s having a bag X containing at least two of f, f', f'' or $f \in X$ and $|X| \leq 3$. Then we are in the above cases. Note that all children of p play the same role (they are all leaves) in G. So it is enough to get that X contains at least two children of p or that X contains one child of p and $|X| \leq 3$.

   Let T_p be the subtree in T induced by all the bags containing p. If $|V(T_p)| \leq 2$, there exists one bag containing at least two children of p since p has at least three children. Then it is done. So we assume that $|V(T_p)| \geq 3$. There is a bag $R \in V(T_p)$ containing both p and g. Root T_p at R and let $L \in V(T_p)$ be one of the furthest leaf bag in T_p from R. If there is no child of p in L, then we can delete p in L and consider $T_p \setminus \{L\}$. So we can assume there is a vertex $l \in L$, which is a child of p in G. Let Y be the neighbor of L in T_p. If the intersection of $L \cap Y = \{p\}$, then p separate any vertex in $L \setminus \{p\}$ and any vertex in $Y \setminus \{p\}$. So at least one of L, Y, denoted as X, contains only p and children of p. Then either X contains at least two children of p or X contains only one children and $|X| = 2$. So $(T, \mathcal{X})$ and X satisfy the desired properties.

   Otherwise, $|L \cap Y| \geq 2$. If Y has no other child except L in T_p, then $Y \neq R$ since $|V(T_p)| \geq 3$. Let $X = \{p, l\}$ if Y contains no child of p; and $X = \{p, l, l'\}$ if Y contains one child l' of p. Add a new bag $Z = Y \cup L \setminus X$. Since $|Y \cap L| \geq 2$, $|Y \cup L| \leq 6$. Then $|Z| \leq 4$, since $X \subseteq Y \cup L$ and $|X| \geq 2$. Make Z adjacent to all neighbors of Y, L in T and delete Y, L. Finally make X adjacent to R. The obtained tree decomposition and X have the desired properties.

   Otherwise, Y has at least another child L' in T_p. Then L' is also a furthest leaf from R in T_p, since L is a furthest leaf from R. For the same reason as L, there is a vertex $l' \in L$, which is a child of p in G. Let $L = \{l, p, x, y\}$ and $L' = \{l', p, x', y'\}$. So the intersection of L (resp. L') and any of its neighbors except Y in T is contained in $\{x, y\}$ (resp. $\{x', y'\}$). Create a new bag $N = \{x, y, x', y'\}$ adjacent to all neighbor of L, L' and delete L, L'. Finally add another bag $X = \{p, l, l'\}$ adjacent to Y. The obtained tree decomposition and X have the desired properties.

□

From Lemmas 12- 16 and Corollary 3, we obtain the following result.

Corollary 5. *s_3 and a minimum size tree decomposition of width at most 3 can be computed in polynomial-time in the class of trees.*

Proof. From Corollary 3, it is enough to prove we can find a 3-potential-leaf in any tree in polynomial time.

Let G be any tree. If $|V(G)| \leq 4$, then $V(G)$ is a 3-potential-leaf. Let us assume that $|V(G)| \geq 5$. Root G at any vertex r. Let f be one of the furthest leaves from r in G. Let p, g, h be the first three vertices on the path from f to r in G (if exist), i.e. p is f's parent; g is p's parent; and h is g's parent in G.

- If g, p both have only one child in G, then $\{f, p, g, h\}$ is a 3-potential-leaf of G from Lemma 12;
- If p has only one child and g has a child f', which is a leaf in G, then $\{f, p, g, f'\}$ is a 3-potential-leaf of G from Lemma 13;
- If p has only one child and any child of g has exactly one child, then $\{f, p, g\}$ is a 3-potential-leaf of G from Lemma 14;
- If p has only one child and there exist a child p' of g, which has exactly two children f_1, f_2, then $\{f_1, f_2, p', g\}$ is a 3-potential-leaf of G from Lemma 15;
- If p has only one child and there exist a child p' of g, which has at least three children f_1, f_2, f_3, then $\{f_1, f_2, f_3, p'\}$ is a 3-potential-leaf of G from Lemma 16;
- If p has exactly two children f, f', then $\{f, f', p, g\}$ is a 3-potential-leaf of G from Lemma 15;
- Otherwise p has at least three children f, f', f'', then $\{f, f', f'', p\}$ is a 3-potential-leaf of G from Lemma 16.

□

In fact, the algorithm for trees can be extended to forests by consider their connected component, i.e., trees. The only difference is in Lemma 14 the 3-potential-leaf becomes $\{f, p, g, \alpha\}$ if there is an isolated vertex α in the given forest.

2.5.2 Computation of s_3 in 2-connected outerplanar graphs

In this subsection, given a 2-connected outerplanar graph G, we show how to find a 3-potential-leaf in G. See all the 3-potential leaf of 2-connected outerplanar graphs in Fig. 2.11.

The following fact is well known for 2-connected outerplanar graphs.

Lemma 17. *[Sys79] A 2-connected outerplanar graph has the unique Hamilton cycle.*

In the rest of this subsection, let G be a 2-connected outerplanar graph and C be the Hamilton cycle in G.

Definition 3. *Any edge in $E(G) \setminus E(C)$ is called a chord in G.*

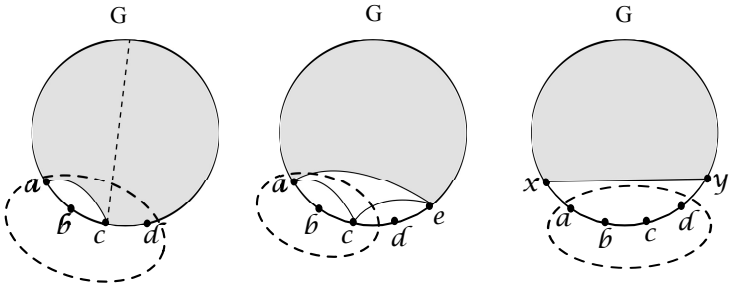


Figure 2.11: All the 3-potential leaves of 2-connected outerplanar graphs.

The vertices $v_1,\ldots,v_j \in V(G)$, for $2 \leq j \leq |V(G)|$, are *consecutive* in C (we also say that they are consecutive in G) if $v_iv_{i+1} \in E(C)$ for $1 \leq i \leq j-1$; and $v_1,\ldots,v_j \in V(G)$ are also called the consecutive vertices from v_1 to v_j.

Lemma 18. *Let $a,b,c,d \in V(G)$ be consecutive in C. If $\{a,b,c\}$ induces a clique and c has degree 3 in G (see in Fig. 2.11(a)), then $H = G[\{a,b,c,d\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T,\mathcal{X})$ be any tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{a,b,c,d\}$ is a leaf bag.

Since $\{a,b,c\}$ induces a clique in G, there is a bag B containing all a,b,c. Let X be a bag in $(T,\mathcal{X})$ containing both c and d (it exists since $cd \in E(G)$). Note that b is not incident to any chords, i.e. has degree 2. (Because if $by \in E(G)$ is a chord in G, then deleting all chords except ac,by in G and contracting the edges in C except ab,bc we get a K_4-minor in G. It is a contradiction with the fact that G is outerplanar.)

Replace b,c with a in all bags of $(T,\mathcal{X})$. Then $(T,\mathcal{X})$ becomes a tree decomposition $(T',\mathcal{X}')$ of the graph G' obtained by contracting the edges ab and bc. The bag X becomes X', which contains both a and d; and B becomes $B' = \{a\}$ if $B = \{a,b,c\}$ or $B' = \{a,x\}$ if $B = \{a,b,c,x\}$. From Corollary 2, in both case there exists a neighbor Y of B' such that $B' \subseteq Y$. So B' can be reduced in $(T',\mathcal{X}')$. The tree decomposition $Reduce(B',Y,(T',\mathcal{X}'))$ has one bag less than $(T,\mathcal{X})$. Finally, add a new bag $N = \{a,b,c,d\}$ adjacent to X', which contained both a and d, in the tree decomposition $Reduce(B',Y,(T',\mathcal{X}'))$. The result is a desired tree decomposition, because b,c are not adjacent to any vertices in $V(G) \setminus N$. □

Lemma 19. *Let $a,b,c,d,e \in V(G)$ be consecutive in C. If $\{a,b,c\}$ and $\{c,d,e\}$ induce two cliques respectively in G and $ae \in E(G)$ (see in Fig. 2.11(b)), then $H = G[\{a,b,c\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T,\mathcal{X})$ be any tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify it to obtain a tree-decomposition with width at most 3 and size at most s and in which $\{a,b,c\}$ is a leaf bag.

Since $\{a,b,c\}$ (resp. $\{c,d,e\}$) induces a clique in G, there is a bag X (resp. Y) containing all a,b,c (resp. c,d,e). Note that b,c,d are not adjacent to any vertices in $V(G) \setminus \{a,b,c,d,e\}$.

Delete b,c,d in all bags of $(T,\mathcal{X})$. Then $(T,\mathcal{X})$ becomes a tree decomposition $(T',\mathcal{X}')$ of the graph $G' = G \setminus \{b,c,d\}$. The bag X becomes becomes $X' = \{a\}$ if $X = \{a,b,c\}$ or $X' = \{a,x\}$ if $B = \{a,b,c,x\}$. From Corollary 2, in both case there exists a neighbor A of X' such that $X' \subseteq A$. So X' can be reduced in $(T',\mathcal{X}')$. Similarly, the bag Y becomes Y', which can also be reduced in $(T',\mathcal{X}')$. After reducing the two bags X',Y' in $(T',\mathcal{X}')$, let the obtained tree decomposition be $(T'',\mathcal{X}'')$. Finally, add two new bags $N_1 = \{a,b,c\}$ and $N_2 = \{a,c,d,e\}$; make N_1 adjacent to N_2 and make N_2 adjacent to a bag Z containing both a and e in the tree decomposition $(T'',\mathcal{X}'')$. (Z exists because $ae \in E(G')$.) The result is a desired tree decomposition. □

Lemma 20. *Let C_l be a cycle of $l \geq 4$ vertices. Let $(T,\mathcal{X})$ be a tree decomposition of C_l of width at most 3. Then there exist either a bag containing all vertices of $V(C_l)$ (only if $l = 4$) or two bags $X,Y \in \mathcal{X}$ such that X (resp. Y) contains at least three consecutive vertices x_1,x_2,x_3 (resp. y_1,y_2,y_3) and the two edge sets $\{x_1x_2,x_2x_3\} \cap \{y_1y_2,y_2y_3\} = \emptyset$.*

Proof. The treewidth of any cycle is bigger than 1, so there exists a bag in any tree decomposition of a cycle (with at least 4 vertices) containing two vertices not consecutive, equivalently they are not adjacent in the cycle. We prove the lemma by induction on l in the following.

First let us prove that it is true for $l = 4$. Let a,b,c,d be the four consecutive vertices in C_4. Let $(T,\mathcal{X})$ be a tree decomposition of width at most 3. Then there exists a bag containing a,c or b,d. W.l.o.g assume a,c are contained in one bag. So $(T,\mathcal{X})$ is also a tree decomposition of H, obtained from C_4 by adding the edge ac. The set $\{a,b,c\}$ induces a clique in H. So there is a bag X containing a,b,c. For the same reason, there is a bag Y containing c,d,a. If $X = Y$ then there is a bag containing all a,b,c,d of $V(C_4)$. Otherwise there are two bags X,Y such that $X \supseteq \{a,b,c\}$ and $Y \supseteq \{c,d,a\}$. We see that $\{ab,bc\} \cap \{cd,da\} = \emptyset$. So the lemma is true for $l = 4$.

Now suppose it is true for $l \leq n-1$ and we prove it for $l = n \geq 5$. Note that since $(T,\mathcal{X})$ has width 3 and $l \geq 5$, there is no bag containing all vertices of $V(C_l)$. So in the following we prove that there always exist two bags X,Y with the desired properties. Let $v_1,\ldots,v_n$ be the n consecutive vertices in C_n. Let (T,X) be a tree-decomposition of width at most 3 of C_n. Then there exists a bag containing two non-adjacent vertices v_i,v_j for $1 \leq i < j \leq n$. So $(T,\mathcal{X})$ is also a tree decomposition of the graph H, obtained from C_n by adding the edge v_iv_j. The graph H is also the union of two subcycles C^1 induced by $\{v_i,\ldots,v_j\}$ and C^2 induced by $\{v_j,\ldots,v_n,\ldots,v_i\}$. Then $\max\{|C^1|,|C^2|\} \leq n-1$. Let (T^1,X^1) (resp. (T^2,X^2)) be the tree decomposition of C^1 (resp. C^2) obtained by deleting all vertices not in C^1 (resp. C^2) in the bags of (T,X).

If $|V(C^1)| = 3$ then there is a bag in (T^1,X^1) containing $V(C^1) = \{v_i,v_{i+1},v_j = v_{i+2}\}$. So $v_iv_j \notin \{v_iv_{i+1},v_{i+1}v_j\}$.

If $|V(C^1)| \geq 4$ then, by induction, there exist either a bag in (T^1,X^1) containing all vertices of $V(C^1) = \{v_i,v_{i+1},v_{i+2},v_j = v_{i+3}\}$ or two bags A,B in (T^1,X^1) containing three consecutive vertices a_1,a_2,a_3 and b_1,b_2,b_3 respectively in C^1; moreover, $\{a_1a_2,a_2a_3\} \cap \{b_1b_2,b_2b_3\} = \emptyset$. So we have either $v_iv_j \notin \{a_1a_2,a_2a_3\}$ or $v_iv_j \notin \{b_1b_2,b_2b_3\}$.

In both cases ($|V(C^1)| = 3$ and $|V(C^1)| \geq 4$), there is at least one bag X in (T^1,X^1) containing three consecutive vertices in C^1, denoted as x_1,x_2,x_3, such that $v_iv_j \notin \{x_1x_2,x_2x_3\}$.

So x_1, x_2, x_3 are also consecutive in C. Similarly, there is at least one bag Y in (T^2, X^2) containing three consecutive vertices in C^2, denoted as y_1, y_2, y_3, such that $v_i v_j \notin \{y_1y_2, y_2y_3\}$. So y_1, y_2, y_3 are also consecutive in C. Finally, we have $\{x_1x_2, x_2x_3\} \cap \{y_1y_2, y_2y_3\} = \emptyset$ because $E(C^1) \cap E(C^2) = \{v_iv_j\}$ and $v_iv_j \notin \{x_1x_2, x_2x_3\}$. □

Lemma 21. *Let xy be a chord in G. Let C' be the set of all the consecutive vertices from x to y in C and $|C'| \geq 4$. If each vertex in $C' \setminus \{x,y\}$ has degree 2 in G, then for any consecutive vertices $a,b,c,d \in C'$ (see in Fig. 2.11(c)), $H = G[\{a,b,c,d\}]$ is a 3-potential-leaf of G.*

Proof. Let $(T, \mathcal{X})$ be any tree decomposition of width at most 3 and size at most $s \geq 1$ of G. We show how to modify $(T, \mathcal{X})$ to obtain a tree-decomposition of G, which has width at most 3, size at most s and a leaf bag $\{a,b,c,d\}$.

Note that the vertices of C' induce a cycle in G. Without confusion, we denote this cycle C'. Let (T', X') be the tree decomposition of C' obtained by deleting all vertices not in C' in the bags of (T, X). From Lemma 20, there is either a bag containing all vertices in C' (only if $|C'| = 4$); or two bags X, Y containing three consecutive vertices in C' respectively and the two corresponding edge sets do not intersect.

In the former case, $V(C') = \{a,b,c,d\}$ and so $(T, \mathcal{X})$ is also a tree decomposition of $G \cup \{ac\}$, from Lemma 18, $\{a,b,c,d\}$ is a 3-potential-leaf of G.

In the latter case, let $X \supseteq \{u,v,w\}$ and $Y \supseteq \{u',v',w'\}$, where u,v,w (resp. u',v',w') are consecutive in C'. Since $\{uv, vw\} \cap \{u'v', v'w'\} = \emptyset$, we have either $xy \notin \{uv, vw\}$ or $xy \notin \{u'v', v'w'\}$. W.l.o.g. assume that $xy \notin \{uv, vw\}$. Then u,v,w are also consecutive in C and at least one of u,w has degree 2 in G. W.l.o.g. suppose w has degree 2 in G, i.e.$w \notin \{x,y\}$ (since x,y have degree at least 3 in G). Let $z \in C'$ be the other neighbor (except v) of w in C'. (z exists because $w \notin \{x,y\}$.)

$(T, \mathcal{X})$ is also a tree decomposition of $G \cup \{uw\}$, which is still an outerplanar graph by assumptions. Note that w has degree 3 in the graph $G \cup \{uw\}$. So from Lemma 18, we can modify $(T, \mathcal{X})$ to obtain a tree-decomposition $(T', \mathcal{X}')$ of $G \cup \{uw\}$, which has width at most 3, size at most s and a leaf bag L containing four consecutive vertices $\{u,v,w,z\}$. Note that $(T', \mathcal{X}')$ is also a tree decomposition of G. So we get a tree decomposition where a leaf bag contains 4 consecutive vertices of C'. It remains to show how to modify it to obtain a tree decomposition with a leaf bag $\{a,b,c,d\}$.

Let B be the neighbor of L in T. Then $u,z \in B$ since each of u,z is adjacent to some vertices in $G \setminus L$. We can assume that L is the single bag containing v,w in $(T', \mathcal{X}')$, because otherwise we can delete them in any other bags. Thus, deleting the bag L in $(T', \mathcal{X}')$, we get a tree decomposition $(T_1, \mathcal{X}_1)$ of the graph G_1, which is the graph obtained by deleting v,w and adding an edge uz in G. So $(T_1, \mathcal{X}_1)$ has width at most 3 and size at most $s-1$. Note that the graph G_1 is isomorphic to the graph $G_2 \equiv G \cup \{ad\} \setminus \{b,c\}$ since $z \in C'$. So from the tree decomposition $(T_1, \mathcal{X}_1)$ of G_1 we can obtain a tree decomposition $(T_2, \mathcal{X}_2)$ of G_2 with the same width and size. Note that since $ad \in E(G_2)$, there is a bag Y containing both a and d. Finally, add a new bag $N = \{a,b,c,d\}$ adjacent to Y in $(T_2, \mathcal{X}_2)$. The result is a desired tree decomposition. □

Lemma 22. *There is an algorithm that, for any 2-connected outerplanar graph G, computes 3-potential leaf of G in polynomial time.*

Proof. Let G be a 2-connected outerplanar graph and C be the unique Hamilton cycle of G. If $|V(G)| \leq 4$, then $V(G)$ is a 3-potential-leaf of G. Otherwise, $|V(G)| \geq 5$ and consider the outerplanar embedding of G.

- If there exists an inner face f with at most one chord of G and f has at least four vertices, then from Lemma 21, the set of any four consecutive vertices in f, which are also consecutive in C, is a 3-potential-leaf in G.
- If there is an inner face $f = \{a,b,c\}$ with only one chord ac of G and c has degree 3, then let d be the other neighbor of c except b,a. From Lemma 18, the set of four consecutive vertices a,b,c,d, is a 3-potential-leaf in G.
- Otherwise, let $\mathcal{F}$ be the set of all inner faces with only one chord of G. Then any face $f \in \mathcal{F}$ has three vertices and both the two endpoints of the chord in f have degree at least 4, i.e., they are incident to some other chords except this one. We can prove by induction on $|V(G)|$ that:

  Claim 1. *There exist two faces $f_1, f_2 \in \mathcal{F}$ such that (1)$f_1 = \{a,b,c\}$; (2) $f_2 = \{c,d,e\}$; (3) a,b,c,d,e are consecutive in G; (4) there is a face f_0 containing both ac and ce and at most one chord, which is not in any face of $\mathcal{F}$. See in Fig. 2.12.*

  It is true when $|V(G)| = 5$. Assume that it is true for $|V(G)| \leq n-1$. Now we prove it is true for $|V(G)| = n$. Note that $\mathcal{F} \neq \emptyset$ if there is at least one chords in G, which is valid in this case. Let $f \in \mathcal{F}$ have three consecutive vertices x,y,z and let $xz \in E(G)$ be the single chord in f. Then the graph $G \setminus y$ is a 2-connected outerplanar graph with $n-1$ vertices. From the assumption, we have the desired faces f_0', f_1', f_2' in $G \setminus y$. If xz is not an edge in any face of f_1', f_2', then these faces are also the desired faces in G. Otherwise, let xz be an edge of f_1' or $f_2' = \{x,z,t\}$. Then z has degree 3 in G, i.e. it is not incident to any other chords except xz, since $xt \in E(G)$. So we are in second case above, which contradicts with the assumption.

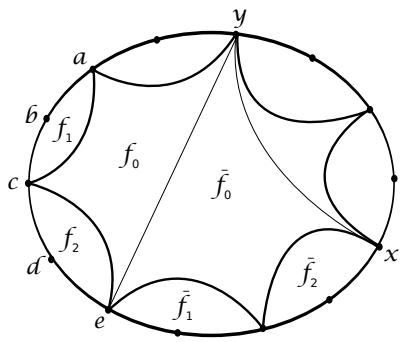


Figure 2.12: $\mathcal{F}$ is the set of all inner faces with only one chord of G, such as $f_1, f_2, \bar{f}_1, \bar{f}_2$. The faces f_0, f_1, f_2 satisfy the properties in Claim 1. But $\bar{f}_0, \bar{f}_1, \bar{f}_2$ does not satisfy the properties since $\bar{f}_0$ contains two edges ey, xy which are not in any face of $\mathcal{F}$.

In the following, let f_0, f_1, f_2 be the faces as in Claim 1. If $ae \in E(G)$, then from Lemma 19, $\{a,b,c\}$ is a 3-potential-leaf of G.

Otherwise, we can prove that any tree decomposition of G of width at most 3 can be modified to a tree decomposition of $G \cup \{ae\}$ with the same width and size in the following. So $\{a,b,c\}$ is a 3-potential-leaf of G.

Let $(T,\mathcal{X})$ be a tree decomposition of width at most 3 and size at most $s \geq 1$ of G. Let $(T_0,\mathcal{X}_0)$ be the tree decomposition obtained by deleting all vertices not in f_0. Then $(T_0,\mathcal{X}_0)$ is a tree decomposition of f_0 (without confusion f_0 is used to denote the face and the cycle induced by vertices in f_0 as well). From Lemma 20, there is a bag containing three consecutive vertices u,v,w in f_0 and uv,vw are edges of some faces in $\mathcal{F}$. (Note that u,v,w are not consecutive in C.) So $(T,\mathcal{X})$ is also a tree decomposition of $G \cup uw$. The graph $G \cup uw$ and the graph $G \cup ae$ are isomorphic. So from $(T,\mathcal{X})$ we can obtain a tree decomposition $(T',\mathcal{X}')$ of $G \cup ae$ with the same width and size. Then $(T',\mathcal{X}')$ is the desired tree decomposition.

□

From Lemmas 22 and Corollary 3, we obtain the following result.

Corollary 6. *s_3 can be computed and a minimum size tree decomposition of width at most 3 can be constructed in polynomial-time in the class of 2-connected outerplanar graphs.*

2.6 Perspective

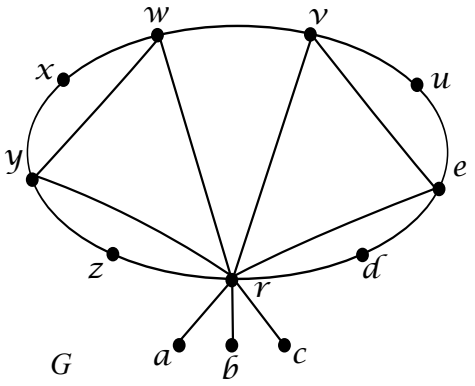
In this chapter, we gave preliminary results on the complexity of minimizing the size of tree-decompositions with given width. Table 2.1 summarizes our results as well as the remaining open questions.

	s_1	s_2	s_3	$s_k, k \geq 4$
$tw = 1$	$P(trivial)$	P	P	?
$tw = 2$	-	P	?	?
$tw = 3$	-	-	?	?
$tw \geq 4$	-	-	-	NP-hard

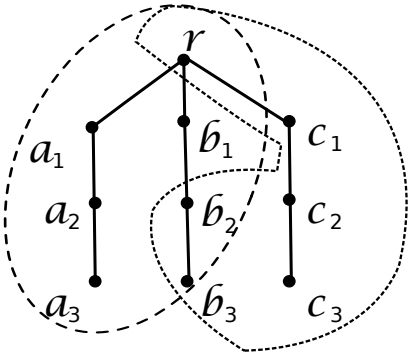
Table 2.1: Summary of the complexity results.

We currently investigate the case of s_3 in the class of connected graphs with treewidth 2 or 3 and we conjecture it is polynomially solvable. But it is more tricky than computing s_3 in trees and 2-connected outerplanar graphs. It seems that a global view of the graph needs to be considered to decide wether a subgraph is a 3-potential leaf of the graph. See an example in Fig. 2.13(a). In this example, G is a connected outerplanar graph. $\{r,a,b,c\}$ is not a 3-potential leaf of G, but it is a 3-potential leaf of $G \setminus \{yw\}$. Let $G' \equiv G \setminus \{a,b,c\}$. Then G' is 2-connected outerplanar. From the algorithm of computing s_3 in 2-connected outerplanar graphs in subsection 2.5.2, we can compute that $s_3(G') = 5$. So if $\{r,a,b,c\}$ is a potential leaf of G, then $s_3(G) = 6$. But there exists a tree decomposition of G of width 3 and size 5, where the bags are $\{a,r,z,y\},\{r,y,x,w\},\{b,r,w,v\},\{r,v,u,e\},\{c,r,d,e\}$. So

$\{r,a,b,c\}$ is not a 3-potential leaf of G. However, in the graph $G'' \equiv G \setminus \{yw\}$, we can prove that $s_3(G'') = 5$ and there is a minimum size tree decomposition containing $\{r,a,b,c\}$ as a leaf bag, i.e. $\{r,a,b,c\}$ is 3-potential leaf of G''. So the existence of the edge yw, not incident to any vertex in $\{r,a,b,c\}$, changes the behavior of $\{r,a,b,c\}$.



(a) $\{r,a,b,c\}$ is not a 3-potential leaf of G, but it is a 3-potential leaf of $G \setminus \{yw\}$. The five bags $\{a,r,z,y\},\{r,y,x,w\},\{b,r,w,v\},\{r,v,u,e\},\{c,r,d,e\}$ connecting as a path in this order forms a tree decomposition of G.



(b) In any minimum size tree decomposition of width 5 (and size 2) of this tree, there exists a bag inducing a forest, not a subtree. For example, in a tree decomposition of width 5 and size 2, one bag is $\{r,a_1,a_2,a_3,b_1,b_2\}$ and the other one is $\{r,b_2,b_3,c_1,c_2,c_3\}$.

Figure 2.13

The problem of computing s_k, for $k \geq 4$, seems more intricate already in the case of trees. Indeed, our polynomial-time algorithms to compute s_k, $k \leq 3$, in trees mainly rely on the fact that, for any tree T, there exists a minimum-size tree-decomposition of T with width at most 3, where each bag induces a connected subtree. This is unfortunately not true anymore in the case of tree-decomposition with width 5. As an example, consider the tree G (with 10 nodes) obtained from a star with three 3 leaves by subdividing twice each edge. See in Fig. 2.13(b). $s_5(G) = 2$ and any minimum size tree decomposition has a bag X such that $G[X]$ is disconnected.

CHAPTER 3

k-Good Tree Decomposition

Contents

This chapter studies a special tree-decomposition of a graph, called *k-good tree decomposition* for an integer $k \geq 2$, in which each bag contains a dominating set of size at most $k-1$ inducing a path in the graph. The main result of this chapter is inspired by a study of the *cops and robber* game in k-chordal graphs for an integer $k \geq 3$. We show that given a graph G and an integer $k \geq 3$, there is a quadratic time algorithm that outputs either an induced cycle of length at least $k+1$ in G or a k-good tree decomposition of G.

It is well-known that given a graph G and an integer k it is NP-complete to decide whether there is an induced cycle of length at least k when k is an input. Equivalently, it is NP-complete to decide whether the chordality of a graph G is at least k. The problem remains NP-complete even when G is planar. However, as mentioned in Chapter 1, the chordality is a contraction-bidimensional parameter: the chordality of an $r \times r$-grid is at least $g(r) = O(r^2)$; and it does not increase when taking contractions. In [ST10], there is an FPT algorithm for finding a longest induced cycle in bounded treewidth graphs, where the parameter is the treewidth of the graph. Combining with the *bidimensionality theory* in [DH08a], there is FPT algorithm to decide whether the chordality of any planar graph G is at least k in $2^{O(\sqrt{k})}n$ time. But as always, this approach is not practical because a tree decomposition of a bounded width is required. Therefore, it is interesting to find a practical algorithm for computing the chordality without using the tree decomposition of bounded width.

Based on our main result mentioned above, given a graph G and an integer $k \geq 3$, if our algorithm outputs an induced cycle of length at least $k+1$, then the chordality of G is bigger than k; otherwise, our algorithm outputs a k-good tree decomposition. Note that in the latter case, it is possible that there exists an induced cycle of length at least $k+1$. So

if we can find a practical algorithm (FPT algorithm without very huge constants because of the NP-completeness) for computing the chordality of a graph admitting a k-good tree decomposition for some integer $k \geq 3$, then we find a practical algorithm (FPT without very huge constants) for deciding the chordality of any graph.

An initial step is to compute the chordality of the subgraph, called *k-super-caterpillar*, induced by each bag of a k-good tree decomposition, which contains a dominating set of size at most $k-1$ inducing a path in the graph. We were expecting polynomial time algorithm for computing the chordality in the class of *planar k-super-caterpillars*, which are planar and k-super-caterpillars. We find a linear time dynamic programming algorithm for computing the chordality in a special subclass of planar k-super-caterpillar in Section 3.4.2. However, after many attempts, we conjecture that it is NP-complete to compute the chordality of a planar k-super-caterpillar when k is part of the input, since we prove that this problem is NP-complete if the problem of deciding whether there is a Hamiltonian cycle in a given planar graph with a Hamiltonian path is NP-complete. More details is shown in Section 3.4.

The results of Section 3.2 and 3.3 is a collaboration with N. Nisse, K. Suchan and A. Kosowski. They appeared in the proceedings of the conference ICALP 2012 [c-KLNS12] and in the journal Algorithmica [j-KLNS14]. The results of Section 3.4 is a collaboration with N. Nisse and A. K. Maia.

3.1 Introduction

Because of the huge size of real-world networks, an important current research effort concerns exploiting their structural properties for algorithmic purposes. Indeed, in large-scale networks, even algorithms with polynomial-time complexity in the size of the instance may become unpractical. Therefore, it is important to design algorithms depending only quadratically or linearly on the size of the network when its topology is expected to satisfy some properties. Among these properties, recall that the chordality of a graph is the length of its longest induced (i.e., chordless) cycle. The (Gromov) *hyperbolicity* of a graph reflects how the metric (distances) of the graph is close to the metric structure of a tree. More precisely, a graph has hyperbolicity $\leq \delta$ if, for any $u,v,w \in V(G)$ and any shortest paths P_{uv}, P_{vw}, P_{uw} between these three vertices, any vertex in P_{uv} is at distance at most δ from $P_{vw} \cup P_{uw}$ [Gro87]. Intuitively, in a graph with small hyperbolicity, any two shortest paths between the same pair of vertices are close to each other. Several recent works take advantage of such structural properties of large-scale networks for algorithm design (e.g., routing [KPBnV09, CSTW09]). As mentioned in Chapter 1, Internet-type networks have a high clustering coefficient (see e.g. [WS98, OP09]), leading to the existence of very few long chordless cycles, whereas their low (logarithmic) diameter implies a small hyperbolicity [dMSV11].

Another way to study tree-likeness of graphs is by tree-decompositions. We have already seen that such decompositions play an important role in design of efficient algorithms in Chapter 1. However, from the practical point of view, this approach has several drawbacks. First, all the mentioned algorithms in Chapter 1 are linear in the size of the graph

but (at least) exponential in the treewidth. Moreover, due to the high clustering coefficient of large-scale networks, their treewidth is expected to be large [dMSV11]. Hence, to face these problems, it is important to focus on the structure of the bags of the tree-decomposition, instead of trying to minimize their size. In this chapter, we consider the k-good tree-decompositions, in which all vertices of each bag induces a k-super-caterpillar in the graph. Such decompositions turn out to be applicable to a large family of graphs, including k-chordal graphs.

The results on k-good tree decomposition are inspired by a study of the so called *cops and robber games*. The aim of such a game is to capture a robber moving in a graph, using as few cops as possible. This problem has been intensively studied in the literature, allowing for a better understanding of the structure of graphs [BN11].

Section 3.2 studies the cops and robber problem. It is proved that $k-1$ cops are sufficient to capture a robber in k-chordal graphs (generalizing some results in [AF84, CN05]). Particularly, for 4-chordal graphs, 2 cops are enough.

Using these results, Section 3.3 provides a polynomial time algorithm that, given a n-vertex graph G and an integer $k \geq 3$, either returns an induced cycle of length at least $k+1$ in G or computes a k-good tree-decomposition of G. In the case when G admits a k-good tree decomposition, this ensures that G has treewidth at most $(k-1)(\Delta-1)+2$ (where Δ is the maximum degree), tree-length at most k and Gromov hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$. In particular, this shows that the treewidth of any k-chordal graph is upper-bounded by $O(k \cdot \Delta)$, improving the exponential bound of [BT97].

We study the complexities of the recognition and the chordality of the k-super-caterpillars in Section 3.4. We show that it is NP-complete to decide whether given a planar graph and an integer k this planar graph is a k-super-caterpillar or not. In the investigation of the chordality of the k-super-caterpillar, we propose an interesting conjecture: given a planar graph G which has a Hamilton path, it is NP-complete to decide whether there is a Hamilton cycle in G. We prove that the conjecture is true when G is general graph. If the conjecture is true, then we can prove that the chordality problem in planar k-caterpillars is also NP-complete. For a special subclass of k-super-caterpillars, in which the vertices except the backbone inducing a cycle, a dynamic programming algorithm is presented for computing the chordality.

3.1.1 Related Work

Chordality and hyperbolicity. Chordality and hyperbolicity are both parameters measuring "tree-likeness" of a graph. Some papers consider relations between them [BC03, WZ11]. In particular, the hyperbolicity of a k-chordal graph is at most k, i.e. the hyperbolicity of a graph is at most its chordality. But the gap, i.e. the difference between the two parameters, may be arbitrary large (take a $3 \times n$-grid). The seminal definition of hyperbolicity is the following. A graph G is δ-hyperbolic provided that for any vertices $x, y, u, v \in V(G)$, the two larger of the three sums $d(u,v)+d(x,y), d(u,x)+d(v,y)$ and $d(u,y)+d(v,x)$ differ by at most 2δ [Gro87]. With this definition, it is proved that any graph with tree-length at most k has hyperbolicity at most k [CDE$^+$08]. This definition is equivalent to that of Gromov hyperbolicity (mentioned at the beginning of the introduction), which we use in

this chapter, up to a constant ratio [AJ13]. There is an $O(n^{3.69})$ algorithm to compute hyperbolicity in [FIV12]. The problem of deciding whether the chordality of a graph G is at most k is NP-complete if k is as part of the input. Indeed, if G' is obtained by subdividing all the edges in G once, then there is an induced cycle of length $2|V(G)|$ in G' if and only if G has a Hamilton cycle. It is NP-complete to decide whether there is a Hamilton cycle in the class of planar graphs [GJT76]. So the problem of computing the chordality is also NP-complete in the class of planar graphs. It is coNP-hard to decide whether an n-vertex graph G is k-chordal for $k = \Theta(n)$ [Ueh99].

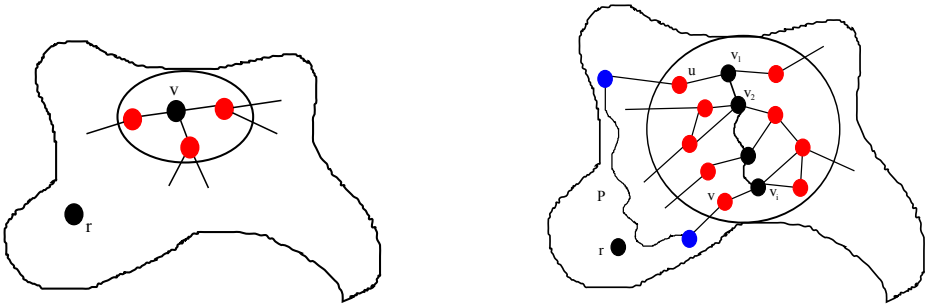
There are several problems related to chordality are considered. Finding the longest induced path is $W[2]$-complete [CF07]. In [KK09], the problem of deciding whether there is an induced cycle passing through k given vertices is studied. This problem is NP-Complete in planar graphs when k is part of the input and in general graphs even for $k = 2$. However, this problem is FPT in planar graphs with parameter k, the number of the given vertices. Finding an induced cycle of size exactly k in *d-degenerate graph* (every induced subgraph has a vertex of degree at most d) is FPT if k and d are fixed parameters [CCC06]. Note that, any planar graph is 5-degenerate. Recently, Coudert and Ducoffe proved that the problem of deciding whether an unweighted graph has hyperbolicity $1/2$ is subcubic equivalent to the problem of determining whether there is an induced cycle of size 4 in a graph [CD14].

Treewidth. Although, it is NP-complete to decide whether the treewidth of a graph G is at most k in general graphs [ACP87], there are polynomial algorithms for (4-)chordal graphs, cographs [BM93], circular arc graphs [SSR94], chordal bipartite graphs [KK95] and etc. Bodlaender and Thilikos proved that the treewidth of a k-chordal graph for ($k \geq 4$) with maximum degree Δ is at most $\Delta(\Delta-1)^{k-3}$ which implies that treewidth is polynomially computable in the class of graphs with chordality and maximum degree bounded by constants [BT97]. They also proved that the treewidth problem is NP-complete for graphs with small maximum degree [BT97].

3.1.2 Notations and Definitions

Throughout this chapter, given a graph $G = (V,E)$ and a vertex set $U \subset V$, we put $N_G[U] = \cup_{u\in U} N_G[u]$ and $N_G(U) = N_G[U] \setminus U$. If the context is clear for graph G, then we use $N(v)$ instead of $N_G(v)$ and similarly for $N[v]$, $N(U)$ and $N[U]$. Given two paths $P = (p_1,\ldots,p_k)$ and $Q = (q_1,\ldots,q_r)$, we denote their concatenation by (P,Q) the path induced by $V(P)\cup V(Q)$; to make descriptions more concise, we omit the detail of reversing P or Q if necessary.

A graph H is called *k-super-caterpillar* if there is a dominating set $B \subseteq V(H)$ in H such that $|B| \leq k-1$ and $H[B]$ is an induced path; this dominating path is called *backbone* of H. A tree decomposition $(T,\mathcal{X})$ of a graph G is called a *k-good tree decomposition* for an integer $k \geq 2$ if each bag of $\mathcal{X}$ induces a k-super-caterpillar in G.



(a) All the cops are placed on v. The robber is on r.

(b) The induced subgraph $(v_1,\dots,v_i,v,P,u)$ is a chordless cycle $\geq i+2$. The robber is on r.

Figure 3.1: illustration for the proof of Theorem 6

3.2 A detour through Cops and Robber games

In this section, we study the cops and robber games introduced by Winkler and Nowakowski [NW83], independently defined by Quilliot [Qui85]. Given a graph G, a player starts by placing $k \geq 1$ cops on some vertices of G, then a visible robber is placed on one vertex of G. Alternately, the cop-player may move each cop along one edge, and then the robber can move to an adjacent vertex. The robber is captured if, at some step, a cop occupies the same vertex.

Aigner and Fromme introduced the notion of *cop-number* of a graph G, i.e., the fewest number of cops $cn(G)$, such that there exists a strategy for the cop-player that assures to capture the robber whatever he does [AF84]. A long standing conjecture due to Meyniel states that $cn(G) = O(\sqrt{n})$ for any n-vertex graph G [Fra87]. To tackle this question, many researchers have focused on particular graph classes and provided many nice structural results (see the recent book [BN11]). For any n-vertex graph G, $cn(G) = O(\frac{n}{2^{(1-o(1))\sqrt{\log n}}})$ [LP12, SS11], $cn(G) \leq 3$ in any planar graph G [AF84], $cn(G) \leq 3 + \frac{3}{2}g$ in any graph G with genus at most g [Sch01], $cn(G) = O(m)$ in any graph G excluding a m-edge graph as a minor [And86], etc. Bounded hyperbolicity graphs have been considered in [CCNV11]. The cop number of graphs with minimum degree d and smallest induced cycle (girth) at least $8t-3$ is known to be $\Omega(d^t)$ [Fra87]. Strangely, little is known related to the largest induced cycle (chordality): in [AF84], it is shown that $cn(G) \leq 3$ for any 2-connected 5-chordal graph G. In this section, we consider the class of k-chordal graphs.

Theorem 6. *Let $k \geq 3$. For any k-chordal connected graph G, $cn(G) \leq k-1$, and there exists a strategy where all $k-1$ cops always occupy a chordless path except for the move that captures the robber.*

Proof. Let $v \in V$ be any vertex and place all cops at it (see in Fig. 3.1(a)). Then, the robber chooses a vertex. Now, at some step, assume that the cops are occupying $\{v_1,\cdots,v_i\}$ which induce a chordless path, $i \leq k-1$, and it is the turn of the cops (initially $i=1$). Let $N = \bigcup_{1\leq j\leq i} N[v_j]$, if the robber occupies a vertex in N, it is captured during the next move.

Else, let $R \neq \emptyset$ be the connected component of $G \setminus N$ occupied by the robber. Finally, let S be the set of vertices in N that have some neighbor in R. Clearly, since R is non-empty, so is S.

Now, there are two cases to be considered.

- If $N(v_1) \cap S \subseteq \bigcup_{1<j\leq i} N[v_j]$. This case may happen only if $i > 1$. Then,"remove" the cop(s) occupying v_1. That is, the cops occupying v_1 go to v_2. Symmetrically, if $N(v_i) \cap S \subseteq \bigcup_{1\leq j<i} N[v_j]$, then the cops occupying v_i go to v_{i-1}. Then, the cops occupy a shorter chordless path while the robber is still restricted to R.

- Otherwise, there is $u \in (N(v_1) \cap S) \setminus (\bigcup_{1<j\leq i} N[v_j])$ and $v \in (N(v_i) \cap S) \setminus (\bigcup_{1\leq j<i} N[v_j])$. First, we show that this case may happen only if $i < k-1$. Indeed, let P be a shortest path between such u and v with all internal vertices in R (possibly, P is reduced to an edge). Such a path exists by definition of S. Then $(v_1, \cdots, v_i, v, P, u)$ is a chordless cycle of length at least $i+2$ (See in Fig. 3.1(b)). Since G is k-chordal, this implies that $i+2 \leq k$.

  Then one cop goes to $v_{i+1} := v$ while all the vertices in $\{v_1, \cdots, v_i\}$ remain occupied. Since $v \in S$, it has some neighbor in R, and then, the robber is restricted to occupy R', the connected component of $G \setminus (N \cup N[v])$ which is strictly contained in R.

Therefore, proceeding as described above strictly reduces the area of the robber (i.e., R) after $< k$ steps, R decreases progressively and the robber is eventually captured. □

Note that previous Theorem somehow extends the model in [CN05], where the authors consider the game when two cops always remaining at distance at most 2 from each other must capture a robber. It is possible to improve the previous result in the case of 4-chordal graphs, i.e. $k = 4$. In the following theorem, we prove that $cn(G) \leq 2$ for any 4-chordal connected graph G.

Theorem 7. *For any 4-chordal connected graph G, $cn(G) \leq 2$ and there always exists a winning strategy for the cops such that they are always at distance at most one from each except for the move that captures the robber.*

Proof. Initially, place the cops on any two adjacent vertices. At some step of the strategy, let us assume that the cops are on two adjacent vertices a and b (or $a = b$) and it is the turn of the cops. If the robber stands at some vertex in $N = N[a] \cup N[b]$, then it is captured during the next move. Hence, let R be the connected component of $G \setminus N$ where the robber stands. Let $S \subseteq N$ be the set of the vertices adjacent to a or b and at least one vertex of R, i.e., S is an inclusion-minimal separator between $\{a,b\}$ and R.

We will prove that there is $z \in \{a,b\}$ and a vertex c in $S \cap N(z)$, such that, $S \subset N[z] \cup N[c] = N'$. Since $c \in S$, $N(c) \cap V(R) \neq \emptyset$. Hence, if the cops move from a,b to c,z, which can be done in one step, then the robber is constrained to occupy a vertex of R' where R' is the connected component of $G \setminus N'$ which is strictly contained in R. Note that, R' is a proper subgraph of R. Iterating such moves, the robber will eventually be captured.

It remains to prove the existence of $z \in \{a,b\}$ and $c \in S \cap N(z)$, such that, $S \subseteq N[z] \cup N[c]$.

- If there is $z \in \{a,b\}$ such that $S \subseteq N[z]$, then any vertex in $N(z) \cap S$ satisfies the requirements.
- Else, let $c \in S \setminus N(b)$ (such a vertex exists because otherwise we would be in the previous case). Clearly, $S \cap N(a) \subseteq N[a] \cup N[c]$. Now, let $x \in S \setminus N(a)$. By definition of S, there is a path P from x to c with internal vertices in R. Moreover, all internal vertices of P are at distance at least two from a and b; also c is not adjacent to b and x is not adjacent to a. Hence, considering the cycle a,b,x,P,c, there must be an edge between x and c because G is 4-chordal. So $S \setminus N(a) \subset N(c)$. Therefore, $S = (S \cap N(a)) \cup (S \setminus N(a)) \subseteq N[a] \cup N[c]$.

The bound provided by this theorem is tight because of the cycle with 4 vertices. □

Theorem 6 relies on chordless paths P in G such that $N[V(P)]$ is a separator of G, i.e., there exist vertices a and b of G such that all paths between a and b intersect $N[V(P)]$. In the next section, we show how to adapt this to compute particular tree-decompositions.

3.3 Structured Tree-decomposition

In this section, we present our main contribution, proving the following theorem:

Theorem 8. *Given an m-edge-graph G and an integer $k \geq 3$, there is an $O(m^2)$-algorithm which:*

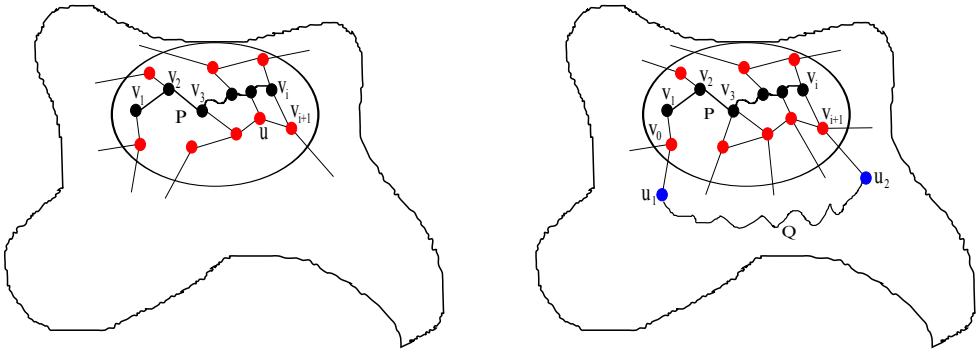
- *either returns an induced cycle of length at least $k+1$;*
- *or returns a k-good tree-decomposition of G.*

Proof. The proof is by induction on $|V(G)| = n$. We prove that either we find an induced cycle larger than k, or for any chordless path $P = (v_1, \ldots, v_i)$ with $i \leq k-1$, there is a k-good tree-decomposition for G with one bag containing $N_G[V(P)]$. Note that the later case does not mean that a large induced cycle does not exist. Obviously, it is true if $|V(G)| = 1$. Now we assume that it is true for any graph G with n' vertexs, $1 \leq n' < n$, and we show it is true for n-vertex graphs.

Let G be a connected n-vertex graph, $n > 1$. Let $P = (v_1, \ldots, v_i)$ be any chordless path with $i \leq k-1$ and let $N = N_G[V(P)]$ and $G' = G \setminus N$. There are three cases to be considered:

Case 1. Let $G' = \emptyset$. In this case, we have $V(G) = N$. The desired tree-decomposition consists of one node, corresponding to the bag N.

Case 2. Let G' be disconnected. Let $C_1, \ldots, C_r$, $r \geq 2$, be the connected components of G' For any $j \leq r$, let G_j be the graph induced by $C_j \cup N$. Note that any induced cycle in G_j, for any $j \leq r$, is an induced cycle in G. By the induction hypothesis, either there is an induced cycle $\mathcal{C}$ larger than k in G_j, then $\mathcal{C}$ is also an induced cycle larger than k in G, or our algorithm computes a k-good tree-decomposition TD_j of G_j with one bag X_j containing N. To obtain the k-good tree-decomposition of G, we combine the TD_j's, for $j \leq r$, by adding a bag $X = N$ adjacent to all the bags X_j for $j = 1, \ldots, r$. It is easy to see that this tree-decomposition satisfies our requirements.



(a) The vertex $u \in N_G(V(P))$ and all the neighbors of u are the neighbors of P.

(b) The induced subgraph $(P, v_{i+1}, u_2, Q, u_1, v_0)$ is an induced cycle with at least $i+2 \geq k+1$ vertices in G.

Figure 3.2: illustration for the proof of Theorem 8

Case 3. Let G' be connected. We consider the order of the path $P = (v_1, \ldots, v_i)$. In the following proof, first we prove that if the order of path P, $i = k-1$, then we can find either an induced cycle larger than k or the required tree-decomposition for G. Subsequently, we prove it is also true for path with order $i < k-1$ by backward induction on i. More precisely, if $i < k-1$, either we find directly the desired cycle or tree-decomposition, or we show that there exists a vertex v_{i+1} such that $P \cup \{v_{i+1}\}$ induces a chordless path P' of order $i+1$. By backward induction on i we can find either an induced cycle larger than k or a k-good tree-decomposition of G with one bag containing $N_G[V(P')] \supseteq N_G[V(P)]$.

(a) If $i = k-1$, then we consider the following two cases. Note that $k \geq 3$, so $i \geq 2$ and then $v_1 \neq v_i$.

- Assume first that there is $u \in N_G(V(P)) \cup \{v_1, v_i\}$ (in particular, $u \notin P \setminus \{v_1, v_i\}$) such that $N_G(u) \subseteq N_G[V(P) \setminus \{u\}]$ (See in Fig. 3.2(a)). Let $\tilde{G} = G \setminus \{u\}$. Then $\tilde{G}$ is a graph with $n' = n-1$ vertices. By the induction hypothesis on $n' < n$, the algorithm either finds an induced cycle larger than k in $\tilde{G}$, then it is also the one in G; Otherwise our algorithm computes a k-good tree-decomposition $\widetilde{TD}$ of $\tilde{G}$ with one bag $\tilde{X}$ containing $N_{\tilde{G}}[V(P) \setminus \{u\}]$. To obtain the required tree-decomposition of G, we just add vertex u into the bag $\tilde{X}$. The tree-decomposition is still k-good.
- Otherwise, there exist two distinct vertices $v_0 \in N_G(v_1) \setminus N_G(V(P) \setminus v_1)$ and $v_{i+1} \in N_G(v_i) \setminus N_G(V(P) \setminus v_i)$, since $v_1 \neq v_i$; and there are vertices $u_1, u_2 \in V(G')$ (possibly $u_1 = u_2$) such that $\{v_0, u_1\} \in E(G)$ and $\{v_{i+1}, u_2\} \in E(G)$ (See in Fig. 3.2(b)). If $\{v_0, v_{i+1}\} \in E(G)$, (P, v_0, v_{i+1}) is an induced cycle with $k+1$ vertices. Otherwise, let Q be a shortest path between u_1 and u_2 in G' (Q exists since G' is connected). So $(P, v_{i+1}, u_2, Q, u_1, v_0)$ is an induced cycle with at least $k+1$ vertices in G.

(b) If $i < k-1$, we proceed by backward induction on i. Namely, assume that, for

any chordless path Q with $i+1$ vertices, our algorithm either finds an induced cycle larger than k in G or computes a k-good tree-decomposition of G with one bag containing $N[V(Q)]$. Note that the initialization of the induction holds for $i = k-1$ as described in case (a). We show it still holds for a chordless path with i vertices. We consider the following two cases.

- Either there is $u \in N_G(V(P)) \cup \{v_1, v_i\}$ (in particular, $u \notin P \setminus \{v_1, v_i\}$) such that $N_G(u) \subseteq N_G[V(P) \setminus \{u\}]$. That is, we are in the same case as the first item of (a). We proceed as above and the result holds by induction on n.
- Or there is $w \in (N_G(v_1) \cup N_G(v_i)) \setminus V(P)$ such that (P,w) is chordless (i.e., the vertex w is a neighbor of v_1 or v_i but not both and $w \notin N_G(V(P) \setminus \{v_1, v_i\})$). Therefore, we apply the induction hypothesis (on i) on $P' = (P,w)$. By the assumption on i, either our algorithm returns an induced cycle larger than k or it computes a k-good tree-decomposition of G with one bag containing $N_G[V(P')] \supseteq N_G[V(P)]$.

Now we describe the algorithm and study its complexity. Let G be an m-edge n-vertex graph with maximum degree Δ. Roughly speaking, the algorithm proceeds by steps. At each step, one vertex is considered and the step takes $O(m)$ time. We prove that at each step (but the initial one), at least one edge will be *considered* and that each edge is considered at most once (but one vertex may be considered several times). This implies a time-complexity of $O(m^2)$ for the algorithm.

The algorithm starts from an arbitrary vertex $v \in V(G)$ and computes the connected components $C_1, \cdots, C_j$ of $G \setminus N[v]$ ($j \geq 1$) in time $O(m)$ [HT73]. We start with the k-good tree-decomposition for the induced graph of $N[v]$ in G that consists of one bag $B = N[v]$ adjacent to, for any $i \leq j$, each bag $B_i = \{v\} \cup \{w \in N(v) \ : \ N(w) \cap C_i \neq \emptyset\}$. This takes time $O(m)$.

Now, at some step of the strategy, assume that we have built a k-good tree-decomposition $(T, \mathcal{X})$ of a connected subgraph G_0 of G. Let $C_1, \cdots, C_j$ ($j \geq 1$) be the connected components of $G \setminus G_0$, and, for any $i \leq j$, let S_i be the set of the vertices of G_0 that are adjacent to some vertex of C_i. Assume finally that, for any $i \leq j$, there is a leaf bag $B_i \supset S_i$ of $(T, \mathcal{X})$ where $P_i = B_i \setminus S_i$ is a chordless path dominating B_i and has minimum number of vertices.

For any $e \in E(G)$, we say that $e = \{x, y\}$ is *alive* if there is $i \leq j$ such that $x \in S_i \cup C_i$ and $y \in C_i$. Note that, if an edge is alive, such an integer i is unique. An edge that is not alive is said *dead*. Note also that, after the initial step, all edges in the bag B are dead and other edges are alive.

The next step consists of the following. Choose any $i \leq j$ and let w be any vertex of S_i such that $Q = P_i \cup \{w\}$ is a chordless path. (Such w exists because P_i is the dominating path with the minimum order. Suppose $P_i = \{v_1, \ldots, v_l\}$. If $N_G(v_1) \setminus V(P_i) = \emptyset$, then the chordless path $P_i \setminus v_1$ dominates B_i and has less vertices than P_i. So $N_G(v_1) \setminus V(P_i) \neq \emptyset$. If any $w \in N_G(v_1) \setminus V(P_i)$ is a neighbor of some vertices in P_i, then the chordless path $P_i \setminus v_1$ dominates B_i and has less vertices than P_i.) Note that by definition of S_i, there is at least one edge from w to C_i and that such an edge is alive before this step. We add the bag $B' = Q \cup B_i \cup (N(w) \cap C_i)$ adjacent to B_i. If Q is larger than k, by the above

proof, the algorithm finds a large cycle. Otherwise, the connected components $C'_1, \cdots, C'_r$ of $C_i \cup B_i \setminus B'$ are computed in time $O(m)$. Let S'_h, $h \leq r$, be the subset of the vertices of S_i that are adjacent to some vertex in C'_h, and let Q_h be the smallest subpath of Q dominating S'_h. Computing the sets $S'_1, \cdots, S'_r$ only requires a time $O(m)$ since we have only to check the edges in B'. For any $h \leq r$, add a bag $B'_h = Q_h \cup S'_h$ adjacent to B'.

One can check that this algorithm follows the above proof and that it eventually computes the desired tree-decomposition or returns a large cycle.

To conclude, we can check that the set of edges alive after one step is contained in the set of edges alive before this step, and that, at each step at least one edge (the one(s) from w to C_i) becomes dead. Therefore, at each step, the number of alive edges strictly decreases and the algorithm terminates when there are no more. Since each step takes time $O(m)$ and there are at most m steps, the result follows. □

The following two theorems discuss some properties of the graphs with k-good tree decompositions.

Theorem 9. *Let G be a graph that admits a k-good tree-decomposition. Let Δ be the maximum degree of G. Then $tw(G) \leq (k-1)(\Delta-1)+2$ and $tl(G) \leq k$.*

Proof. It directly follows the fact that, in a k-good tree-decomposition, each bag has a dominating path with $< k$ vertices. □

Recall that a graph G has Gromov hyperbolicity $\leq \delta$ if, for any $u, v, w \in V(G)$ and any shortest paths P_{uv}, P_{vw}, P_{uw} between these three vertices, any vertex in P_{uv} is at distance at most δ from $P_{vw} \cup P_{uw}$. In the next theorem, we prove that the Gromov hyperbolicity of the graph admitting a k-good tree-decomposition is at most $\lfloor \frac{3}{2}k \rfloor$.

Notice that the result given in [CDE+08] refers to the seminal hyperbolicity and does not imply our result for Gromov hyperbolicity.

Theorem 10. *Any graph G that admits a k-good tree-decomposition has Gromov hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$.*

Proof. Let $G = (V, E)$ be a graph that admits a k-good tree-decomposition $(\{X_i | i \in I\}, T = (I, M))$. Let T be rooted at bag X_0, $0 \in I$. For any $u, v \in V$, let us denote the distance between u and v in G by $d(u, v)$. By definition of a k-good decomposition, for any $i \in I$ and for any $u, v \in X_i$, we have $d(u, v) \leq k$.

Let $x, y, z \in V$ and let P_1, P_2, P_3 be any three shortest paths in G between x and y, y and z, x and z respectively. Let $u \in P_1$. To prove the Theorem, we show that there is $v \in P_2 \cup P_3$ such that $d(u, v) \leq \lfloor \frac{3}{2}k \rfloor$.

First, let us assume that there is $i \in I$ such that $u \in X_i$ and there is $v \in (P_2 \cup P_3) \cap X_i \neq \emptyset$. In that case, $d(u, v) \leq k$ and the result holds.

Otherwise, let T_u be the subtree of T induced by $\{i \in I \ : \ u \in X_i\}$. Similarly, let T_x be the subtree of T induced by $\{i \in I \ : \ x \in X_i\}$ and T_y be the subtree of T induced by $\{i \in I \ : \ y \in X_i\}$. Let P be the path in T between T_x and T_y. Note that P may be empty if $V(T_x) \cap V(T_y) \neq \emptyset$. Let $j \in V(T_x) \cup V(T_y) \cup P$ that is closest to T_u in T. If $j \in V(T_u)$, then X_j is a separator between x and y or $x \in X_j$ or $y \in X_j$. If $x \in X_j$ or $y \in X_j$, then we are in

the first case above; otherwise we have X_j is a separator between x and y. Then $z \in X_j$ or z cannot be in both the component of $G \setminus X_j$ containing x and of the one containing y. So one of the paths P_2 or P_3 should pass trough X_j and we are in the the first case again.

Assume that $j \notin V(T_u)$, then we have that either X_j is a separator between x and u or $x \in X_j$, and that either X_j is a separator between y and u or $y \in X_j$. Let P_{xu} and P_{uy} be the subpaths of P_1 from x to u and from u to y respectively. By remark above, there exist vertices $w \in P_{xu} \cap X_j$ and $t \in P_{uy} \cap X_j$. Possibly, $w = t$. Then $d(w,u) + d(u,t) = d(w,t)$ because P_1 is a shortest path, therefore, $d(w,u) + d(u,t) = d(w,t) \leq k$. So there is $\ell \in X_j$ with $d(u,\ell) \leq \lfloor \frac{k}{2} \rfloor$.

Finally, let us show that there is $h \in (P_2 \cup P_3) \cap X_j$. If $x \in X_j$ or $y \in X_j$ or $z \in X_j$, it is obvious. Otherwise, z cannot be in both the component of $G \setminus X_j$ containing x and of the one containing y, because X_j separates x and y in G. Therefore one of the paths P_2 or P_3 should pass trough X_j.

To conclude, $d(u,h) \leq d(u,\ell) + d(\ell,h) \leq \lfloor \frac{k}{2} \rfloor + k \leq \lfloor \frac{3}{2}k \rfloor$. □

From the above theorems, it is easy to get the following corollaries.

Corollary 7. *Any k-chordal graph G with maximum degree Δ has treewidth at most $(k-1)(\Delta-1)+2$, tree-length at most k and Gromov hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$.*

Proof. By definition of k-chordal graph and Theorem 8, any k-chordal graph admits a k-good tree-decomposition. The result follows from Theorems 9 and 10. □

Corollary 8. *There is an algorithm that, given an m-edge graph G and $k \geq 3$, states that either G has chordality at least $k+1$ or G has Gromov hyperbolicity at most $\lfloor \frac{3}{2}k \rfloor$, in time $O(m^2)$.*

3.4 Recognition and Chordality of Planar k-Super-Caterpillars

In this section, we investigate the recognition of the k-super-caterpillars for an integer $k \geq 1$. We show that their recognition is NP-complete even when restricted in the class of planar graphs. Moreover, we study the problem of computing the chordality of any k-super-caterpillar.

3.4.1 NP-completeness of the recognition of planar k-super-caterpillars

Let us define the recognition problem of planar k-super-caterpillars as follows:

RECOGNITION OF PLANAR k-SUPER-CATERPILLARS
Instance: A planar graph G and a positive integer k.
Question: Is G a k-super-caterpillar?

We prove the NP-completeness of RECOGNITION OF PLANAR k-SUPER-CATERPILLARS by doing a reduction from HAMILTON PATH OF PARTIAL GRIDS problem [IPS82]. Obviously, it is in NP, because a dominating induced path smaller than k of a graph G is a certification that G is a k-super-caterpillar. We show now that it is NP-hard.

Theorem 11. RECOGNITION OF PLANAR k-SUPER-CATERPILLARS *is NP-hard.*

Proof. Given a planar graph G, we construct a new planar graph H by subdividing every edge $e \in E(G)$, i.e., adding a new vertex a_e for every edge $e = uv$ of G and replacing uv by a path ua_ev. In the following, we prove that:

Claim 2. *There is a Hamilton path in G if and only if H is a planar k-super-caterpillar for some $k > 0$.*

If G has a Hamilton path, then after the subdivision of each edge, the Hamilton path in G becomes an induced path P in H of order $k-1 = 2n-1$, $n = |V(G)|$. Every $v \in V(G)$ is in P and every a_e is a neighbor of some $v \in V(G)$. So every vertex in H is either in P or adjacent to some vertex in P. That is H is planar k-super-caterpillar.

Suppose now that H is a planar k-super-caterpillar, for some $k > 0$. Let $P = \{b_1b_2\dots b_p\}$, $p \leq k-1$ be a dominating induced path in H. From the construction of H, we see that every vertex $v \in V(G)$ is exactly adjacent to a_e for each e incident to v; for any $e = uv \in E(G)$, every a_e has degree 2 only adjacent to u and v. Equivalently, the two neighbors of a_e for any $e \in E(G)$ are adjacent in G. So the vertices in P are arranged alternately one vertex in G and one vertex $a_e \notin V(G)$ for some $e \in E(G)$. If every $v \in V(G)$ is in P, then suppose $\{b_i, b_{i+2}, \dots, b_{i+2(n-1)}\} = V(G)$ for $i = 1$ or 2. Indeed, if $b_1 \in V(G)$, then $i = 1$; otherwise, $i = 2$. So each $a_e \in \{b_{i+1}, b_{i+3}, \dots b_{i+2(n-1)-1}\} \cap V(G) = \emptyset$ has two neighbors in P, which are adjacent in G. Then $b_ib_{i+2}\dots b_{i+2(n-1)}$ for $i = 1$ or 2 is a Hamilton path in G because for any $i \leq j \leq i+2(n-2)$, $b_jb_{j+2} \in E(G)$. Otherwise, let $u \in V(G) \setminus P$. Then one neighbor a_e of u is in P for some e incident to u in G. Since a_e has degree 2 in H and one of its neighbor u is not in P, a_e has only one neighbor in P. So a_e is an endpoint in P. There are at most two endpoints in P, so, at most two vertices $u, w \in V(G)$ are not in P. Then, we take $P' = uPw$, if $u \neq w$, or $P' = uP$, otherwise, as a dominating induced path in H. Then we are in the previous case, in which it is proved that G has a Hamilton path. □

From the above proof, we see that given an n-vertex planar graph G with a Hamilton path, we can create a $2n$-planar super-caterpillar by subdividing all the edges in G. This is useful for proving the NP-completeness of deciding the chordality of any k-super-caterpillars in the next subsection.

3.4.2 Chordality of planar k-super-caterpillars

In this subsection, we study the chordality of planar k-super-caterpillars.

From the proof of the Theorem 11, we see that the chordality of a planar k-super-caterpillar is closely related with the following problem:

HAMILTON CYCLE WITH HAMILTON PATH(HCHP)
Instance: A graph G, which has a Hamilton Path.
Question: Is there a Hamilton Cycle in G?

Claim 3. *If the* HAMILTON CYCLE WITH HAMILTON PATH *problem is NP-complete in the class of planar graphs, then it is NP-complete to decide whether the chordality of a given planar k-super-caterpillar is at least k.*

Proof. Given a planar graph G with a Hamilton path and $|V(G)| = n$, create a planar graph H by subdividing all edges in G. Then from the proof of Theorem 11, H is a $2n$-super-caterpillar because G has a Hamilton path. Then there is a Hamilton cycle in G if and only if there is an induced cycle of size at least $2n$ in H, equivalently the chordality of H is at least $2n$. □

So we propose the following conjecture.

Conjecture 1. *The* HAMILTON CYCLE WITH HAMILTON PATH *problem is NP-complete in the class of planar graphs.*

In the following, we prove that the HAMILTON CYCLE WITH HAMILTON PATH problem is NP-complete for general directed graphs by reduction from EXACT COVER problem. The proof is similar with the one in [BM08], which proved that it is NP-complete to decide whether there is a directed hamilton cycle in a given directed graph.

Let $\mathcal{F}$ be a family of subsets of a finite set U. An *exact cover* of U by $\mathcal{F}$ is a partition of U, each set of which belongs to $\mathcal{F}$ [BM08].

EXACT COVER (EC)
Instance: A set U and a family $\mathcal{F}$ of subset of U.
Question: Is there an exact cover of X by $\mathcal{F}$?

Lemma 23. *[BM08] Exact Cover problem is NP-complete.*

Theorem 12. *The* HAMILTON CYCLE WITH HAMILTON PATH *problem is NP-complete in the class of directed graph.*

Proof. Given a set $U = \{e_1, \ldots, e_m\}$ and a family $\mathcal{F} = \{S_1, \ldots, S_n\}$ of subsets of U, construct a digraph $G = (V, A)$ with a Hamilton path, which is only one arc different from the digraph constructed in [BM08] (page 185-188). For the sake of completeness, we describe it as follows. Let P be a directed path $u_1u_2 \ldots u_mu_{m+1}$ with each arc $(u_i, u_{i+1}) = e_i \in U$ for $i = 1, \ldots, m$. Let Q be another directed path $f_1f_2 \ldots f_nf_{n+1}$ with each arc $(f_j, f_{j+1}) = S_j$ for $j = 1, \ldots, n$. Add two arcs (u_1, f_1) and (f_{n+1}, u_{m+1}) between the two directed paths P and Q. For each element $e_i \in S_j$, $i = 1, \ldots, m$, $j = 1, \ldots, n$, let P_{ij} be a 'path' of length two whose edges are pairs of oppositely oriented arcs, i.e., $P_{ij} = b_{ij}c_{ij}d_{ij}$ with four arcs $(b_{ij}, c_{ij}), (c_{ij}, b_{ij}), (c_{ij}, d_{ij}), (d_{ij}, c_{ij})$. Take each P_{ij} for $e_i \in S_j$ as a vertex and connect them to be a directed path. Denote the obtained 'path' as D_j, $j = 1, \ldots, n$ (See in Figure 3.3). Add arcs from f_j to the initial vertex on D_j and arcs from the terminal vertex on D_j to f_{j+1}. Additionally, add arcs from b_{ij} to u_i and arcs from u_{i+1} to d_{ij} for each $e_i \in S_j$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Finally, add one more arc (u_{m+1}, u_1), which is not added in the proof in [BM08]. There is a Hamilton path from f_1 to u_m in G, which is $f_1D_1f_2 \cdots \cup f_nD_nf_{n+1} \cup \{f_{n+1}u_{m+1}u_1\} \cup (P \setminus \{u_{m+1}\})$. As proved in [BM08], the digraph G has a directed Hamilton cycle C if and only if the set U has an exact cover by the family of subsets $\mathcal{F}$. If C does not use the arc S_j, it is obliged to traverse D_j from its initial to its terminal vertex. Conversely, if C uses the arc S_j , it is obliged to include each one of the paths P_{ij} of D_j in its route from u_{m+1} to u_1. Moreover, C contains exactly one of the paths

P_{ij} ($e_i \in S_j$) in traveling from u_{i+1} to u_i. The arcs S_j of Q which are included in C therefore form a partition of U. Conversely, every partition of U corresponds to a directed Hamilton cycle of G. Further more, the numbers of vertices and arcs of G are linear functions of the size of the instance $(U, \mathcal{F})$. So the HAMILTON CYCLE WITH HAMILTON PATH problem for directed graph is NP-complete. □

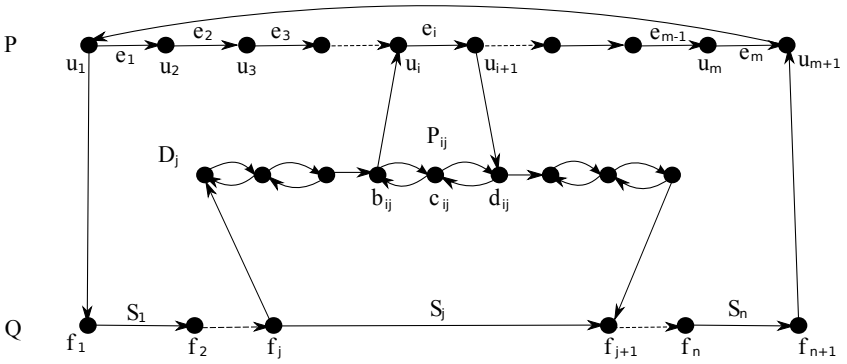


Figure 3.3: The digraph G constructed from the exact cover problem.

Corollary 9. *The* HAMILTON CYCLE WITH HAMILTON PATH *problem for general undirected graph is NP-complete.*

Proof. There is a classical transformation from directed graphs to undirected graphs. Given a digraph $D = (V, A)$, construct an undirected graph G as follows. Replace each vertex $v \in V$ with a path $v_{in}v_{mid}v_{out}$ and make each arc $(u, v) \in A$ (resp. $(v, u) \in A$) incident to v_{in} (resp. v_{out}). Change every arc to an edge, i.e., ignore the orientations of arcs. It is easy to check that there is a directed Hamilton cycle in D (resp. Hamilton path from u to v for $u, v \in V$) if and only if there is a Hamilton cycle in G (resp. Hamilton path from u_{in} to v_{out}). □

3.4.2.1 Properties of planar k-super-caterpillar

Let us prove some properties of planar k-super-caterpillars graphs in this subsection, which are going to be used for computing the chordality. In the rest of this section, let $G = (V, E)$ be a planar k-super-caterpillar graph with backbone $B = \{b_1 b_2 \dots b_p\}$, for $p < k$. Denote by O the subgraph induced by $G \setminus B$.

Lemma 24. *O is an outerplanar graph.*

Proof. Suppose by contradiction that O is not outerplanar. So, O has a K_4 or a $K_{2,3}$ as a minor. In G, contract the backbone B to a unique vertex v. Then v is adjacent to every other vertex in G. The K_4 ($K_{2,3}$) of O together with v is a K_5-minor ($K_{3,3}$-minor) of G. This is a contradiction with the fact that G is a planar graph. □

For a given planar k-super-caterpillar G, suppose that O is 2-connected. Then from Lemma 17 O is a cycle $\{v_1, v_2, \dots, v_l\}$ with some chords. Observe that if all neighbors of b_1 are neighbors of some vertex in $\{b_2, \dots, b_p\}$, we could consider the backbone as

$\{b_2,\ldots,b_p\}$, which is still an induced dominating path in G. So we can always suppose there exists a vertex v_1 that is a neighbor of b_1 but not a neighbor of any vertex in $\{b_2,\ldots,b_p\}$. Similarly, we suppose that v_q is a neighbor of b_p but it is not a neighbor of $\{b_1,\ldots,b_{p-1}\}$. We can prove the following property for graphs where O is 2-connected:

Fact 1. *Let v_j be a neighbor of b_i and $v_{j'}$ be a neighbor of $b_{i'}$, for $1 \leq j < j' \leq l$ and $1 \leq i,i' \leq p$. Then $i \leq i'$ when $1 \leq j < j' \leq q$, and $i > i'$, when $q < j < j' \leq l$.*

Proof. Suppose $i > i'$ when $1 \leq j < j' \leq q$. We contract the backbone to an edge $b_{i'}b_i$ and O to a cycle with vertex set $\{v_1, v_j, v_{j'}, v_q\}$. Then v_1 is adjacent to $b_{i'}$ and v_q is adjacent to b_i. With the two edges v_jb_i and $v_{j'}b_{i'}$, the obtained graph is a $K_{3,3}$, with one part $\{v_j, v_q, b_{i'}\}$ and another part $\{v_1, v_{j'}, b_i\}$. So, there is a $K_{3,3}$ as a minor of G. It is a contradiction with the fact that G is a planar graph. Similarly, we can prove that $i \geq i'$ when $q+1 \leq j < j' \leq l$. □

Before continuing, we give another notation. A *segment* in O is a set of consecutive vertices in the cycle $\{v_1, v_2, \ldots, v_l\}$. Take two vertices v_s, v_t for $1 \leq s < t \leq l$ as the two ends of a segment, then there are two possible segments, denoted as $\{v_s, v_{s+1}, \ldots, v_t\}$ and $\{v_t, v_{t+1}, \ldots, v_l, v_1, \ldots, v_s\}$. We say a segment $\{v_s, v_{s+1}, \ldots, v_t\}$ is *between v_1 and v_q* (resp. *between v_{q+1} and v_l*) if $1 \leq s < t \leq q$ (resp. $q+1 \leq s < t \leq l$). From the above Fact, we obtain:

Fact 2. *The neighbors of b_i in O, for $1 \leq i \leq p$, are at most two segments: one between v_1 and v_q and another one between v_{q+1} and v_l.*

Proof. If for any $1 \leq i \leq p$, b_i has at most one neighbor in O, then it is true. Suppose for $1 \leq s < t \leq q$, v_s and v_t are adjacent to b_i but v_{s-1} and v_{t+1} are not adjacent to b_i. Let v_r for $s < r < t$ be adjacent to b_x for some $1 \leq x \leq p$. Then from v_s adjacent to b_i and $1 \leq s < r \leq q$, we have $x \geq i$ by Fact 1; but from v_t adjacent to b_i and $1 \leq r < t \leq q$, we have $x \leq i$ by Fact 1. So $i = x$, i.e. $b_i = b_x$ and v_r for any $s < r < t$ is only adjacent to b_i. So take the first and the last neighbor of b_i in $\{v_1, \ldots, v_q\}$. The segment between this two vertices are all neighbors of b_i. Similarly we can prove that there is at most another one segment between v_{q+1} and v_l, which is a set of neighbors of b_i. □

3.4.2.2 Polynomial algorithm for chordality when O is an induced cycle

Let us now consider the chordality problem for planar k-super-caterpillars in the easy case in which O is an induced cycle, i.e., there is no chords in O. Recall that G is a planar k-super-caterpillar with backbone $B = \{b_1, \ldots, b_p\}$, for $p < k$, and O is an induced cycle $\{v_1, v_2, \ldots, v_l\}$. Let v_1 be a neighbor of b_1 but not a neighbor of b_2 and v_q a neighbor of b_p but not a neighbor of b_{p-1}.

Lemma 25. *If every vertex on the backbone has two segments neighbors, then O is a longest induced cycle in G.*

Proof. The proof is done by induction on $|B| = p$. Obviously, the lemma is true for $p = 1$. Assume that it is true for any k-super-caterpillar with backbone B such that $|B| < p$. Now we prove it is true for $|B| = p$.

Suppose O is not a longest induced cycle in G. Then any longest induced cycle C of G has to contain at least one vertex of the backbone. If C passes only by b_1 at the backbone, then it needs to passes by another (and only by) two neighbors v, v' of b_1. If v is adjacent to v', then C is a triangle. So O is also a longest induced cycle in G. Otherwise, since C does not contain $\{b_2, \ldots, b_p\}$, it passes by all their neighbors in O. Then from C we get O by replacing $\{vb_1v'\}$ in C with the segment in O from v to v', which has at least two edges (three vertices), since v and v' are non-adjacent. So, $|O| \geq |C|$. It can be proved similarly if C passes only b_p or C passes for both b_1 and b_p.

So, C passes by at least one b_i for some $1 < i < p$. Since every vertex on the backbone has two segments neighbors in O, let u (resp. u') be a neighbor of b_i in O between v_1 (resp. v_{q+1}) and v_q (resp. v_l). Then $\{u, b_i, u'\}$ is a separator of G. Denote the two connect components obtained by removing this separator as G_1 and G_2. The subgraph induced by $\{u, b_i, u'\}$ is denoted as S. Note that C is either in $G_1 \cup S$ or in $G_2 \cup S$ (or passes u and u' but not b_i, which is not the case). Suppose that C is in $G' = G_1 \cup S$. To transform G' to G'', a k-super-caterpillar with backbone $B = \{b_i, b_{i+1}, \ldots, b_p\}$, we add a new vertex v_0 adjacent to b_i, u and u'. So C is also an induced cycle in G''. By induction, the longest induced cycle in G'' is $O_{G''}$, which is the union of $\{uv_0u'\}$ and the segment from u to u' in $O \cap G_1$. So, $|C| \leq |O_{G''}| \leq |O|$, and O is the longest induced cycle in G. □

Now we consider the general case in which each vertex on the backbone have at most two segment of neighbors in the induced cycle O. For easy description, we see some notations first. For $1 \leq i \leq p$, let G_i be the subgraph of G induced by $\{b_1, \ldots, b_i\}$ and all their neighbors in O. Denote by u_i (resp. w_i) the vertex v_j in $G_i \cap O$ with the maximum (resp. minimum) j between 1 and q (resp. $q+1$ and l). Call the segment of neighbors of b_i between v_1 and v_q (resp. v_{q+1} and v_l) its *up neighborhood* (resp. *down neighborhood*), denoted by $upN(b_i)$ (resp. $downN(b_i)$). If $upN(b_i) \neq \emptyset$ (resp. $downN(b_i) \neq \emptyset$) denote by f_i (resp. g_i) the smallest (resp. biggest) neighbor of b_i. Observe that u_i (resp. w_i) is not always a neighbor of b_i, since it is possible that b_i has no neighbors between v_1 and v_q (resp. v_{q+1} and v_l), the case in which $u_i = u_{i-1}$ (resp. $w_i = w_{i-1}$). See Figure 3.4 for the visual explanations of the defined vertices.

We are going to present a dynamic programming algorithm for finding a longest induced cycle in G. For $1 \leq i \leq p$, in each step i, we consider the induced subgraph G_i, which is induced by $\{b_1, \ldots, b_i\}$ and their neighbors in O. We compute a longest induced cycle in G_i and if $i \neq p$ the longest induced paths in G_i between every two vertices in $\{u_i, b_i, w_i\}$. Because for $1 \leq i \leq p-1$, $\{u_i, b_i, w_i\}$ is a separator of G_i and G_{i+1}, the induced paths and cycles in G_{i+1} can be computed based on the ones in G_i. In the following, we give the detailed algorithm.

In order to simplify the understanding, we are going to define a subroutine for each parameter calculated on the main algorithm. Given two paths P and Q, denote $\max\{P, Q\}$ as the longer path of P and Q in the following; denote their concatenation by (P, Q) the path induced by $V(P) \cup V(Q)$; to make descriptions more concise, we omit the detail of reversing P or Q if necessary. Given two vertices $u, w \in V(G)$, let $P(u, w)$ be an induced path between u and w. Given three vertices $u, w, b \in V(G)$, let $P(u, w, \bar{b})$ be an induced path between u and w not passing through b. For $v_s, v_t \in O$, $1 \leq s \leq t \leq l$, denote $O[v_s, \ldots, v_t]$ as

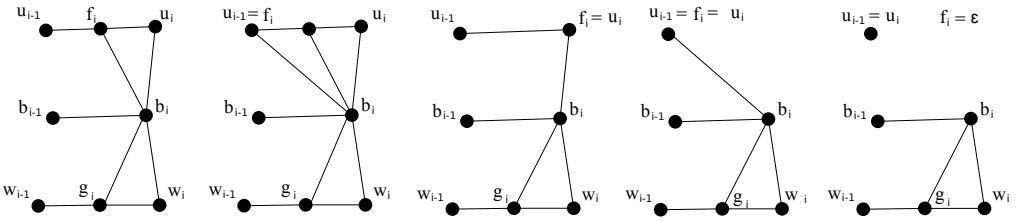


Figure 3.4: The special defined neighbors f_i, g_i, u_i, w_i of b_i. $f_i = \varepsilon$ means f_i does not exist since $upN(b_i) = \emptyset$

the induced path $v_s v_{s+1} \dots v_{t-1} v_t$ in O from v_s to v_t.

Algorithm 1 computes a longest induced path between u_i and b_i in G_i. A longest induced path between w_i and b_i in G_i is calculated similarly.

Algorithm 1: Longest induced path $LIP(u_i, b_i)$

Input: G_i and a longest induced path $P(u_{i-1}, b_{i-1})$ between u_{i-1} and b_{i-1} in the subgraph G_{i-1}, also a longest induced path $P(u_{i-1}, w_{i-1}, \bar{b}_{i-1})$ (resp. $P(u_{i-1}, b_{i-1}, \bar{w}_{i-1})$) between u_{i-1} and w_{i-1} (resp. b_{i-1}) not passing through b_{i-1} (resp. w_{i-1}) in the subgraph G_{i-1}

Output: $P(u_i, b_i)$

if $upN(b_i) = \emptyset$ *and* $downN(b_i) = \emptyset$ **then**
  $P(u_i, b_i) \leftarrow (P(u_{i-1}, b_{i-1}), b_i)$
else if $upN(b_i) \neq \emptyset$ **then**
  $P(u_i, b_i) \leftarrow (u_i, b_i)$
else if $b_i w_{i-1} \in E(G_i)$ **then**
  $P(u_i, b_i) \leftarrow max\{(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), b_i), (P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i)\}$
else
  $P(u_i, b_i) \leftarrow max\{(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), g_i, b_i), (P(u_{i-1}, b_{i-1}), b_i)\}$

Lemma 26. *Algorithm 1 outputs a longest induced path $P(u_i, b_i)$ between u_i and b_i in G_i.*

Proof. Recall that G_i is the graphs induced by $V(G_{i-1})$, b_i and its neighbors in O. If the up neighborhood and the down neighborhood of b_i are empty, then $u_i = u_{i-1}$ and b_{i-1} is the only vertex that connects b_i to the rest of the graph. So, every path from any vertex to b_i passes necessarily by b_{i-1}. Then, $P(u_i, b_i) = (P(u_{i-1}, b_{i-1}), b_i)$. If the up neighborhood of b_i is not empty, then there is an edge between u_i and b_i, and $P(u_i, b_i) = (u_i, b_i)$. Consider now that the up neighborhood of b_i is empty but its down neighborhood is not. So, again, $u_i = u_{i-1}$, and any path from u_i to b_i should pass through b_{i-1} or w_{i-1} (or both).

If $b_i w_{i-1} \in E(G_i)$, then any induced path from u_i to b_i has to pass through exactly one of b_{i-1} and w_{i-1}. If it passes through w_{i-1}, then it is at most as long as the induced path $(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), b_i)$; otherwise it is at most as long as $(P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i)$. So the maximum one of these two paths is a longest induced path from u_i to b_i in G_i.

Otherwise, consider $w_{i-1}b_i \notin E(G_i)$. So $g_i \notin V(G_{i-1})$. (Recall that g_i is the biggest neighbor of b_i in O between v_{q+1} and v_l when $downN(b_i) \neq \emptyset$.) Then any induced path from u_i to b_i has to pass through exactly one of b_{i-1} and g_i. If it passes through g_i, then it is at most as long as the induced path $(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), g_i, b_i)$, since any induced path from u_i to g_i has to pass through w_{i-1}; otherwise it is at most as long as $(P(u_{i-1}, b_{i-1}), b_i)$. So the maximum one of these two paths is a longest induced path from u_i to b_i in G_i. □

Algorithm 2 computes a longest induced path between u_i and b_i in G_i not passing through w_i. A longest induced path between w_i and b_i in G_i not passing through u_i is calculated similarly.

Algorithm 2: Longest induced path $LIP(u_i, b_i, \bar{w}_i)$

Input: G_i and a longest induced path $P(u_{i-1}, b_{i-1})$ between u_{i-1} and b_{i-1} in the subgraph G_{i-1}, also a longest induced path $P(u_{i-1}, w_{i-1}, \bar{b}_{i-1})$ (resp. $P(u_{i-1}, b_{i-1}, \bar{w}_{i-1})$) between u_{i-1} and w_{i-1} (resp. b_{i-1}) not passing through b_{i-1} (resp. w_{i-1}) in the subgraph G_{i-1}

Output: $P(u_i, b_i, \bar{w}_i)$

if $upN(b_i) = \emptyset$ *and* $downN(b_i) = \emptyset$ **then**
  $P(u_i, b_i, \bar{w}_i) \leftarrow (P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i)$
else if $upN(b_i) \neq \emptyset$ **then**
  $P(u_i, b_i, \bar{w}_i) \leftarrow (u_i, b_i)$
else if $|downN(b_i)| = 1$ *and* $b_i w_{i-1} \notin E(G_i)$ **then**
  $P(u_i, b_i, \bar{w}_i) \leftarrow (P(u_{i-1}, b_{i-1}), b_i)$
else if $|downN(b_i)| = 1$ *and* $b_i w_{i-1} \in E(G_i)$ **then**
  $P(u_i, b_i, \bar{w}_i) \leftarrow (P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i)$
else if $|downN(b_i)| > 1$ *and* $b_i w_{i-1} \notin E(G_i)$ **then**
  $P(u_i, b_i, \bar{w}_i) \leftarrow max\{(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), g_i, b_i), (P(u_{i-1}, b_{i-1}), b_i)\}$
else
  $P(u_i, b_i, \bar{w}_i) \leftarrow max\{(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), b_i), (P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i)\}$

Lemma 27. *Algorithm 2 outputs a longest induced path $P(u_i, b_i, \bar{w}_i)$ between u_i and b_i not passing through w_i in G_i.*

Proof. Recall that G_i is the graphs induced by $V(G_{i-1})$, b_i and its neighbors in O. If the up neighborhood and the down neighborhood of b_i are empty, then $u_i = u_{i-1}$, $w_i = w_{i-1}$ and b_{i-1} is the only vertex that connects b_i to the rest of the graph. So, every path from any vertex to b_i passes necessarily by b_{i-1}. Then, $P(u_i, b_i, \bar{w}_i) = (P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i)$. If the up neighborhood of b_i is not empty, then there is an edge between u_i and b_i, and $P(u_i, b_i, \bar{w}_i) = (u_i, b_i)$. Consider now that the up neighborhood of b_i is empty but its down neighborhood is not. So $u_{i-1} = u_i$. If b_i has only one neighbor in its down neighborhood and $b_i w_{i-1} \notin E(G_i)$, then $w_i \neq w_{i-1}$ and $w_i w_{i-1} \in E(G_i)$. Since $u_i = u_{i-1}$ and w_i

is not in G_{i-1}, $(P(u_{i-1},b_{i-1}),b_i)$ is a longest induced path between u_i and b_i not passing by w_i. If b_i has only one neighbor in its down neighborhood and $b_iw_{i-1} \in E(G_i)$, then $w_i = w_{i-1}$ and the desired path is $(P(u_{i-1},b_{i-1},\bar{w}_{i-1}),b_i)$. If b_i has more than one neighbor in its down neighborhood and $b_iw_{i-1} \notin E(G_i)$, then $P(u_i,b_i,\bar{w}_i)$ is the longer path between $(P(u_{i-1},w_{i-1},\bar{b}_{i-1}),g_i,b_i)$ and $(P(u_{i-1},b_{i-1}),b_i)$, since any path from u_i to b_i should pass through b_{i-1} or w_{i-1}. Otherwise, $P(u_i,b_i,\bar{w}_i)$ is the longer one between $(P(u_{i-1},w_{i-1},\bar{b}_{i-1}),b_i)$ and $(P(u_{i-1},b_{i-1},\bar{w}_{i-1}),b_i)$. □

Algorithm 3 computes a longest induced path between u_i and w_i in G_i which does not pass through b_i.

Algorithm 3: Longest induced path $LIP(u_i,w_i,\bar{b}_i)$

Input: G_i and a longest induced path $P(u_{i-1},w_{i-1})$ between u_{i-1} and w_{i-1} in the subgraph G_{i-1}

Output: $P(u_i,w_i,\bar{b}_i)$

1 $P(u_i,w_i,\bar{b}_i) \leftarrow (O[u_i,\cdots,u_{i-1}],P(u_{i-1},w_{i-1}),O[w_{i-1},\cdots,w_i])$

Lemma 28. *Algorithm 3 outputs a longest induced path $P(u_i,w_i,\bar{b}_i)$ between u_i and w_i not passing through b_i in the subgraph graph G_i.*

Proof. Any induce path between u_i and w_i not passing through b_i has to pass through u_{i-1} and w_{i-1}. The only induced path from u_i (resp. w_i) to u_{i-1} (resp. w_{i-1}) is $O[u_i,\ldots,u_{i-1}]$ (resp. $O[w_i,\ldots,w_{i-1}]$), the segment on O. So $P(u_i,w_i,\bar{b}_i) = (O[u_i,\cdots,u_{i-1}],P(u_{i-1},w_{i-1}),O[w_{i-1},\cdots,w_i])$. □

Algorithm 4 calculates a longest induced $P(u_i,w_i)$ path between u_i and w_i in G_i. Note that $P(u_i,w_i)$ is the longer path between a longest induced path from u_i to w_i passing through b_i and one not passing through b_i, i.e., $P(u_i,w_i,\bar{b}_i)$. It is simple to compute $P(u_i,w_i,\bar{b}_i)$ in Algorithm 3. However, many cases need to be considered to compute a longest induced path between u_i and w_i passing through b_i.

Lemma 29. *Algorithm 4 outputs a longest induced path $P(u_i,w_i)$ between u_i and w_i in the subgraph graph G_i.*

Proof. A longest induced $P(u_i,w_i)$ path between u_i and w_i in G_i is the longer path of two paths, a longest induced path between u_i and w_i not passing through b_i, i.e., $P(u_i,w_i,\bar{b}_i)$, and a longest induced path between u_i and w_i passing through b_i. As proved in Lemma 28, $P(u_i,w_i,\bar{b}_i)$ can be computed by Algorithm 3. The main difficulty is to compute a longest induced path between u_i and w_i passing through b_i. In the following, the proof is performed case by case.

- If b_i has no neighbors in O, i.e. $upN(b_i) = \emptyset$ and $downN(b_i) = \emptyset$, then b_i is only adjacent to b_{i-1} in G_i and $u_i = u_{i-1}$, $w_i = w_{i-1}$. No path in G_i between u_i and w_i passes through b_i. So $P(u_i,w_i) = P(u_i,w_i,\bar{b}_i)$.

Algorithm 4: Longest induced path $LIP(u_i, w_i)$

Input: G_i and a longest induced path $P(u_{i-1}, w_{i-1})$ between u_{i-1} and w_{i-1} in the subgraph G_{i-1}, also longest induced path $P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))$ and $P(b_{i-1}, w_{i-1})$ in the subgraph G_{i-1}

Output: $P(u_i, w_i)$

$P(u_i, w_i, \bar{b}_i) \leftarrow$ The output of Algorithm 3 Longest induced path $LIP(u_i, w_i, \bar{b}_i)$
if $upN(b_i) = \emptyset$ *and* $downN(b_i) = \emptyset$ **then**
    $P(u_i, w_i) \leftarrow P(u_i, w_i, \bar{b}_i)$
else if $upN(b_i) \neq \emptyset$ *and* $downN(b_i) \neq \emptyset$ **then**
    $P(u_i, w_i) \leftarrow \max\{P(u_i, w_i, \bar{b}_i), u_i, b_i, w_i\}$
else
    Exactly one of $upN(b_i)$ and $downN(b_i)$ is empty. /*We deal with the cases in which $upN(b_i) \neq \emptyset$ and $downN(b_i) = \emptyset$ in the following (the case in which $upN(b_i) = \emptyset$ and $downN(b_i) \neq \emptyset$ is analogue)*/
    if $|upN(b_i)| \geq 2$ *and* $b_i u_{i-1} \notin E(G_i)$ **then**
        $P(u_i, w_i) \leftarrow max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}))\}$
    else if $|upN(b_i)| \geq 2$ *and* $b_i u_{i-1} \in E(G_i)$ **then**
        $P(u_i, w_i) \leftarrow max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))\}$
    else if $|upN(b_i)| = 1$ *and* $b_i u_{i-1} \notin E(G_i)$ **then**
        $P(u_i, w_i) \leftarrow max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))\}$
    else
        $h \leftarrow \max\{x : b_x u_{i-1} \in E(G_{i-1}), 1 \leq x \leq i-1\}$
        $m \leftarrow \max\{y : b_y u_{i-1} \in E(G_{i-1}), 1 \leq y \leq i-1\}$
        if $h < m$ **then**
            $P(u_i, w_i) \leftarrow max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, \ldots, b_m, w_{i-1})\}$
        else
            $P(u_i, w_i) \leftarrow P(u_i, w_i, \bar{b}_i)$

- If b_i has two segments of neighbors in O, i.e., $upN(b_i) \neq \emptyset$ and $downN(b_i) \neq \emptyset$, then $u_i b_i, w_i b_i \in E(G_i)$. So if $P(u_i, w_i)$ passes through b_i, then $P(u_i, w_i) = (u_i, b_i, w_i)$. So $P(u_i, w_i) = \max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, w_i)\}$.
- Otherwise b_i has only one segment of neighbors in O. Suppose $upN(b_i) \neq \emptyset$ and $downN(b_i) = \emptyset$. The other case can be proved symmetrically. So $w_i = w_{i-1}$, $b_i w_i \notin E(G_i)$ and $b_i u_i \in E(G_i)$.
  - If $|upN(b_i)| \geq 2$ and $b_i u_{i-1} \notin E(G_i)$, then $u_i \neq u_{i-1}$ and $u_i u_{i-1} \notin E(G_i)$. A longest induced path between u_i and w_i passing through b_i also passes through b_{i-1}. (If not, it needs to passes through u_{i-1}. But an induced path from u_i to u_{i-1} passing through b_i is not longer than the one $O[u_i, \cdots, u_{i-1}]$.) Then it consists of two subpaths, (u_i, b_i, b_{i-1}) and a

longest induced path $P(b_{i-1}, w_{i-1})$ between b_{i-1} and $w_i = w_{i-1}$ in G_{i-1}. So $P(u_i, w_i) = \max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}))\}$. Note that b_i is only adjacent to b_{i-1} in G_{i-1} and u_i is not adjacent to any vertex in G_{i-1}. So $(u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}))$ is an induced path. But it will be different for the next two cases below (b_i or u_i will be adjacent to u_{i-1}).

- Consider the case in which $|upN(b_i)| \geq 2$ and $b_i u_{i-1} \in E(G_i)$. As mentioned above, the path $(u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}))$ may be not induced if $P(b_{i-1}, w_{i-1})$ passes through u_{i-1} because $b_i u_{i-1} \in E(G_i)$. So we replace $P(b_{i-1}, w_{i-1})$ with $P(b_{i-1}, w_{i-1}, \bar{u}_{i-1})$. Then $(u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))$ is a longest induced path between u_i and w_i passing through b_i. So $P(u_i, w_i) = \max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))\}$.
- If $|upN(b_i)| = 1$ and $b_i u_{i-1} \notin E(G_i)$, then $u_i \neq u_{i-1}$ and $u_i u_{i-1} \in E(G_i)$. Similarly, the path $(u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}))$ may be not induced if $P(b_{i-1}, w_{i-1})$ passes through u_{i-1} because $u_i u_{i-1} \in E(G_i)$. So we replace $P(b_{i-1}, w_{i-1})$ with $P(b_{i-1}, w_{i-1}, \bar{u}_{i-1})$. Then the path $(u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))$ is a longest induced path between u_i and w_i passing through b_i. So $P(u_i, w_i) = \max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}, \bar{u}_{i-1}))\}$.
- Now see the last case in which $|upN(b_i)| = 1$ and $b_i u_{i-1} \in E(G_i)$, i.e. b_i has only one neighbor $u_i = u_{i-1}$ in O. Any path between u_i and w_i passing through b_i also passes through b_{i-1}, because b_i is only adjacent to u_i and b_{i-1} in this case. Then it consists of two subpaths, (u_i, b_i, b_{i-1}) and a longest induced path $P(b_{i-1}, w_{i-1})$ between b_{i-1} and $w_i = w_{i-1}$ in G_{i-1} not passing through u_{i-1} and its neighbors (to make sure $(u_i, b_i, b_{i-1}, P(b_{i-1}, w_{i-1}))$ is induced). Since $u_i = u_{i-1}$ and $w_i = w_{i-1}$ are both in G_{i-1}, they are neighbors of some vertices in $\{b_1, \ldots, b_{i-1}\}$. As defined in the algorithm, $b_h \in \{b_1, \ldots, b_{i-1}\}$ is the one with biggest index h such that b_h is a neighbor of u_{i-1}, i.e., no vertex b_x is adjacent to u_{i-1} for $x = h+1, \ldots, i-1$. Fact 1 shows that $upN(b_x) = \emptyset$ for $x = h+1, \ldots, i-1$. Similarly $b_m \in \{b_1, \ldots, b_{i-1}\}$ is the one with biggest index m such that b_m is a neighbor of w_{i-1}. Then from Fact 1 each vertex $b_y \in \{b_{m+1}, \ldots, b_{i-1}\}$ satisfies $downN(b_y) = \emptyset$ for $y = m+1, \ldots, i-1$. So no induced path from b_{i-1} to w_{i-1} can reach O until arriving b_h or b_m. If $h < m$, then the path passes b_m first and goes to w_{i-1} directly. So $P(u_i, w_i) = \max\{P(u_i, w_i, \bar{b}_i), (u_i, b_i, b_{i-1}, \ldots, b_m, w_{i-1})\}$. Otherwise, the path passes b_h and $u_{i-1} b_h \in E(G_i)$ then $u_i b_i b_{i-1} \cdots b_h$ is a cycle. So $P(u_i, w_i) = P(u_i, w_i, \bar{b}_i)$.

□

Finally, we describe the main algorithm. It receives G with O an induced cycle and returns, after successive calls of the previous described subroutines, a longest induced cycle on it.

Theorem 13. *Given a planar k-super-caterpillar G with a backbone $B = \{b_1, \ldots, b_p\}$ and $O = G \setminus B$ being an induced cycle, Algorithm 5 gives the longest induce cycle C in linear time.*

Algorithm 5: Main: longest induced cycle

Input: G is a planar k-super-caterpillar G with a backbone $B = \{b_1, \ldots, b_p\}$ and $O = G \setminus B$ being an induced cycle $\{v_1, \ldots, v_l\}$

Output: C

if $downN(b_1) = \emptyset$ **then**
  $P(u_1, b_1) \leftarrow (u_1, b_1)$; $P(u_1, b_1, \bar{w}_1) \leftarrow (u_1, b_1)$; $P(w_1, b_1) \leftarrow \emptyset$;
  $P(w_1, b_1, \bar{u}_1) \leftarrow \emptyset$; $P(u_1, w_1) \leftarrow \emptyset$; $P(u_1, w_1, \bar{b}_1) \leftarrow \emptyset$;
  if $upN(b_1) = \{v_1\}$ **then**
    $C_1 \leftarrow \emptyset$
  else
    $C_1 \leftarrow (v_1, b_1, v_2, v_1)$
else
  $P(u_1, b_1) \leftarrow (u_1, b_1)$; $P(u_1, b_1, \bar{w}_1) \leftarrow (u_1, b_1)1$; $P(w_1, b_1) \leftarrow (w_1, b_1)$;
  $P(w_1, b_1, \bar{u}_1) \leftarrow (w_1, b_1)$; $P(u_1, w_1) \leftarrow O[w_1, \cdots, u_1]$;
  $P(u_1, w_1, \bar{b}_1) \leftarrow O[w_1, \cdots, u_1]$; $C_1 \leftarrow (v_1, g_1, b_1, v_1)$
for $i \leftarrow 2$ **to** p **do**
  if $upN(b_i) = \emptyset$ *and* $downN(b_i) = \emptyset$ **then**
    $C_i \leftarrow C_{i-1}$
  else if $upN(b_i) \neq \emptyset$ *and* $downN(b_i) = \emptyset$ *(the case in which* $upN(b_i) = \emptyset$ *and* $downN(b_i) \neq \emptyset$ *is analogue)* **then**
    $C_i \leftarrow max\{C_{i-1}, (P(u_{i-1}, b_{i-1}), b_i, f_i, u_{i-1})\}$
  else if $b_i u_{i-1} \in E(G_i)$ *and* $b_i w_{i-1} \in E(G_i)$ **then**
    $C_i \leftarrow max\{C_{i-1}, (P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), b_i, u_{i-1}),$
      $(P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i, u_{i-1}), (P(w_{i-1}, b_{i-1}, \bar{u}_{i-1}), b_i, w_{i-1})\}$;
  else if $b_i u_{i-1} \in E(G_i)$ *and* $b_i w_{i-1} \notin E(G_i)$ *(the case in which* $b_i u_{i-1} \notin E(G_i)$ *and* $b_i w_{i-1} \in E(G_i)$ *is analogue)* **then**
    $C_i \leftarrow max\{C_{i-1}, (P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), g_i, b_i, u_{i-1}),$
      $(P(u_{i-1}, b_{i-1}), b_i, u_{i-1}), (P(w_{i-1}, b_{i-1}, \bar{u}_{i-1}), b_i, g_i, w_{i-1})\}$;
  else
    $C_i \leftarrow max\{C_{i-1}, (P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}) g_i, b_i, f_i, u_{i-1}),$
      $(P(u_{i-1}, b_{i-1}), b_i, f_i, u_{i-1}), (P(w_{i-1}, b_{i-1}), b_i, g_i, w_{i-1})\}$;
  $P(u_i, b_i) \leftarrow$ The output of Algorithm 1 Longest induced path $LIP(u_i, b_i)$
  $P(u_i, b_i, \bar{w}_i) \leftarrow$ The output of Algorithm 2 Longest induced path $LIP(u_i, b_i, \bar{w}_i)$
  $P(w_i, b_i) \leftarrow$ Longest induced path $LIP(w_i, b_i)$ computed as in Algorithm 1
  $P(w_i, b_i, \bar{u}_i) \leftarrow$ Longest induced path $LIP(w_i, b_i, \bar{u}_i)$ computed as in Algorithm 2
  $P(u_i, w_i, \bar{b}_i) \leftarrow$ The output of Algorithm 3 Longest induced path $LIP(u_i, w_i, \bar{b}_i)$
  $P(u_i, w_i) \leftarrow$ The output of Algorithm 4 Longest induced path $LIP(u_i, w_i)$
return C_p

Proof. Let C_i be the cycle computed by the algorithm for $1 \leq i \leq p$. It is enough to prove that C_i is a longest induced cycle in G_i, which is the subgraph of G induced by $\{b_1, \ldots, b_i\}$ and all their neighbors in O. The proof is by induction. For $i = 1$, G_1 is induced by b_1 and its neighbors in O. As we assumed before, v_1 is a neighbor of b_1 (but not a neighbor of b_2). So $upN(b_1) = \{v_1, v_2, \ldots, u_1\}$. $downN(b_1)$ may be an empty set. Because of the simple structure of G_1, one can see that for both cases, $downN(b_1)$ is empty or not, C_1 is a longest induced cycle, a triangle, in G_1 if it exists. Simultaneously, $P(u_1, b_1)$ is a longest induced path from u_1 to b_1 and $P(u_1, b_1, \bar{w}_1)$ is the one not passing through w_1; similarly $P(w_1, b_1)$, $P(w_1, b_1, \bar{u}_1)$, $P(u_1, w_1)$, $P(u_1, w_1, \bar{b}_1)$ are the longest induced paths if they exist. Suppose C_{i-1} is a longest induced cycle in G_{i-1}. In the following, we prove that C_i is a longest induced cycle in G_i.

Note that $S = \{u_{i-1}, b_{i-1}, w_{i-1}\}$ is a separator in G_i. In $G_i \setminus S$, there are two connected components, $L = G_{i-1} \setminus S$ and $R = G_i \setminus G_{i-1}$. Any induced cycle in G_i is contained either in $L \cup G_i[S] = G_{i-1}$ or in $G_i[S] \cup R$ or none of them. A longest induced cycle in G_i is the longest one among these three case, denote them as C^{left}, C^{right} and C^{mid} respectively. We see that C^{left} is a longest induced cycle in G_{i-1}, so $C^{left} = C_{i-1}$ by assumption. The subgraph $H_i = G_i[S] \cup R$ is induced by S, b_i and its neighbors in G_i. Because of the simple structure of H_i, C^{right} is easy to get if it exists, either a triangle or a square. We are going to see that C^{right} is always smaller than C^{mid}. So $C_i = \max\{C_{i-1}, C^{mid}\}$. Since S is a separator between L and R and $|S| = 3$, $C^{mid} \cap R$ is an induced path in G_i and $|V(C^{mid} \cap R)| \geq 1$. The end(s) of this induced path is (are) adjacent to two vertices in S. There are three possibilities for these two vertices in S, $\{u_{i-1}, w_{i-1}\}$, $\{u_{i-1}, b_{i-1}\}$ or $\{w_{i-1}, b_{i-1}\}$. Then C^{mid} consists of two induced paths between these two vertices, one in G_{i-1} and the other one in H_i with all interior vertices in R. So $C^{mid} \cap G_{i-1}$ is also an induced path between these two vertices. (Note that this induced path may also pass through the third vertex in S, i.e. $|S \cap C^{mid}|$ may be 3.) Consider the for loop in the algorithm. In the $(i-1)$th loop, we get that $P(u_{i-1}, b_{i-1})$, $P(u_{i-1}, b_{i-1}, \bar{w}_{i-1})$, $P(w_{i-1}, b_{i-1})$, $P(w_{i-1}, b_{i-1}, \bar{u}_{i-1})$, $P(u_{i-1}, w_{i-1})$, $P(u_{i-1}, w_{i-1}, \bar{b}_{i-1})$ are the longest induced paths in G_{i-1} between two vertices and some of them not passing through specific vertices from Lemmas 26 - 29. To compute the desired induced paths in H_i and to make sure that C^{mid} is induced, several cases need to be considered.

- If b_i has no neighbors, i.e., $upN(b_i) = \emptyset$ and $downN(b_i) = \emptyset$, then there is no desired induced path in H_i. Then C^{right} and C^{mid} do not exist. So $C_i = C_{i-1}$.

- Suppose b_i has only one segment neighbors, i.e., $upN(b_i) \neq \emptyset$ and $downN(b_i) = \emptyset$ or the contrary. We consider only the first case and the other one can be proved similarly. There is only one induced path $(b_{i-1}, b_i, f_i, u_{i-1})$ from b_{i-1} to u_{i-1}. Remind that f_i denotes the smallest neighbor of b_i if $upN(b_i) \neq \emptyset$. Then $f_i = u_{i-1}$ if $u_{i-1}b_i \in E(G)$. So $C^{mid} = (P(u_{i-1}, b_{i-1}), b_i, f_i, u_{i-1})$ and it is induced because b_i and f_i is not adjacent to any interior vertices in $P(u_{i-1}, b_{i-1})$. Then $C_i = \max\{C_{i-1}, (P(u_{i-1}, b_{i-1}), b_i, f_i, u_{i-1})\}$.

- Now consider b_i has two segments neighbors, i.e., $upN(b_i) \neq \emptyset$ and $downN(b_i) \neq \emptyset$. We see that all the induced paths in H_i between two vertices in S passes through b_i. So to make sure C^{mid} is induced, it is necessary to consider the cases b_i is adjacent to u_{i-1}

and w_{i-1} or not separatorly. If $b_iu_{i-1} \in E(G_i)$ and $b_iw_{i-1} \in E(G_i)$, then the induced path from w_{i-1} to u_{i-1} is (w_{i-1}, b_i, u_{i-1}). Combining it with $P(u_{i-1}, w_{i-1}, \bar{b}_{i-1})$ or $P(u_{i-1}, w_{i-1})$ will obtain a cycle in G_i. To make sure the obtained cycle is induced, we get $(P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), b_i, u_{i-1})$ since $b_ib_{i-1} \in E(G_i)$. Similarly, the induced path in H_i from b_{i-1} to u_{i-1} (resp. w_{i-1}) is (b_{i-1}, b_i, u_{i-1}) (resp. (b_{i-1}, b_i, w_{i-1})). Combining it with $P(u_{i-1}, b_{i-1}, \bar{w}_{i-1})$ (resp. $P(w_{i-1}, b_{i-1}, \bar{u}_{i-1})$) gives an induced cycle, $(P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i, u_{i-1})$ (resp. $(P(w_{i-1}, b_{i-1}, \bar{u}_{i-1}), b_i, w_{i-1})$). Then C^{mid} is the longest cycle among these three. We see that C^{right} is a triangle and it is smaller than C^{mid}. So we have that $C_i = \max\{C_{i-1}, (P(u_{i-1}, w_{i-1}, \bar{b}_{i-1}), b_i, u_{i-1}), (P(u_{i-1}, b_{i-1}, \bar{w}_{i-1}), b_i, u_{i-1}), (P(w_{i-1}, b_{i-1}, \bar{u}_{i-1}), b_i, w_{i-1})\}$. The other three cases can be proved similarly, in which $b_iu_{i-1} \in E(G_i)$ and $b_iw_{i-1} \notin E(G_i)$, $b_iu_{i-1} \notin E(G_i)$ and $b_iw_{i-1} \in E(G_i)$, $b_iu_{i-1} \notin E(G_i)$ and $b_iw_{i-1} \notin E(G_i)$. We omit the tedious proofs.

In the for loop, the induce paths and cycle are updated in constant time. So the algorithm runs in linear time. □

3.5 Perspectives

Inspired by the study of cops and robber games on k-chordal graphs, we get a polynomial algorithm that, given a graph G and $k \geq 3$, either returns an induced cycle larger than k in G, or computes a k-good tree decomposition of G. So for any k-chordal graph, we always get a k-good tree decomposition of this graph. A graph with a k-good tree decomposition is proved to have bounded ($O(k)$) treelength and hyperbolicity; also its treewidth is bounded by $O(k-1)(\Delta-1)+2$, where Δ is the maximum degree of the graph. It is interesting to explore some algorithmic applications of k-good tree decompositions. In next chapter, we apply it to design a compact routing schemes for graphs admitting a k-good tree decompositions, including k-chordal graphs.

In the k-good tree decomposition for $k = 2$, each bag contains a dominating vertex, which is adjacent to all other vertices in the bag. So we also call the 2-good tree decompositions as *star tree decompositions*. Note that in the proof of Theorem 8, it is a necessary condition that $k \geq 3$. One of the ongoing works is to characterize graphs admitting a star tree decomposition.

Recall that given a tree decomposition $(T, \mathcal{X})$ of a graph $G = (V, E)$, the *breadth* of $(T, \mathcal{X})$ is the minimum integer r such that for every bag $X \in \mathcal{X}$ there is a vertex $v_X \in V$ at distance at most r from all other vertices in X. Note that in the definition of breadth, v_X is not necessary contained in X. But we can prove that a graph has a star tree decomposition if and only if the graph has *treebreadth* 1, which is the minimum breadth over all tree decompositions of the graph. In [AAD14], Abu-Ata and Dragan said that ' it is conceivable that the problem of determining whether a given graph has treebreadth at most $\rho \geq 1$ is NP-complete'. But no proof is given. The complexity is still open to decide whether a given graph has a star tree decomposition. We have not gotten a strong clue for the NP-completeness.

Another challenge is to prove or disprove the Conjecture 1, which says that it is NP-complete to decide whether a graph with a Hamilton path has a Hamilton cycle.

Part II

Routing Problems

CHAPTER 4

Compact Routing Scheme for k-Good Tree Decomposable Graphs

Contents

Compact routing problem, is one of the research areas in which tree decompositions have already been proved useful [Dou05]. As an application of k-good tree decomposition presented in Chpater 3, this chapter proposes a *compact routing scheme* for the class of graphs admitting such tree-decompositions, the so called *k-good tree decomposable graphs.* The results of this chapter is a collaboration with N. Nisse, K. Suchan and A. Kosowski. It has been published in the proceeding of the conference ICALP 2012 [c-KLNS12] and in the journal Algorithmica [j-KLNS14].

4.1 Introduction

Transmitting messages between pairs of routers (vertices) is an elementary and essential activity of any communication networks. This mission is performed by using a *routing scheme*, which is an algorithm directing traffic in a network. More precisely, when any source vertex in the network has a message to a destination vertex, a *routing scheme* is an algorithm selecting the best path to send the message to its destination according to its *routing tables*, which stores some informations of the network. Naturally, it is fastest to route messages along shortest paths from the sources to the destinations. To guarantee optimal routes, a straightforward approach is to store a *complete routing table* in each vertex s of the network, which for each destination vertex t, specifies the first edge on some shortest path from s to t. This approach requires $O(n \log d)$ memory bits for a vertex of degree d in an n-vertex network. So it becomes too expensive for large scale communication networks, e.g. the Internet, the Facebook. Thus it is interesting to design *compact routing schemes*,

which requires lower memory and produces paths not too much longer than the shortest paths.

The efficiency of a routing scheme is measured in terms of its *multiplicative* or *additive stretch* factors, which are the maximum ratio or difference between the length of a path computed by the scheme and the one of a shortest path between the same pair of vertices. The space complexity of a routing scheme is mainly (not only) measured in terms of the size of its *routing table* in each vertex, which stores the necessary information for routing messages. The *address* or (*name*) of each vertex and the port number of each edge also requires some memory space. If the designer of the scheme is allowed to label the vertices in the way he wants, then the routing scheme is called *labelled* routing scheme; otherwise, that is, each vertex in the network already has a predefined fixed label or name, it is called *name-independent* routing scheme. Let us see some related works below.

4.1.1 Related Work

In a *name-independent* routing scheme, the designer of the scheme is not allowed to label the vertices in the way he wants, that is, each vertex in the network has a predefined fixed label. Abraham et al. provided a universal name-independent routing scheme with multiplicative stretch $O(k)$ and $O(n^{1/k}polylog(n))$ space in [AGM04]. There is a big constant hidden in the multiplicative stretch $O(k)$ of this routing scheme. But for $k = 2$, Abraham et al. reduces the hidden constant. They presented a compact routing scheme using $O(\sqrt{n}polylog(n))$ space and multiplicative stretch 3 [AGM$^+$08]. This is optimal in the sense that there are weighted trees for which every name-independent routing scheme with space less than $n^{1/k}$ requires stretch at least $2k+1$ and average stretch at least $k/4$ [AGM06].

In *labelled* routing scheme, a routing scheme with multiplicative stretch at most $2k-1$, $k \geq 2$ and using $n^{1/k}polylog(n)$ bits per vertex in arbitrary graph is designed in [TZ05]. Moreover, for any shortest path routing scheme, for any constant ε, $0 < \varepsilon < 1$, and for every integer d such that $3 \leq d \leq \varepsilon n$, there exists a n-vertex network of maximum degree d that locally requires $\Theta(n \log d)$ bits of memory on $\Theta(n)$ vertices [GP96], which is the same as the complete routing table. Subsequently, the interest of the scientific community was turned toward specific properties of graphs. Several routing schemes have been proposed for particular graph classes: e.g., trees [FG01], bounded *growth*[1] [AM05], bounded hyperbolic graph [CDE$^+$12], bounded *doubling dimension*[2] [AGGM06, KRX06], excluding a fixed graph as a minor [AGM05, AG06], etc. The best compact routing scheme in k-chordal graphs (independent from the maximum degree) is due to Dourisboure and achieves a stretch of $k+1$ using routing tables of size $O(\log^2 n)$ bits [Dou05]. A routing scheme achieving stretch $k-1$ with a distributed algorithm for computing routing tables of size $O(\Delta \log n)$ bits has been proposed in [NRS12], where Δ denotes the maximum degree of the graph.

[1] A weighted undirected graph is δ *growth bounded* if the number of vertices at distance $2r$ around any given vertex is at most δ times the number of vertices at distance r [AM05].

[2] The *doubling dimension* of a graph G is the minimum k such that any ball of radius R can be covered by 2^k balls of radius $R/2$ [GKL03].

The rest of this chapter is organized as follows. We describe the model and data structures applied in our compact routing scheme in the next section. Then the details of the compact routing algorithm is presented for k-good tree decomposable graphs in Section 4.3. It is proved that our routing scheme has additive stretch at most $O(k \log \Delta)$ in Section 4.4. Note that in the following the degree of any $v \in V(G)$ is denoted as $d_G(v)$.

4.2 Model And Data Structures

We propose a *labelled* routing scheme in which we are allowed to give one identifier, $name(v)$, of $O(\log n)$ bits to any vertex v of G. Moreover, following [FG01], we consider the *designer-port* model, which allows us to choose the permutation of ports (assign a label of $\log d_G(v)$ bits to any edge incident to v in $V(G)$). Finally, to any vertex $v \in V(G)$, we assign a routing table, denoted by $Table(v)$, where local information of $O(k \cdot \log \Delta + \log n)$ bits is stored. Any message has a *header* that contains the address $name(t)$ of the destination t, three modifiable integers $pos \in \{-1, 1, 2, \cdots, k-1\}, cnt, cnt' \in \{-1, 0, \cdots, \Delta+1\}$, one bit $start$ and some memory, called $path$, of size $O(k \cdot \log \Delta)$ bits. The two items $start$ and $path$ change only once.

Following our routing scheme, a vertex v that receives a message uses its header, $name(v)$, $Table(v)$ and the port-numbers of the edges incident to v to compute its new header and to choose the edge $e = \{v, u\}$ over which it relays the message. Then, the vertex u knows that the message arrived from v. The length of the path followed by a message from a source $s \in V(G)$ to a destination $t \in V(G)$, using the routing scheme, is denoted by $|P(s,t)|$, and so the additive stretch of the scheme is $\max_{s,t \in V(G)}(|P(s,t)| - d(s,t))$ where $d(s,t)$ is the distance between s and t in G.

To design our routing scheme, we combine the compact routing scheme in trees of [FG01] together with the k-good tree-decomposition. Roughly, the scheme consists of following the paths in a BFS-tree F of G according to the scheme in [FG01], and using one bag of the tree-decomposition as a short-cut between two branches of F. Intuitively, if the source s and the destination t are "far apart", then there is a bag X of the tree-decomposition that separates s and t in G. The message follows the path in F to the root of F until it reaches X, then an exhaustive search is done in X until the message finds an ancestor y of t, and finally it follows the path from y to t in F using the scheme of [FG01]. The remaining part of this chapter is devoted to the proof of the next Theorem that summarizes the performances of our routing scheme.

Theorem 14. *For any n-vertex m-edge graph G with maximum degree Δ and with a k-good tree-decomposition, there is a labelled routing scheme $\mathcal{R}$ with the following properties. $\mathcal{R}$ uses addresses of size $O(\log n)$ bits, port-numbers of size $O(\log \Delta)$ bits and routing tables of size $O(k \cdot \log \Delta + \log n)$ bits. The routing tables, addresses and port-numbers can be computed in time $O(m^2)$. Except the address of the destination (not modifiable), the header of a message contains $O(k \cdot \log \Delta)$ modifiable bits. The header and next hop are computed in time $O(1)$ at each step of the routing. Finally, the additive stretch is $\leq 2k(\lceil \log \Delta \rceil + \frac{5}{2}) - 2\lceil \log \Delta \rceil - 4$.*

4.2.1 Routing In Trees [FG01]

Since we use the shortest path routing scheme proposed in [FG01] for trees, we start by recalling some of the data structures they use. Let F be a tree rooted in $r \in V(F)$. For any $v \in V(F)$, let F_v be the subtree of F rooted in v and let $w_F(v) = |V(F_v)|$ be the *weight* of v. Consider a Depth-First-Search (DFS) traversal of F, starting from r, and guided by the weight of the vertices, i.e., at each vertex, the DFS visits first the largest subtree, then the second largest subtree, and so on. For any $v \in V(F)$, let $Id_F(v) \in \{1, \cdots, n\}$ be the preordering rank of v in the DFS.

Lemma 30. *For any $u, v \in V(F)$, $v \in V(F_u)$ if and only if $Id_F(u) \leq Id_F(v) \leq Id_F(u) + w_F(u) - 1$.*

For any $v \in V(F)$ and any e incident to v, the edge e receives a *port-number* $p_F(e,v)$ at v as follows. Set $p_F(e,v) = 0$ if $v \neq r$ and e leads to the parent of v in F, i.e., the edge e is the first edge on the path from v to r. Otherwise, let $(u_1, \cdots, u_d)$ be the children of v (where $d = d_F(v)$ if $v = r$ and $d = d_F(v) - 1$ otherwise) ordered by their weight, i.e., such that $w_F(u_1) \geq \cdots \geq w_F(u_d)$. Then, let $p_F(\{u_i, v\}, v) = i$, for any $i \leq d$. Finally, each vertex $v \in V(F)$ is assigned a routing table $RT_F(v)$ and an address $\ell_F(v)$ of size $O(\log n)$ bits allowing a shortest path routing in trees (see details in [FG01]).

4.2.2 Data Structures

Let G be a graph with the k-good tree-decomposition $(T = (I, M), \{X_i | i \in I\})$. Let $r \in V(G)$. Let F be a Breadth-First-Search(BFS) tree of G rooted at r. Let T be rooted in $b \in I$ such that $r \in X_b$.

We use (some of) the data structures of [FG01] for both trees F and T. More precisely, for any $v \in V(G)$, let $Id_F(v), w_F(v), \ell_F(v)$ and $RT_F(v)$ be defined as above for the BFS-tree F. Moreover, we add $d_F(v)$ to store the degree of v in the tree F . Set $p_{e,v} = p_F(e,v)$ for edges that belong to F defined as above, the ports $> d_F(v)$ will be assigned to edges that do not belong to F. Knowing $d_F(v)$, the ports that correspond to edges in F can be easily distinguished from ports assigned to edges in $G \setminus E(F) \equiv \overline{F}$.

For any $v \in V(G)$, let $(u_1, \cdots, u_d) = N_{\overline{F}}(v)$ be the neighborhood of v in $\overline{F}$ ordered such that $Id_F(u_1) < \cdots < Id_F(u_d)$. We assign $p_{e_i,v} = d_F(v) + i$, where $e_i = \{v, u_i\}$, for each u_i in this order. This ordering will allow to decide whether one of the vertices in $N_{\overline{F}}(v)$ is an ancestor of a given vertex t in time $O(\log \Delta)$ by binary search.

For any $i \in I$, let $Id_T(i)$ and $w_T(i)$ be defined for the tree T as above. For any $v \in V(G)$, let $B_v \in I$ be the bag of T containing v which is closest to the root b of T. To simplify the notations, we set $Id_T(v) = Id_T(B_v)$ and $w_T(v) = w_T(B_v)$. These structures will be used to decide "where" we are in the tree-decomposition when the message reaches $v \in V(G)$.

Finally, for any $i \in I$, let $P_i = (v_1, \cdots, v_\ell)$ be the backbone of B_i with $\ell \leq k - 1$ (recall that we consider a k-good tree decomposition). Let $(e_1, \cdots, e_{\ell-1})$ be the set of edges of P_i in order. Set $Backbone_i = (p_{e_1,v_1}, p_{e_1,v_2}, p_{e_2,v_2}, \cdots, p_{e_{\ell-1},v_\ell})$. For any $v \in V(G)$ such that $Id_T(v) = i \in I$, if $v = v_j \in P_i$, then $back(v) = (\emptyset, j)$ and if $v \notin P_i$, let $back(v) = (p_{e_v}, j)$ where $e = \{v, v_j\}$ and v_j ($j \leq \ell$) is the neighbor of v in P_i with j minimum. This information will be used to cross a bag (using its backbone) of the tree-decomposition.

Now, for every $v \in V(G)$, we define the address $name(v) = \langle \ell_F(v), Id_T(v) \rangle$. Note that, in particular, $\ell_F(v)$ contains $Id_F(v)$. We also define the routing table of v as $Table(v) = \langle RT_F(v), d_F(v), w_T(v), Backbone(v), back(v) \rangle$, where $Backbone(v) = Backbone_i$ for $i = B_v$, i.e. the backbone of the bag containing v and closest to the root of T.

Next table summarizes all these data structures.

	notation	description
$name(v)$	$\ell_F(v)$	the address of v in tree F [FG01]
	$Id_T(v)$	the identifier of the highest bag B_v containing v in T
$Table(v)$	$RT_F(v)$	the routing table used of v for routing in F [FG01]
	$d_F(v)$	the degree of v in F
	$w_T(v)$	the weight of the subtree of T rooted in B_v
	$Backbone(v)$	information to navigate in the backbone of B_v
	$back(v)$	information to reach the backbone of B_v from v

Clearly, $name(v)$ has size $O(\log n)$ bits and $Table(v)$ has size $O(k \cdot \log \Delta + \log n)$ bits. Moreover, any edge e incident to v receives a port-number $p_{e,v}$ of size $O(\log \Delta)$ bits.

4.3 Compact Routing algorithm in k-good tree-decomposable graphs

Let us consider a message that must be sent to some destination $t \in V(G)$. Initially, the header of the message contains $name(t)$, the three counters pos, cnt, $cnt' = -1$, the bit $start = 0$ and the memory $path = \emptyset$, which stores the backbone of the bag containing an ancestor (in F) of the destination vertex of the message. Let $v \in V(G)$ be the current vertex where the message stands. First, using $Id_F(t)$ in $name(t)$, $Id_F(v)$ in $name(v)$ and $w_F(v)$ in $RT_F(v) \in Table(v)$, it is possible by using Lemma 30 to decide in constant time if v is an ancestor of t in F. Similarly, using $Id_T(t)$ in $name(t)$, $Id_T(v)$ in $name(v)$ and $w_T(v)$ in $Table(v)$, it is possible to decide if the highest bag B_v containing v is an ancestor of B_t in T. There are several cases to be considered.

- If v is an ancestor of t in F, then using the protocol of [FG01] the message is passed to the child w of v that is an ancestor of t in F towards t. Recursively, the message arrives at t following a shortest path in G, since F is a BFS-tree.
- Else, if $path = \emptyset$, then
  - if neither B_v is an ancestor of B_t in T nor $B_t = B_v$, then the message follows the edge leading to the parent of v in F, i.e., the edge with port-number $p_{e,v} = 0$. Note that the message will eventually reach a vertex w that either is an ancestor of t in F or B_w is an ancestor of B_t in T, since the message follows a shortest path to the root r of F and B_r is the ancestor of any bag in T.
  - Else, an ancestor of t belongs to B_v since either $B_v = B_t$, or B_v is an ancestor of B_t. (This is because T is a tree-decomposition, B_v has to contain a vertex on the shortest path from t to r in F.) Now the goal is to explore the bag B_v using

its backbone $P = (v_1, \cdots, v_\ell)$ ($\ell < k$), until the message finds an ancestor of t in F.

In this case we put the message on the backbone, and then explore the backbone using $Backbone(v)$ copied in $path$ in the header of the message. Using $back(v) = (p, j) \in Table(v)$, pos is set to j. If $p = \emptyset$ then the message is already on the backbone. Otherwise, the message is sent over the port p. Recall that by the definition of $back(v)$, port p leads to $v_j \in P$. The idea is to explore the neighborhoods of vertices on the backbone, starting from v_1. Note that in what follows $path \neq \emptyset$ and $pos \neq -1$.

- Else, if $start = 0$ (This is the case initially), then the message is at $v = v_j \in P$ and pos indicates the value of j. Moreover, in the field $path$ of the header, there are the port-numbers allowing to follow P. If $pos > 1$ then $pos = j - 1$ is set and the message follows the corresponding port-number $p_{e_{j-1},v_j} \in Backbone(v_j)$ to reach v_{j-1}. Otherwise, $start$ is set to 1, $cnt = d_F(v_1)$ and $cnt' = d_G(v_1) + 1$.

- Else, if $start = 1$, then the exploration of a bag containing an ancestor of t (or t itself) has begun. The key point is that any ancestor w of t in F satisfies that $Id_F(w) \leq Id_F(t) \leq Id_F(w) + w_F(w) - 1$ by Lemma 30. Using this property, for each vertex v_j of the backbone $P = (v_1, \cdots, v_\ell)$, the message visits v_j first. If v_j is an ancestor of t or $v_j = t$ then we are in the first case; otherwise the message is sent to the parent of v_j in F. If v_j's parent is an ancestor of t (or t itself) then we are in the first case; otherwise we explore $N_{\overline{F}}(v_j)$ by binary search. Notice that the other neighbors of v_j are its descendants in F, so if t has an ancestor among them, then v_j also is an ancestor of t.

  - If $cnt = cnt' - 1$, the neighborhood of the current vertex $v = v_j$, where $j = pos$, has already been explored and no ancestor of t has been found. In that case, using $path$, the message goes to v_{j+1} the next vertex in the backbone. Then pos is set to $j + 1$.
  - Otherwise, let $pn = \lfloor \frac{cnt' + cnt}{2} \rfloor$. The message takes port-number pn from v towards vertex w. If w is an ancestor of t, we go to the first case of the algorithm. Otherwise, the message goes back to $v = v_j$. This is possible since the vertex w knows the port over which the message arrives. Moreover, if $Id_F(t) > Id_F(w) + w_F(w) - 1$, then cnt is set to pn and cnt' is set to pn otherwise.

The fact that the message eventually reaches its destination follows from the above description. Moreover, the computation of the next hop and the modification of the header clearly takes time $O(1)$.

4.4 Performance of the routing scheme

In this section, we give an upper bound on the additive stretch of the routing scheme described in previous section.

Lemma 31. *The routing scheme has additive stretch* $\leq 2k(\lceil \log \Delta \rceil + \frac{5}{2}) - 2\lceil \log \Delta \rceil - 4$.

Proof. Let s be the source and t be the destination. Recall the main idea of the algorithm: We route along the path from s to r in tree F until we arrive a vertex x, whose bag B_x is an ancestor of t's bag B_t in tree T. Then applying binary search algorithm, we search in the bag B_x for a vertex y, which is an ancestor of t in tree F. In the end, we route from y to t in tree F.

Because F is a BFS tree and x is an ancestor of s in F, the length of the path followed by the message from s to x is $d(s,x)$, the distance between s and x in G. Similarly, because y is an ancestor of t in F, the length of the path followed by the message from y to t is $d(y,t)$. Let $track(x,y)$ be the length of the path followed by the message in B_x from x to y. Therefore, the length of the path followed by the message from s to t is $d(s,x) + track(x,y) + d(y,d)$.

From the binary search algorithm, for any vertex of the backbone, the message visits at most $\lceil \log \Delta \rceil$ neighbors and this causes a path of length $2\lceil \log \Delta \rceil$. There are at most $k-1$ vertices on the backbone of the bag B_x. The worst case occurs when x is the neighbor of the last vertex of the backbone v_l, for $l \leq k-1$, then the message goes to the first vertex of the backbone, v_1, while y is a neighbor of v_l. After arriving at x, the message goes to v_1, i.e., visits $l \leq k-1$ vertices, then it visits $\lceil \log \Delta \rceil$ neighbors of each of the $l \leq k-1$ vertices of the backbone and y is the last vertex visited. Therefore, $track(u,a) \leq 2k(\lceil \log \Delta \rceil + 1) - 2\lceil \log \Delta \rceil - 4$. Then it is sufficient to prove $d(s,x) + d(y,t) \leq d(s,t) + 3k$.

If B_s is an ancestor of B_t, then $x = s$ and $d(s,x) = 0$. Moreover, if $B_t = B_x$, $d(y,t) = 0$. Otherwise, let B be the nearest common ancestor of B_s and B_t in the tree-decomposition T. Let Q be a shortest path between s and t. Because the set of vertices in B separates s from t in G, let x' be the first vertex of Q in B and let y' the last vertex of Q in B. Let $Q = Q_1 \cup Q_2 \cup Q_3$ where Q_1 is the subpath of Q from s to x', Q_2 is the subpath of Q from x' to y' and Q_3 is the subpath of Q from y' to t. Note that because each bag has diameter at most k, $d(x',y') \leq k$.

We first show that $x \in B$. If $B_x = B$, it is trivially the case. Let P_x be the path followed from s to x. Since B_x is an ancestor of B, B separates s from x. Therefore, $P_x \cap B \neq \emptyset$. Let h be the first vertex of P_x in B. Since $h \in B$, the highest bag containing h is an common ancestor of B_s and B_t. Therefore, when arriving at h, the message must explore B_h. Hence, we have $h = x \in B$.

Finally, since $x \in B$, $d(x,x') \leq k$. Moreover, $y \in B_x$ therefore $d(y,x) \leq k$. Thus, $d(y,y') \leq d(y,x) + d(x,x') + |Q_2| \leq 2k + |Q_2|$. Finally, $d(s,x) \leq d(s,x') + d(x',x) \leq k + |Q_1|$ and $d(y,t) \leq d(y,y') + d(y',t) \leq 2k + |Q_2| + |Q_3|$. Therefore, $d(s,x) + d(y,t) \leq |Q_1| + |Q_2| + |Q_3| + 3k \leq |Q| + 3k = d(s,t) + 3k$. □

4.5 Perspectives

A k-good tree decomposition is applied to design a compact routing scheme with routing tables, addresses and headers of size $O(k \log \Delta + \log n)$ bits and achieving an additive stretch of $O(k \log \Delta)$. A first step improvement can be to reduce the $O(k \cdot \log \Delta)$ additive stretch due to the dichotomic search phase of our routing scheme.

To make this compact routing more practical, the second challenge is to design a distribute algorithm to construct the routing tables of the scheme. So it is interesting to find a k-good tree decomposition for a given graph and integer k in a distributed way.

Last but not least, our routing scheme is labelled routing scheme, which limits its applicability to static topologies networks. Then it is not applicable in dynamic network, e.g. the Internet. It would be more useful to modify the scheme to be name-independent.

CHAPTER 5

Prize Collecting Steiner Tree Problem on Partial 2-Trees

Contents

This chapter studies the *Prize Collecting Steiner Tree (PCST)* problem. Given a connected graph $G = (V, E)$ with a nonnegative cost for each edge in E, a nonnegative prize for each vertex in V, and a target set $V' \subseteq V$, the PCST problem is to find a subtree T of G interconnecting all vertices of V' such that the total cost on edges in T minus the total prize at vertices in T is minimized[1]. It is a generalization of the classical *Steiner Minimum Tree (SMT)* problem [HR92], in which each vertex is associated with a prize zero.

For instance, a typical application of PCST occurs when a natural gas provider wants to build a most profitable transportation system for natural gas delivery from a station to some customers on scattered locations, where each link (segment of pipeline) is associated with a cost which is incurred if the link is installed, and each location is associated with a profit which is obtained if the location is connected to the station by links installed. Moreover, the transportation system is required to contain some specified customers. This problem can be formulized as a PCST problem. The graph is the complete graph on the vertex sets consisting of the gas station and all customers. The target set includes the gas station and the specific customers needed to be contained in the transportation system. The prize of each customer is the profit they pay and the prize of the station can be set to be 0. The cost of each edge is the cost of the pipeline to connect the two endpoints. Then we see that a most profitable transportation system corresponds to an optimal solution of the PCST problem. We design linear algorithm for solving the PCST problem in the class of graphs of treewidth at most 2.

[1] There are several variations of PCST problem, see in e.g. [JMP00]. We only consider this one in the thesis.

In real life, we may not know exactly the information we need. Instead, we can estimate the range, i.e. an upper bound and a lower bound, of the required data. For instance, in the above example, the cost of a pipeline e may be any value $c_e \in [c_e^-, c_e^+]$; the profit from a customer v may be any value $p_v \in [p_v^-, p_v^+]$. So it is interesting to study the PCST problem with interval data on the costs of the edges and prizes of the vertices. We propose two risk models for the PCST problem with interval data and solve them in polynomial time in the class of graphs of treewidth at most 2.

The results of this chapter is a collaboration with X. Hu, X. Chen, E. A. Miranda and A. C. Vejar. It appears in a publication in the conference AAIM 2010 [c-AMCC$^+$10] and in the journal Acta Mathematicae Applicatae Sinica [j-AMCC$^+$14].

5.1 Introduction

The PCST problem introduced by Balas [Bal89] has been extensively studied in the areas of computer science and operations research. For example, using the PCST model, [PDL10] designed a leakage detection system[2] for finding the optimal location of detectors and their corresponding transponders to provide a desired coverage under budget constraints in the water distribution network of the Swiss city, Lausanne; [LWP$^+$06] carried out a concrete application of the PCST problem to the design of fiber optic networks for some German cities. Both the trade-off between connection costs (represented by edge costs) and customers revenues (represented by vertex prizes) and the goal of establishing the most profitable network were perfectly modeled by the PCST problem. Moreover, the PCST problem is applied in bioinformatics. Using the algorithmic framework in [LWP$^+$06], [DKR$^+$08] solved the problem of finding functional modules in protein-to-protein interaction networks by modeling it as the PCST problem. Also, [BBBZ09] and [BBBB$^+$11] used the PCST model to solve some problems in cell communication.

Since the *Steiner Minimum Tree* (SMT) problem is NP-complete in general [Kar72], even in planar graphs [GJ77], so is the PCST problem. In [GW95], Goemans and Williamson presented a 2-approximation $O(n^3 \log n)$-time algorithm for the PCST problem in general graphs. The time complexity of this approximation algorithm was improved to $O(n^2 \log n)$ in [FFFdP07].

To obtain optimal solutions in polynomial time, it is helpful to consider some specific structural properties of the graphs. Telecommunication networks typically possess certain sparse and planar structural properties [MS89], their representations as *series-parallel graphs*[3] are often convenient [Rag04], offering clearer representations of real-world instances. In particular, the densest (maximum number of edges with fixed number of ver-

[2] It is used for detecting leakages in the water distribution networks.

[3] A *series parallel graph* is a multi graph with two distinguished vertices, called *source* s and *sink* t, which can be formed with the following rules: (1) A graph with two vertices: source s and sink t and one edge st is a series parallel graph. (2) If for $i = 1,2$, $G_i = (V_i, E_i)$ with source s_i and sink t_i are series parallel graphs, then the graph obtained by taking the disjoint union of G_1 and G_2 and then identifying t_1 and s_2 (resp. identifying s_1 and s_2 and identifying t_1 and t_2) is a series parallel graph with source s_1 (resp. s_1 and s_2) and sink t_2 (resp. t_1 and t_2). Note that series parallel graphs are proper subset of partial 2-trees. For instance, $K_{1,3}$ is not a series parallel graph.

tices) series-parallel graphs, known as *2-trees* (see formal definition in next section) and independently reliable networks [WC83], play an important role in the reliable broadcasting problem on independently reliable networks in which all pairs of nodes can communicate as long as the failures of nodes and edges are isolated [BSK94]. It is well-known that the class of series-parallel graphs is a subclass of graphs of treewidth at most 2 [Bod98], called *partial 2-trees* for short in the following. When restricted to partial 2-trees, [WC83] proved that the SMT problem is polynomial-time solvable. In this chapter, we extend their approach to an efficient algorithm for the PCST problem on partial 2-trees. The reader is referred to [YT94, KZ06, WNC08, RSS11] for more research and applications on series-parallel networks (or partial 2-trees)[4].

In real-world applications, the information or data, e.g. the costs and the prizes in the PCST problem, may be uncertain. One of the simplest form of the uncertainty representation is to specify the data as closed intervals. A lot of research have been done about *interval combinatorial optimization problems* [KZ10], such as *shortest path, minimum spanning tree* [AL04], and *minimum-cut* [ABV08] with the edge-costs belonging to some given intervals, which are used to indicate the ranges of the edge-costs. Most of these literature has been developed under the name of *robust optimization*, in which one optimizes against the worst instance that might arise among all the possible values in the given intervals [KY10]. One of the most popular objectives in robust optimization is to find a solution that minimizes the maximum *regret*, which is the maximum difference between the value of any solution with any value realizations and the value of the optimal solution under this value realization. Despite the popularity, many robust optimization problems, such as the robust shortest path and robust spanning tree problems [AL04, Zie04, AVH04], suffer from two major drawbacks: they are NP-hard even though their deterministic counterparts are polynomial-time solvable; their solutions tend to be over-conservative, as it considers the maximum regret among all the possible values.

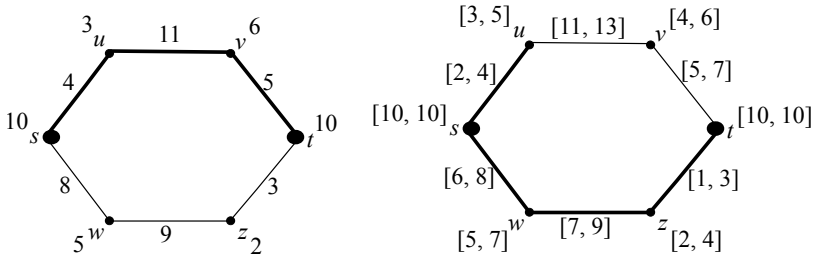


Figure 5.1: The number or interval beside each vertex (resp. edge) is its prize (resp. cost) or prize interval (resp. cost interval). The terminal set is $\{s,t\}$.

See a simple example for explaining the PCST problem and robust PCST problem in Fig. 5.1. In (PCST), the optimal solution is the subtree T_1 induced by $\{s,u,v,t\}$, whose value is -9. In (Robust PCST), the optimal solution is the subtree T_2 induced by $\{u,s,w,z,t\}$, whose maximum regret is 3. This maximum regret achieves when the cost

[4] A commonly repeated mistake is that a graph is series parallel graph if and only it is partial 2-trees.

(resp. prize) of any edge (resp. vertex) in T_2 takes the upper (resp. lower) bound of its interval; and the cost (resp. prize) of any other edge (resp. vertex) takes the lower (resp. upper) bound of its interval. With these realized costs and prizes, the optimal solution is the subtree T_1, whose value is -9 and the value of T_1 is -6. So the regret is 3. Actually, in the same way, we can compute the maximum regret of T_1 is 14.

To address the tractability and over-conservatism of the robust solution, [BS03] proposed an approach to controlling the degree of conservation of the solution by regulating the number of the values which are allowed to vary in the given intervals. They established a bounded probability of their robust solutions being infeasible, but their robust solutions still might be far from optimal.

Along a different line, [CHH09] and [Hu10] proposed two novel models for minimum spanning tree and shortest path problems with interval data on the edge costs. In these problems, each edge e is associated with a cost interval $[c_e^-, c_e^+]$. The solutions of these models are paths or trees together with the cost $x_e \in [c_e^-, c_e^+]$ for each edge e on the paths or trees. (Note that these models are different from the classical robust optimization models mentioned above, in which the solution are only paths or trees, not including an exact cost for each edge on the paths or trees.) The risk of edge e with cost x_e is quantified as $(c_e^+ - x_e)/(c_e^+ - c_e^-)$, which is consistent with the common sense that higher expense often brings about more satisfactory service (prevention of malfunction). The two models are to find solutions that minimizes the maximum risk or the sum of the risks over all the edges of the solution respectively under the constraint that the total cost of the solution (the sum of the costs of edges on the path or tree) is at most a given bound. The shortest path and minimum spanning tree problems with interval data under these two models are polynomial-time solvable, preserving the polynomial-time solvability of their deterministic counterparts. In this chapter, we will extend their approaches to the PCST problem in partial 2-trees by considering the interval costs on edges and interval prizes at vertices.

Taking a concrete instance, in the above PCST applications mentioned at the beginning of the chapter, the gas provider may spend c_e^+ dollars to install a gas pipe link e using the best materials to assure continuous transmission through e during a long period without any interruption for maintenance. On the other hand, the provider can also spend $c_e^-(< c_e^+)$ dollars to install the link e using ordinary materials while taking the risk of malfunction at e and service suspension for repair. Generally, lower expense on link construction could lead to higher risk of transmission malfunction. Similarly, if the gas provider wants to collect the highest possible prize p_v^+ from the customer at location v for the gas transmitted to v, then the provider faces the highest risk of rejection by customer at v because of the existence of competitors who sell the same kind of products. Usually, the smaller prize is demanded, the smaller the risk of being rejected is taken. For easy description, in the reminder of the chapter we use the *risks of edges* (resp. *vertices*) to denote the risk of malfunction at links (resp. rejection by customers). Naturally, it is desirable to make good balances between low net expense and small risk. Under budget constraint, the gas provider may have full control over the edge payments and can ask for any reasonable prize at any reachable vertex (customer). Our work contributes to modeling this kind of practical trade-off between the expense and risk and minimizing the risk under budget restriction.

Compared to the models in [CHH09] and [Hu10], our PCST models consider not only

the risks of edges, but also the risks of vertices, which are not studied in the previous models. Vertices behave quite differently from edges, and turn out less amenable in combinatorial optimization. (A typical example consists of the minimum vertex cover problem, which is NP-hard, and the minimum edge cover problem equivalently the maximum matching problem, which is polynomial time solvable.) Algorithmic approaches successful in dealing with edges do not always work for vertices. Our success in coping with risks of vertices relies on exploiting the structural properties of partial 2-trees.

In Section 5.2, there is a linear-time algorithm for the PCST problem on partial 2-trees. In Section 5.3, we establish the min-max risk model and min-sum risk model for the PCST problem with interval data, and propose two polynomial-time algorithms for the PCST problem on partial 2-trees under these two models.

5.2 Dynamic Programming for PCST on Partial 2-Trees

In this section, we present a linear time dynamic programming to solve the PCST problem on partial 2-trees.

5.2.1 Preliminary Definitions and Notations

Let us first see some definitions and notations in this subsection.

In the PCST problem, given a connected graph $G = (V, E)$, each vertex $v \in V$ is assigned a nonnegative prize $p_v \in \mathbb{R}_+$, and each edge $e \in E$ is assigned a nonnegative cost $c_e \in \mathbb{R}_+$. For any subgraph S of G, we abbreviate $\sum_{e \in E(S)} c_e$ to $c(S)$, and $\sum_{v \in V(S)} p_v$ to $p(S)$. Let $c \in \mathbb{R}_+^E, p \in \mathbb{R}_+^V$ be the vectors of edge costs and vertex prizes in G. The *value* of subgraph S is defined as

$$\nu(S, c, p) \equiv c(S) - p(S).$$

In the definition of the PCST problem, the input includes a *target set* $V' \subseteq V$, also called a *terminal set*. The objective of the PCST problem is to find a tree in G such that it spans V' and its value is minimum among all trees in G spanning V':

$$\text{(PCST)} \qquad \min \left\{ \nu(T, c, p) \mid T \text{ is a tree in } G \text{ and } V' \subseteq V(T) \right\}$$

Such a tree is called an *optimal* PCST in $(G, V'; c, p)$ or simply in G, and denoted by $T_{opt}(G, V'; c, p)$.

In the following lemma, we show that the PCST problem $(G, V'; c, p)$ can be transformed to the PCST problem $(G, \emptyset; c, p')$ by reassigning to each vertex in V' a sufficiently large prize.

Lemma 32. *Given $G = (V, E)$ with target set $V' \subseteq V$, $c \in \mathbb{R}_+^E, p \in \mathbb{R}_+^V$, and a real number $M > c(G)$, let $p' \in \mathbb{R}_+^V$ be defined by $p'_v = M$ for every $v \in V'$, and $p'_v = p_v$ for every $v \in V \setminus V'$. If T^* is an optimal PCST in $(G, \emptyset; c, p')$, then T^* is an optimal PCST in $(G, V'; c, p)$.*

Proof. First, let us prove that $V' \subseteq V(T^*)$. Suppose on the contrary that T^* does not contain some target vertex $u \in V' \setminus V(T^*)$. Let P be a path in G from u to a vertex in T^* such that P intersects T^* only at this vertex, written as t. It follows that the union of T^* and P, written

as $T^* \cup P$, is a tree in G, and its value $\nu(T^* \cup P, c, p')$ is smaller than $\nu(T^*, c, p')$ as seen from the following (in)equalities:

$$\begin{aligned} \nu(T^* \cup P, c, p') &= \nu(T^*, c, p') + \nu(P, c, p') + p_t \\ &\leq \nu(T^*, c, p') - p_u + c(E) = \nu(T^*, c, p') - M + c(G) < \nu(T^*, c, p') \end{aligned}$$

The contradiction to the optimality of T^* proves $V' \subseteq V(T^*)$.

Now, it is sufficient to prove that any subtree spanning V' in G has value at least the value of T^*. Suppose on the contrary that there exists a tree T' in G with $V(T') \supseteq V'$ and $\nu(T', c, p) < \nu(T^*, c, p)$. Then

$\nu(T', c, p') = \nu(T', c, p) + p(V') - p'(V') < \nu(T^*, c, p) + p(V') - p'(V') = \nu(T^*, c, p')$,
which contradicts the optimality of T^*. So no such a tree T' exists. □

Remark 1. *From Lemma 32, we just focus on the PCST problem with empty terminal set in the rest of the chapter. We denote its optimal solution as* $T_{opt}(G, c, p)$.

There is a linear algorithm for the PCST problem in bounded treewidth graphs by applying the dynamic programming based on a tree decomposition of bounded width. Given a tree decomposition of bounded width $(T, \mathcal{X})$ of a graph $G = (V, E)$, root T at a node $r \in V(T)$. The dynamic programming processes from the leaves of T to the root r. In each step, a bag $X \in \mathcal{X}$ is considered. It computes a set of sub-solutions, i.e., forests, in the subgraph of G, induced by all vertices in bag X and in any descendant-bag of X in T. The number of forests and the time complexity of computing these forests are both functions of the width of the tree decomposition. In the end, when the root bag X_r is considered, they choose a tree of optimal value among the computed forests of the graph G. So the PCST problem is solved in $O(f(tw(G))|V(G)|)$ time, where f is a function of the treewidth of the graph G at most $(tw(G))^{O(tw(G))}$.

We see that the base of the above algorithm is a tree decomposition of bounded width of the graph. As mentioned in Chapter 1, the fastest algorithm, as far as we know, for computing a tree decomposition of width $O(k)$ of a graph of treewidth at most $O(k)$ is in time $2^{O(k)}n$. So the final time complexity of the above algorithm is still at most $O((tw(G))^{O(tw(G))}|V(G)|)S$.

In this chapter, we study the PCST problem in graphs of treewidth at most 2. Our algorithm is less unified, because it takes advantage of the structural properties of the graphs of treewidth at most 2, which may not be extended easily to the graphs of treewidth at least 3. First let us see the definition of the densest graphs of treewidth at most 2, called 2-*tree*s.

The graph class of 2-*tree*s is defined recursively as follows: An edge is a 2-tree. Given a 2-tree, picking an edge uv in it, adding a new vertex z adjacent with both u and v yields a 2-tree. See Fig.5.2 for an illustration, where vertex z has degree 2 in the new 2-tree. 2-trees are the densest graphs of treewidth 2 in the sense that no edge can be added in a 2-tree without increasing the treewidth. The constructive definition guarantees the following.

Figure 5.2: Construction of 2-trees.

Remark 2. *Every 2-tree $G = (V,E)$ is either an edge or has at least one vertex $z \in V$ of degree 2 which is contained in a triangle (a complete graph of three vertices). Moreover $G \setminus \{z\}$ is still a 2-tree.*

Fact 3. *[Bod98] Any graph of treewidth at most 2 is a spanning subgraph of a 2-tree.*

The following result plays an important role in our algorithm for PCST problem in graphs of treewidth at most 2.

Lemma 33. *[WC83] In linear time, edges can be added to a graph of treewidth at most 2 to construct a 2-tree whenever this is possible.*

Now we show that the PCST problem in any graph of treewidth at most 2 can be transformed to the PCST problem in a 2-tree as follows:

Lemma 34. *Given a graph $G = (V,E)$ of treewidth at most 2 and cost vector $c \in \mathbb{R}_+^E$ and prize vector $p \in \mathbb{R}_+^V$, and a real number $L > p(G)$, let $G' = (V, E \cup E')$ be a 2-tree obtained by adding edges in G. Let $c' \in \mathbb{R}_+^{E \cup E'}$ be defined by $c'_e = L$ for every $e \in E'$, and $c'_e = c_e$ for every $e \in E$. If T^* is an optimal PCST in (G', c', p), then T^* is an optimal PCST in (G, c, p).*

Proof. It is sufficient to prove that T^* does not contain any edge in E'. Suppose it is not true and there is an edge $e' \in E' \cap E(T^*)$. Then $\nu(T*, c', p) > c_{e'} - p(G) > 0$. Take a tree T containing only one vertex $v \in V$. Then $\nu(T, c', p) = -p_v \leq 0 < \nu(T^*, c', p)$. It is a contradiction. □

From Lemma 33 and 34, we have that:

Corollary 10. *From any algorithm for PCST problem in 2-tree with time complexity $f(n)$, we can obtain an algorithm for PCST problem in the class of graphs of treewidth at most 2 with time complexity $f(n) + n$, where n is the number of vertices in the graph.*

So in the following, we only describe our algorithms with input graphs being 2-trees.

From Remark 2, any 2-tree can be reduced to a single edge by deleting recursively a vertex z of degree 2 contained in a triangle $\{u, v, z\}$. The algorithm in [WC83] for SMT problem in the class of 2-trees applied this vertex elimination procedure. During each elimination, it summarizes information about the triangle $\{u, v, z\}$ on two arcs (u, v) and (v, u), where z has degree 2. This summary information encodes information about the sub-solutions of SMT problem, i.e. forests, in the subgraph of the 2-tree, which has been reduced to the edge uv so far. In fact, from the vertex elimination procedure, a tree decomposition of width 2 can be constructed. The idea of the algorithm in [WC83] is essentially the same as the classical dynamic programming algorithm for SMT problem in bounded treewidth graphs.

We generalize the algorithm in [WC83] for SMT problem in 2-trees to solve the PCST problem in 2-trees. Since we consider empty terminal set in the PCST problem, our algorithm needs to encode less information than their algorithm during each elimination. We keep the notations in [WC83] and explain them as follows.

Given a 2-tree $G = (V,E)$ and cost vector $c \in \mathbb{R}_+^E$ and prize vector $p \in \mathbb{R}_+^V$, order all vertices in V as $z_1, \ldots, z_n$ such that each vertex z_i, for $1 \leq i \leq n-2$, has degree 2 and is

contained in a triangle, i.e., complete graphs on three vertices, in the induced subgraph $G_i \equiv G[\{z_i, \ldots, z_n\}]$. Note that G_i is still a 2-tree.

Definition 4. *For $1 \leq i \leq n-2$ and each pair (u,v) corresponding to an edge $uv \in E(G_i)$, define the subgraph $H_i(u,v)$ as follows:*

- *For $i = 1$, $H_1(u,v) = G[\{u,v\}]$;*
- *For $2 \leq i \leq n-2$, $H_i(u,v) = H_{i-1}(u,v) \cup H_{i-1}(u,z_{i-1}) \cup H_{i-1}(z_{i-1},v)$ if u,v are the two neighbor of z_{i-1} in G_{i-1}; and $H_i(u,v) = H_{i-1}(u,v)$ otherwise.*

By induction, it is easy to see that, for any $2 \leq i \leq n-2$, $H_i(u,v) \setminus \{u,v\}$ only contains some vertices in $\{z_1, \ldots, z_{i-1}\}$. So for $2 \leq i \leq n-2$ $H_i(u,v)$ has been reduced to the edge uv after deleting $z_1, \ldots, z_{i-1}$. From the definitions, we can get the following lemma, which is crucial for our dynamic programming algorithm.

Lemma 35. *For each $i = 1, \ldots, n-2$ and each edge $uv \in E(G_i)$, $\{u,v\}$ separates $H_i(u,v)$ from $G \setminus H_i(u,v)$.*

Proof. It is sufficient to prove that any vertex in $H_i(u,v) \setminus \{u,v\}$ is not adjacent to any vertex in $G \setminus H_i(u,v)$, for each $i = 1, \ldots, n-2$ and each edge $uv \in E(G_i)$. For $i = 1$, $G_1 = G$ and $H_1(u,v) = G[u,v]$ for each pair (u,v) corresponding to an edge $uv \in E(G)$, so it is true. Assume that it is true for any $1 \leq j \leq i-1$. We prove that it is true for $j = i$ in the following.

If u,v are not the two neighbor of z_{i-1} in G_{i-1}, then $H_i(u,v) = H_{i-1}(u,v)$. From the assumption of the induction, it is true.

Otherwise u,v are the two neighbor of z_{i-1} in G_{i-1} and $H_i(u,v) = H_{i-1}(u,v) \cup H_{i-1}(u,z_{i-1}) \cup H_{i-1}(z_{i-1},v)$. Let z be any vertex in $H_i(u,v) \setminus \{u,v\}$. If $z \in V(H_{i-1}(u,v))$, then from the assumption of the induction, z is not adjacent to any vertex in $G \setminus H_{i-1}(u,v) \supseteq G \setminus H_i(u,v)$. So z is not adjacent to any vertex in $G \setminus H_i(u,v)$. Similarly, if $z \in V(H_{i-1}(u,z_{i-1}))$ or $z \in V(H_{i-1}(z_{i-1},v))$, then z is not adjacent to any vertex in $G \setminus H_i(u,v)$. Thus any vertex in $H_i(u,v) \setminus \{u,v\}$ is not adjacent to any vertex in $G \setminus H_i(u,v)$. □

Remark 3. *By induction, it is easy to see that $\bigcup_{uv \in E(G_i)} H_i(u,v) = G$ for any $1 \leq i \leq n-2$. So $H_{n-2}(z_{n-1}, z_n) = G$.*

Definition 5. *For each $1 \leq i \leq n-2$ and each (ordered) pair (u,v) corresponding to an edge $uv \in G_i$, we will associate five measures defined as follows, which summarize the values incurred in the subgraph $H_i(u,v)$.*

(i) $st_i(u,v)$ is the minimum value of a tree in $H_i(u,v)$ containing both u and v; $T_{st_i(u,v)}$ denotes such a tree;

(ii) $dt_i(u,v)$ is the minimum value of the forest consisting of two vertex-disjoint trees in $H_i(u,v)$, one containing u and the other containing v; $T_{dt_i(u,v)}$ denotes such a forest;

(iii) $yn_i(u,v)$ is the minimum value of a tree in $H_i(u,v)$ containing u but not v; $T_{yn_i(u,v)}$ denotes such a tree;

(iv) $ny_i(u,v)$ is the minimum value of a tree in $H_i(u,v)$ containing v but not u; $T_{ny_i(u,v)}$ denotes such a tree;

(v) $nn_i(u,v)$ *is the minimum value of a tree in* $H_i(u,v)$ *containing neither u nor v;* $T_{nn_i(u,v)}$ *denotes such a tree.*

Note that $T_{nn_i(u,v)}$ might be empty, i.e., possibly $T_{nn_i(u,v)} = (\emptyset,\emptyset)$. Since edge uv can also be written as vu, we have st_i, dt_i, nn_i are symmetric, but $yn_i(u,v) = ny_i(v,u), ny_i(u,v) = yn_i(v,u)$.

Then we have the following lemma.

Lemma 36. *The optimal value of the PCST problem* (G,c,p) *is:* $\min\{st_{n-2}(z_{n-1},z_n), yn_{n-2}(z_{n-1},z_n), ny_{n-2}(z_{n-1},z_n), nn_{n-2}(z_{n-1},z_n)\}$. *Moreover, the tree* $T \in \{T_{st_{n-2}(z_{n-1},z_n)}, T_{yn_{n-2}(z_{n-1},z_n)}, T_{ny_{n-2}(z_{n-1},z_n)}, T_{nn_{n-2}(z_{n-1},z_n)}\}$ *with the minimum value is an optimal solution of the PCST problem* (G,c,p).

Proof. From Remark 3, $st_{n-2}(z_{n-1},z_n)$ is the minimum value of a tree in G containing both z_{n-1} and z_n; $yn_{n-2}(z_{n-1},z_n)$ (resp. $ny_{n-2}(z_{n-1},z_n)$) is the minimum value of a tree in G containing z_{n-1} (resp. v) but not z_n (resp. z_{n-1}); and $nn_{n-2}(z_{n-1},z_n)$ is the minimum value of a tree in G containing neither z_{n-1} nor z_n. Any $T_{opt}(G,c,p)$ has to satisfy one of the following: it contains both z_{n-1} and z_n; it contains only one of them; it contains neither of them. So its value is the minimum value of the four values $st_{n-2}(z_{n-1},z_n), yn_{n-2}(z_{n-1},z_n), ny_{n-2}(z_{n-1},z_n), nn_{n-2}(z_{n-1},z_n)$. Then the tree $T \in \{T_{st_{n-2}(z_{n-1},z_n)}, T_{yn_{n-2}(z_{n-1},z_n)}, T_{ny_{n-2}(z_{n-1},z_n)}, T_{nn_{n-2}(z_{n-1},z_n)}\}$ with the minimum value is an optimal solution of the PCST problem (G,c,p). □

For brevity, we define $\Pi_i \equiv \{st_i, dt_i, nv_i, un_i, nn_i\}$ for $i = 1,\ldots,n-2$. For $i = 1,\ldots,n-2$ and $\pi_i \in \Pi_i$, $\pi_i(u,v)$ denotes any measure of the pair (u,v) as defined above. For any edge $uv \in E(G)$, $H_1(u,v)$ only contains one edge uv. It is easy to find the five measures and its corresponding forests as follows:

$$\begin{array}{ll} st_1(u,v) = c_{uv} - p_u - p_v, & T_{st_1(u,v)} = (\{u,v\}, uv); \\ dt_1(u,v) = -p_u - p_v, & T_{dt_1(u,v)} = (\{u,v\}, \emptyset); \\ yn_1(u,v) = -p_u, & T_{yn_1(u,v)} = (\{u\}, \emptyset); \\ ny_1(u,v) = -p_v, & T_{ny_1(u,v)} = (\{v\}, \emptyset); \\ nn_1(u,v) = 0, & T_{nn_1(u,v)} = (\emptyset, \emptyset). \end{array} \tag{1}$$

These are the initial state in the dynamic programming, which is presented in the next subsection.

For any set $\mathcal{S}$ of subgraphs of G, we use $Min(\mathcal{S})$ to denote an arbitrary subgraph $R \in \mathcal{S}$ whose value $v(R,c,p)$ is the minimum among all elements of $\mathcal{S}$.

5.2.2 Dynamic Programming

In this subsection, we describe the dynamic programming algorithm for the PCST problem in 2-trees. From Lemma 36, to solve the PCST problem (G,c,p), it is sufficient to compute the measures $st_{n-2}(z_{n-1},z_n), yn_{n-2}(z_{n-1},z_n), ny_{n-2}(z_{n-1},z_n), nn_{n-2}(z_{n-1},z_n)$ of the pair (z_{n-1},z_n) and the corresponding trees achieving these values.

For $2 \le i \le n-2$ and any pair (u,v) corresponding to an edge $uv \in E(G_i)$, assume that any measure $\pi_i(u,v)$ for $\pi_i \in \Pi_i$, are computed already after the elimination

of $\{z_1,\ldots,z_{i-1}\}$. In the following, we see how to compute the measures $\pi_{i+1}(u,v)$, for any $\pi_{i+1}\in\Pi_{i+1}$ after the elimination of z_i in G_i.

Lemma 37. *For any pair (u,v) corresponding to an edge $uv\in E(G_{i+1})$, given $\pi_i(u,v)$, for any $\pi_i\in\Pi_i$, $st_{i+1}(u,v)$ and $T_{st_{i+1}(u,v)}$ can be computed as follows:*

- *If u,v are not the two neighbors of z_i in G_i, $st_{i+1}(u,v)=st_i(u,v)$ and $T_{st_{i+1}(u,v)}=T_{st_i(u,v)}$;*
- *Otherwise,* $$\begin{aligned} st_{i+1}(u,v)=\min\{&st_i(u,v)+yn_i(u,z_i)+ny_i(z_i,v)+p_u+p_v,\\ &st_i(u,v)+st_i(u,z_i)+dt_i(z_i,v)+p_u+p_{z_i}+p_v,\\ &st_i(u,v)+dt_i(u,z_i)+st_i(z_i,v)+p_u+p_{z_i}+p_v,\\ &dt_i(u,v)+st_i(u,z_i)+st_i(z_i,v)+p_u+p_{z_i}+p_v\},\end{aligned}$$ *and* $$\begin{aligned} T_{st_{i+1}(u,v)}=\min\{&T_{st_i(u,v)}\cup T_{yn_i(u,z_i)}\cup T_{ny_i(z_i,v)},\quad T_{st_i(u,v)}\cup T_{st_i(u,z_i)}\cup T_{dt_i(z_i,v)},\\ &T_{st_i(u,v)}\cup T_{dt_i(u,z_i)}\cup T_{st_i(z_i,v)},\quad T_{dt_i(u,v)}\cup T_{st_i(u,z_i)}\cup T_{st_i(z_i,v)}\}.\end{aligned}$$

Proof. Recall that $st_{i+1}(u,v)$ is the minimum value of a tree containing both u and v in the subgraph $H_{i+1}(u,v)$, denoted as $T_{st_{i+1}(u,v)}$.

If u,v are not the two neighbors of z_i in G_i, then $H_{i+1}(u,v)=H_i(u,v)$. So $st_{i+1}(u,v)=st_i(u,v)$ and $T_{st_{i+1}(u,v)}=T_{st_i(u,v)}$.

Otherwise, u,v are the two neighbors of z_i in G_i, then $H_{i+1}(u,v)=H_i(u,v)\cup H_i(u,z_i)\cup H_i(z_i,v)$. Let S be the triangle, i.e. a complete graph on three vertices, induced by $\{z_i,u,v\}$ in G_i. The intersection of S and any tree in $H_{i+1}(u,v)$ containing both u and v is one of the following cases: $S_1=S\setminus\{z_{i+1}\}$, $S_2=S\setminus\{z_{i+1}v_{i+1}\}$, $S_3=S\setminus\{z_{i+1}u_{i+1}\}$, $S_4=S\setminus\{u_{i+1}v_{i+1}\}$, shown in Fig. 5.3.

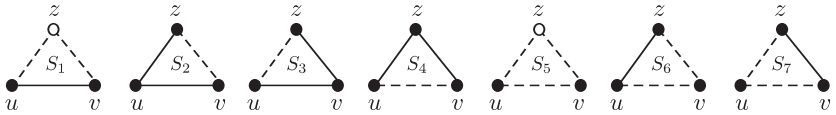


Figure 5.3: Forests in a triangle containing both u and v.

- If $T_{st_{i+1}(u,v)}\cap S=S_1$, then from Lemma 35, $T_{st_{i+1}(u,v)}\cap H_i(u,v)=T_{st_i(u,v)}$, $T_{st_{i+1}(u,v)}\cap H_i(u,z_i)=T_{yn_i(u,z_i)}$ and $T_{st_{i+1}(u,v)}\cap H_i(z_i,v)=T_{ny_i(z_i,v)}$. So $T_{st_{i+1}(u,v)}=T_{st_i(u,v)}\cup T_{yn_i(u,z_i)}\cup T_{ny_i(z_i,v)}$. Then $st_{i+1}(u,v)=st_i(u,v)+yn_i(u,z_i)+ny_i(z_i,v)+p_u+p_v$, since u (resp. v) is contained both in $T_{st_i(u,v)}$ and $T_{yn_i(u,z_i)}$ (resp. $T_{ny_i(z_i,v)}$).

  Similarly, we can prove that:

- If $T_{st_{i+1}(u,v)}\cap S=S_2$, then $T_{st_{i+1}(u,v)}=T_{st_i(u,v)}\cup T_{st_i(u,z_i)}\cup T_{dt_i(z_i,v)}$ and $st_{i+1}(u,v)=st_i(u,v)+st_i(u,z_i)+dt_i(z_i,v)+p_u+p_{z_i}+p_v$.
- If $T_{st_{i+1}(u,v)}\cap S=S_3$, then $T_{st_{i+1}(u,v)}=T_{st_i(u,v)}\cup T_{dt_i(u,z_i)}\cup T_{st_i(z_i,v)}$ and $st_{i+1}(u,v)=st_i(u,v)+dt_i(u,z_i)+st_i(z_i,v)+p_u+p_{z_i}+p_v$.
- If $T_{st_{i+1}(u,v)}\cap S=S_4$, then $T_{st_{i+1}(u,v)}=T_{dt_i(u,v)}\cup T_{st_i(u,z_i)}\cup T_{st_i(z_i,v)}$ and $st_{i+1}(u,v)=dt_i(u,v)+st_i(u,z_i)+st_i(z_i,v)+p_u+p_{z_i}+p_v$.

So $st_{i+1}(u,v)$ (resp. $T_{st_{i+1}(u,v)}$) is the minimum among the four cases. The lemma is proved. □

Lemma 38. *For any pair (u,v) corresponding to an edge $uv \in E(G_{i+1})$, given $\pi_i(u,v)$, for any $\pi_i \in \Pi_i$, $dt_{i+1}(u,v)$ and $T_{dt_{i+1}(u,v)}$ can be computed as follows:*

- *If u,v are not the two neighbors of z_i in G_i, $st_{i+1}(u,v) = dt_i(u,v)$ and $T_{dt_{i+1}(u,v)} = T_{dt_i(u,v)}$;*
- *Otherwise,* $$dt_{i+1}(u,v) = \min\{dt_i(u,v) + yn_i(u,z_i) + ny_i(z_i,v) + p_u + p_v,\\ dt_i(u,v) + st_i(u,z_i) + dt_i(z_i,v) + p_u + p_{z_i} + p_v,\\ dt_i(u,v) + dt_i(u,z_i) + st_i(z_i,v) + p_u + p_{z_i} + p_v,\},$$ *and* $$T_{dt_{i+1}(u,v)} = \min\{T_{dt_i(u,v)} \cup T_{yn_i(u,z_i)} \cup T_{ny_i(z_i,v)},\quad T_{dt_i(u,v)} \cup T_{st_i(u,z_i)} \cup T_{dt_i(z_i,v)},\\ T_{dt_i(u,v)} \cup T_{dt_i(u,z_i)} \cup T_{st_i(z_i,v)}\}.$$

Proof. Recall that $dt_{i+1}(u,v)$ is the minimum value of the forest consisting of two vertex-disjoint trees, one containing u and the other containing v, in the subgraph $H_{i+1}(u,v)$, denoted as $T_{dt_{i+1}(u,v)}$.

If u,v are not the two neighbors of z_i in G_i, then $H_{i+1}(u,v) = H_i(u,v)$. So $dt_{i+1}(u,v) = dt_i(u,v)$ and $T_{dt_{i+1}(u,v)} = T_{dt_i(u,v)}$.

Otherwise, u,v are the two neighbors of z_i in G_i, then $H_{i+1}(u,v) = H_i(u,v) \cup H_i(u,z_i) \cup H_i(z_i,v)$. Let S be the triangle, i.e. a complete graph on three vertices, induced by $\{z_i,u,v\}$ in G_i. The intersection of S and any forest consisting of two vertex-disjoint trees, one containing u and the other containing v, in $H_{i+1}(u,v)$ is one of the following cases: $S_5 = S \setminus \{z_{i+1}\} \setminus \{uv\}$, $S_6 = S \setminus \{z_{i+1}v_{i+1}, uv\}$, $S_7 = S \setminus \{z_{i+1}u_{i+1}, uv\}$, shown in Fig. 5.3.

- If $T_{dt_{i+1}(u,v)} \cap S = S_5$, then from Lemma 35, $T_{dt_{i+1}(u,v)} \cap H_i(u,v) = T_{dt_i(u,v)}$, $T_{dt_{i+1}(u,v)} \cap H_i(u,z_i) = T_{yn_i(u,z_i)}$ and $T_{dt_{i+1}(u,v)} \cap H_i(z_i,v) = T_{ny_i(z_i,v)}$. So $T_{dt_{i+1}(u,v)} = T_{dt_i(u,v)} \cup T_{yn_i(u,z_i)} \cup T_{ny_i(z_i,v)}$. Then $dt_{i+1}(u,v) = dt_i(u,v) + yn_i(u,z_i) + ny_i(z_i,v) + p_u + p_v$, since u (resp. v) is contained both in $T_{dt_i(u,v)}$ and $T_{yn_i(u,z_i)}$ (resp. $T_{ny_i(z_i,v)}$).

  Similarly, we can prove that:

- If $T_{dt_{i+1}(u,v)} \cap S = S_6$, then $T_{dt_{i+1}(u,v)} = T_{dt_i(u,v)} \cup T_{st_i(u,z_i)} \cup T_{dt_i(z_i,v)}$ and $dt_{i+1}(u,v) = dt_i(u,v) + st_i(u,z_i) + dt_i(z_i,v) + p_u + p_{z_i} + p_v$.
- If $T_{dt_{i+1}(u,v)} \cap S = S_7$, then $T_{dt_{i+1}(u,v)} = T_{dt_i(u,v)} \cup T_{dt_i(u,z_i)} \cup T_{st_i(z_i,v)}$ and $dt_{i+1}(u,v) = dt_i(u,v) + dt_i(u,z_i) + st_i(z_i,v) + p_u + p_{z_i} + p_v$.

So $dt_{i+1}(u,v)$ (resp. $T_{dt_{i+1}(u,v)}$) is the minimum among the three cases. The lemma is proved. □

Similarly, we can prove that:

Lemma 39. *For any pair (u,v) corresponding to an edge $uv \in E(G_{i+1})$, given $\pi_i(u,v)$, for any $\pi_i \in \Pi_i$, $\pi_{i+1}(u,v)$ and $T_{\pi_{i+1}(u,v)}$ for any $\pi_{i+1} \in \{yn_{i+1}, ny_{i+1}, nn_{i+1}\}$ can be computed as follows:*

- *If u, v are not the two neighbors of z_i in G_i, $\pi_{i+1}(u,v) = \pi_i(u,v)$ and $T_{\pi_{i+1}(u,v)} = T_{\pi_i(u,v)}$ for any $\pi_{i+1} \in \{yn_{i+1}, ny_{i+1}, nn_{i+1}\}$;*

- *Otherwise, we have:*

  - $yn_{i+1}(u,v) = \min\{yn_i(u,v) + yn_i(u,z_i) + p_u,\ yn_i(u,v) + st_i(u,z_i) + yn_i(z_i,v) + p_u + p_{z_i}\}$, *and* $T_{yn_{i+1}(u,v)} = \min\{T_{yn_i(u,v)} \cup T_{yn_i(u,z_i)},\ T_{yn_i(u,v)} \cup T_{st_i(u,z_i)} \cup T_{yn_i(z_i,v)}\}$;
  - $ny_{i+1}(u,v) = \min\{ny_i(u,v) + ny_i(z_i,v) + p_v,\ ny_i(u,v) + ny_i(u,z_i) + st_i(z_i,v) + p_{z_i} + p_v\}$, *and* $T_{ny_{i+1}(u,v)} = \min\{T_{ny_i(u,v)} \cup T_{ny_i(u,z_i)},\ T_{ny_i(u,v)} \cup T_{ny_i(u,z_i)} \cup T_{st_i(z_i,v)}\}$;
  - $nn_{i+1}(u,v) = \min\{nn_i(u,v),\ nn_i(u,z_i),\ nn_i(z_i,v),\ ny_i(u,z_i) + yn_i(z_i,v) + p_{z_i}\}$, *and* $T_{ny_{i+1}(u,v)} = \min\{T_{nn_i(u,v)},\ T_{nn_i(u,z_i)},\ T_{nn_i(z_i,v)},\ T_{ny_i(u,z_i)} \cup T_{yn_i(z_i,v)}\}$.

The following pseudo-code in Algorithm 6 gives the details of our algorithm.

Algorithm 6: Algorithm for PCST on 2-tree (ALG_PCST)

Input: 2-tree $G = (V,E)$ with $c \in \mathbb{R}_+^E, p \in \mathbb{R}_+^V$
Output: An optimal tree T^* of the PCST problem on (G,c,p) with optimal value ν^*

$G_1 \leftarrow G$;
$z_1 \leftarrow$ a vertex in G_1 of degree 2;
for *each pair* (u,v) *corresponding to an edge* uv *in* G_1 **do**
    $H_1(u,v) \leftarrow G[\{u,v\}]$;
    set $\pi_1(u,v)$ and its corresponding forest as in (1) for any $\pi_1 \in \Pi_1$;
$n \leftarrow |V(G)|$;
for $i = 2$ *to* $n-2$ *and* **do**
    $G_i \leftarrow G \setminus \{z_1, \ldots, z_{i-1}\}$;
    $z_i \leftarrow$ a vertex in G_i of degree 2;
    for *each pair* (u,v) *corresponding to an edge* uv *in* G_i **do**
        if u, v *are not the two neighbors of* z_{i-1} *in* G_{i-1} **then**
            $H_i(u,v) \leftarrow H_{i-1}(u,v)$;
        else
            $H_i(u,v) \leftarrow H_{i-1}(u,v) \cup H_{i-1}(u,z_{i-1}) \cup H_{i-1}(z_{i-1},v)$
        set $\pi_i(u,v)$ and its corresponding forest as in Lemma 37-39 for any $\pi_i \in \Pi_i$;
$\nu^* \leftarrow \min\{st_{n-2}(u,v), yn_{n-2}(u,v), ny_{n-2}(u,v), nn_{n-2}(u,v)\}$, where $u,v \in G \setminus G_{n-2}$;
$T^* \leftarrow T_{\pi_{n-2}(u,v)}$, where $\pi_{n-2} \in \{st_{n-2}, yn_{n-2}, ny_{n-2}, nn_{n-2}\}$ and $\pi(u,v) = \nu^*$;
return T^* and ν^*.

From Lemma 36-39, we get:

Theorem 15. *Given any PCST instance on a 2-tree of n vertices, Algorithm* ALG_PCST *outputs its optimal PCST and optimal value in $O(n)$ time.*

5.3 Risk Models for PCST Problem with Interval Data

In this section, we consider the PCST problem with interval data. We design two risk models for this problem and propose strongly polynomial-time algorithms in 2-trees.

Given an undirected graph $G=(V,E)$, each edge $e\in E$ is associated with a cost interval $[c_e^-,c_e^+]$, and each vertex $v\in V$ is associated with a prize interval $[p_v^-,p_v^+]$, where $0\le c_e^-\le c_e^+$ and $0\le p_v^-\le p_v^+$. These intervals indicate possible ranges of construction cost of edge $e\in E$ and collection prize of vertex $v\in V$, respectively.

We define the *risk* at edge e with cost $x_e\in[c_e^-,c_e^+]$ as[5]

$$r(x_e)\equiv\frac{c_e^+-x_e}{c_e^+-c_e^-},$$

and the *risk* at vertex v with prize $y_v\in[p_v^-,p_v^+]$ as

$$r(y_v)\equiv\frac{y_v-p_v^-}{p_v^+-p_v^-}.$$

We see that the risks $r(x_e)$, for any $e\in E$, and $r(y_v)$, for any $v\in V$, both range from 0 to 1. It is consistent with the sense that the higher cost $x_e\in[c_e^-,c_e^+]$ one pays to construct the link e, the lower risk it is that the link e breaks down; and that the lower prize $y_v\in[p_v^-,p_v^+]$ one asks for from the costumer v, the lower risk it is that the costumer v refuse the deal. In particular, $r(x_e)=0$ when $x_e=c_e^+$ (resp. $r(y_v)=0$ when $y_v=p_v^-$), meaning no risk occurs if the payment is high enough (the expected prize is low enough). On the other hand, $r(x_e)=1$ when $x_e=c_e^-$ (resp. $r(y_v)=1$ when $y_v=p_v^+$), meaning a full risk occurs at the lowest payment (resp. the highest prize).

Let $\mathscr{T}$ denote the set of trees, i.e., acyclic connected subgraphs, in G. We define the value of $T\in\mathscr{T}$ with charged payment $x\in\mathbb{R}_+^{E(T)}$ and collected prize $y\in\mathbb{R}_+^{V(T)}$ as

$$\nu(T,x,y)\equiv\textstyle\sum_{e\in E(T)}x_e-\sum_{v\in V(T)}y_v,$$

where by $x\in\mathbb{R}_+^{\emptyset}$ (in case of $E(T)=\emptyset$) we mean that x is a null vector.

Let B be a given budget. It is required that $T\in\mathscr{T}$ with charged payment x and collected prize y satisfy $\nu(T,x,y)\le\mathsf{B}$. So in case of negative B, the construction of tree T must make profit. To assure the feasibility of $\nu(T,x,y)\le\mathsf{B}$ subject to

$$x_e\in[c_e^-,c_e^+],\forall\ e\in E(T)\text{ and }y_v\in[p_v^-,p_v^+],\forall\ v\in V(T),\tag{2}$$

bound B obviously cannot be smaller than, B_{min}, the optimal value of the PCST problem with respect to $c_e=c_e^-$ for every $e\in E$ and $p_v=p_v^+$ for every $v\in V$. Meanwhile, $\mathsf{B}_{max}\equiv c^+(E)$ is a trivial upper bound on $\nu(T,x,y)$ for any $T\in\mathscr{T}$, $x\in\mathbb{R}_+^{E(T)}$ and $y\in\mathbb{R}_+^{V(T)}$ satisfying (2). We assume $\mathsf{B}\in[\mathsf{B}_{min},\mathsf{B}_{max}]$ throughout. Moreover, we assume that $\mathsf{B}<0$ to prevent the trivial solution $T=\emptyset$, which has value 0.

5.3.1 PCST Problem under Min-Max Risk Model

The PCST problem under min-max risk model, denoted by MMR_PCST, consists of finding a tree T along with payment x and prize y such that the maximum risk at edges and vertices in T is minimized and the value $\nu(T,x,y)$ is no greater than the given budget B. This

[5]For ease of description, we make the notational convention that $\frac{0}{0}=0$.

problem can be formulated as follows:

$$\begin{aligned}(\text{MMR\_PCST}) \quad \min_{T\in\mathscr{T}} \max_{e\in E(T),v\in V(T)} & \{\frac{c_e^+ - x_e}{c_e^+ - c_e^-}, \frac{y_v - p_v^-}{p_v^+ - p_v^-}\} \\ \text{s.t.} \quad & \nu(T,x,y) \le \mathsf{B}; \\ & x_e \in [c_e^-, c_e^+],\ \forall\, e \in E(T); \\ & y_v \in [p_v^-, p_v^+],\ \forall\, v \in V(T).\end{aligned}$$

Let (T^*,x^*,y^*) be an optimal solution to the MMR_PCST problem, where T^* is called an *optimal tree*. We reserve symbol r^* for the value $r_m(T^*,x^*,y^*)$ of the optimal solution, i.e.,

$$r^* \equiv r_m(T^*,x^*,y^*) \equiv \max_{e\in E(T^*),v\in V(T^*)} \{\frac{c_e^+ - x_e^*}{c_e^+ - c_e^-}, \frac{y_v^* - p_v^-}{p_v^+ - p_v^-}\}. \tag{3}$$

For example, given a graph G with cost intervals and prize interval as shown in Fig. 5.1 (Robust PCST), let the budget B is -12. The optimal solution of the MMR_PCST problem is (T^*,x^*,y^*), where T^* is the subtree induced by $\{s,w,z,t\}$, $x^* = (7,8,2)$ is the cost vector of the edges (sw,wz,zt) on T^* and $y^* = (10,6,3,10)$ is the prize vector of the vertices (s,w,z,t) on T^*. The optimal value $r^* = 0.5$ achieves at each vertex and edges on T^*.

The following lemma shows that (T^*,x^*,y^*) possesses an evenness property – the risks of edges and vertices are all equal, which will play an important role in our algorithm design.

Lemma 40 (Evenness property)**.** *For every edge e and every vertex v in T^*, either we have $c_e^- = c_e^+$ and $p_v^- = p_v^+$ or it holds that*

$$\frac{c_e^+ - x_e^*}{c_e^+ - c_e^-} = \frac{y_v^* - p_v^-}{p_v^+ - p_v^-} = r^*.$$

Proof. Suppose that the lemma is not true. Then there exists $f \in E(T^*)$ with $c_f^+ > c_f^-$ and $0 \le (c_f^+ - x_f^*)/(c_f^+ - c_f^-) < r^*$ or $u \in V(T^*)$ with $p_u^+ > p_u^-$ and $0 \le (y_u^* - p_u^-)/(p_u^+ - p_u^-) < r^*$. Let $Q = \{e \in E(T^*) : \frac{c_e^+ - x_e^*}{c_e^+ - c_e^-} = r^*\} \cup \{v \in V(T^*) : \frac{y_v^* - p_v^-}{p_v^+ - p_v^-} = r^*\}$. Then $Q \neq \emptyset$ since $T^* \neq \emptyset$. We can take sufficiently small $\varepsilon > 0$ such that the MMR_PCST has a solution (T^*,x',y') with $x' \in \mathbb{R}_+^{E(T^*)}$ and $y' \in \mathbb{R}_+^{V(T^*)}$ given by

$$x'_e = \begin{cases} x_e^* + \varepsilon, & \text{if } e \in Q \cap E \\ x_e^* - |Q|\varepsilon, & \text{if } e = f \\ x_e^*, & \text{otherwise} \end{cases} \quad y'_v = \begin{cases} y_v^* - \varepsilon, & \text{if } v \in Q \cap V \\ y_v^*, & \text{otherwise} \end{cases} \quad \text{or,}$$

$$x'_e = \begin{cases} x_e^* + \varepsilon, & \text{if } e \in Q \cap E \\ x_e^*, & \text{otherwise} \end{cases} \quad y'_v = \begin{cases} y_v^* - \varepsilon, & \text{if } v \in Q \cap V \\ y_v^* + |Q|\varepsilon, & \text{if } v = u \\ y_v^*, & \text{otherwise} \end{cases}$$

In either case, we have $r_m(T^*,x',y') < r^*$, which contradicts with the optimality of r^*. □

Definition 6. *For any $r \in [0,1]$, each edge $e \in E$ and each vertex $v \in V$, we define that:*

$$x_e^r = c_e^+ - r(c_e^+ - c_e^-), \text{ and } y_v^r = p_v^- + r(p_v^+ - p_v^-). \tag{4}$$

Moreover, define $T^r = T_{opt}(G, x^r, y^r) \in \mathscr{T}$ *be an optimal PCST with cost vector* x^r *and prize vector* y^r*. So we have that:*

$$\nu(T^r, x^r, y^r) = \min_{T \in \mathscr{T}} \nu(T, x^r, y^r) \text{ for any } r \in [0,1]. \tag{5}$$

Note that for any fixed $r \in [0,1]$, the PCST problem (G, x^r, y^r) can be solved by Algorithm 6, i.e., T^r can be found in $O(n)$ time.

The following lemmas are important for our polynomial algorithm of MMR_PCST problem in 2-trees, presented in next subsection.

Lemma 41. *For any* $r, s \in [0,1]$*,* $r > s$ *if and only if* $\nu(T^r, x^r, y^r) < \nu(T^s, x^s, y^s)$.

Proof. If $r > s$, then from (4), we have $x_e^r < x_e^s$ for every $e \in E$ and $y_v^r > y_v^s$ for every $v \in V$. Then $\nu(T^s, x^r, y^r) < \nu(T^s, x^s, y^s)$. From (5), we have that $\nu(T^r, x^r, y^r) \leq \nu(T^s, x^r, y^r)$. So $\nu(T^r, x^r, y^r) < \nu(T^s, x^s, y^s)$.

Now we prove that if $\nu(T^r, x^r, y^r) < \nu(T^s, x^s, y^s)$ then $r > s$. Suppose it is not true. If $r = s$ then $\nu(T^r, x^r, y^r) = \nu(T^s, x^s, y^s)$ from the definition. Otherwise $r < s$. By previous paragraph, reversing the role of r and s, we get $\nu(T^s, x^s, y^s) < \nu(T^r, x^r, y^r)$, a contradiction. So $r > s$. □

Lemma 42. *If* $r^* > 0$*, then* $\nu(T^{r^*}, x^{r^*}, y^{r^*}) = \mathsf{B}$.

Proof. Since $(T^{r^*}, x^{r^*}, y^{r^*})$ is a feasible solution of the MMR_PCST problem, $\nu(T^{r^*}, x^{r^*}, y^{r^*}) \leq \mathsf{B}$. Suppose $\nu(T^{r^*}, x^{r^*}, y^{r^*}) < \mathsf{B}$. Since $r^* > 0$, $x_e^{r^*} > c_e^-$ and $y_v^{r^*} > p_v^-$ for each $e \in E(T^{r^*})$ and each $v \in V(T^{r^*})$. Then we can increase $x_e^{r^*}$ and decrease $y_v^{r^*}$ a little bit for each $e \in E(T^{r^*})$ and each $v \in V(T^{r^*})$ such that its value is still at most B. But then the risks of each edge and vertex in T^{r^*} are decreased. So r^* is not the optimal value of the MMR_PCST problem. It is a contradiction. So $\nu(T^{r^*}, x^{r^*}, y^{r^*}) = \mathsf{B}$. □

From the above two lemmas, it is easy to get the following corollary.

Corollary 11. *If* $r^* > 0$*, then* $r = r^*$ *if and only if* $\nu(T^r, x^r, y^r) = \mathsf{B}$.

For $r^* = 0$, the following fact is trivial.

Fact 4. $r^* = 0$ *if and only if* $\nu(T^0, x^0, y^0) \leq \mathsf{B}$.

Remark 4. *If* $\nu(T^1, x^1, y^1) > \mathsf{B}$*, then we have* $\nu(T^r, x^r, y^r) > \mathsf{B}$ *for any* $r \in [0,1]$*. So the MMR_PCST problem has no feasible solution. In the following subsection, we always assume that* $\nu(T^1, x^1, y^1) \leq \mathsf{B}$*. Then the MMR_PCST problem is to find the minimum* r *such that* $\nu(T^r, x^r, y^r) \leq \mathsf{B}$.

5.3.1.1 Polynomial Algorithm for the MMR_PCST problem in 2-Trees

In this subsection, we describe a polynomial algorithm for the MMR_PCST problem on 2-trees.

Given a 2-tree $G = (V, E)$, each edge $e \in E$ is associated with a cost interval $[c_e^-, c_e^+]$, and each vertex $v \in V$ is associated with a prize interval $[p_v^-, p_v^+]$, where $0 \leq c_e^- \leq c_e^+$ and

$0 \le p_v^- \le p_v^+$. For any $r \in [0,1]$, x_e^r for each edge $e \in E$, y_v^r for each vertex $v \in V$ and T^r are defined as in Definition 6. Recall that in Section 5.2.1, we order all vertices in V as $z_1, \ldots, z_n$ such that each vertex z_i, for $1 \le i \le n-2$, has degree 2 and is contained in a triangle in the induced subgraph $G_i \equiv G[\{z_i, \ldots, z_n\}]$. Moreover, $H_i(u,v)$ is the same as defined in Definition 4 for $i = 1, \ldots, n-2$ and any pair (u,v) corresponding to an an $uv \in E(G_i)$.

From Fact 4, we can apply Algorithm 6 with input (G, x^0, y^0) to see wether $\nu(T^0, x^0, y^0) \le \mathsf{B}$ or not. If yes, then $r^* = 0$. In the following, we consider the case $\nu(T^0, x^0, y^0) > \mathsf{B}$, in which $r^* > 0$.

From Corollary 11, it is sufficient to compute the r such that $\nu(T^r, x^r, y^r) = \mathsf{B}$. For each fixed $r \in (0,1]$, $\nu(T^r, x^r, y^r)$ can be computed in linear time by Algorithm 6 with input (G, x^r, y^r). But there are infinite number of r in the $(0,1]$. It is impossible to check for each $r \in (0,1]$ wether $\nu(T^r, x^r, y^r) = \mathsf{B}$. Our idea is to compute a linear function f of r and, a lower bound $\alpha \in [0,1]$ and an upper bound $\beta \in [0,1]$ of r^*, such that $\nu(T^r, x^r, y^r) = f(r)$ for any $r \in [\alpha, \beta]$. From Corollary 11, r^* is the solution of the linear equation $f(r) = \mathsf{B}$. Then T^{r^*} can be computed by Algorithm 6 with input (G, x^{r^*}, y^{r^*}).

To compute the required linear function $\nu(T^r, x^r, y^r)$ (of r) and the bounds of r^*, we apply the same dynamic programming as in Algorithm 6 with each measure $\pi_i \in \Pi_i$ for $i = 1, \ldots, n-2$ replaced by a linear function of r, where r is contained some interval containing r^*. To be clear, denote the measure as π_i^r corresponding to each $\pi_i \in \Pi_i$ for $i = 1, \ldots, n-2$, defined in Definition 5.

Corresponding to the Lemma 36, we have the following lemma for the MMR_PCST problem.

Lemma 43. *For any $r \in [0,1]$, we have:*

$$\nu(T^r, x^r, y^r) = \min\{st_{n-2}^r(z_{n-1}, z_n), yn_{n-2}^r(z_{n-1}, z_n), ny_{n-2}^r(z_{n-1}, z_n), nn_{n-2}^r(z_{n-1}, z_n)\}.$$

Corresponding to (1), we define the initial measures, which are linear functions of r, for $i = 1$, $r \in [\alpha_1, \beta_1] \equiv [0,1]$ and each pair (u,v) corresponding to an edge $uv \in E$ as follows:

$$\begin{array}{lll} st_1^r(u,v) = x_{uv}^r - y_u^r - y_v^r; & dt_1^r(u,v) = -y_u^r - y_v^r; & yn_1^r(u,v) = -y_u^r; \\ ny_1^r(u,v) = -y_v^r; & nn_1^r(u,v) = 0 & \end{array} \tag{6}$$

Note that it is not necessary to compute the forests $T_{\pi^r(u,v)}$, because T^{r^*} will be computed by Algorithm 6 with input (G, x^{r^*}, y^{r^*}) after r^* is computed.

For $2 \le i \le n-3$ and any pair (u,v) corresponding to an edge $uv \in E(G_i)$, assume that any measure $\pi_i^r(u,v)$ for $\pi_i \in \Pi_i$ and the range of r^*, $[\alpha_i, \beta_i]$ are computed already after the elimination of $\{z_1, \ldots, z_{i-1}\}$; moreover $\pi_i^r(u,v)$ is a linear function of r for $r \in [\alpha_i, \beta_i]$. In the following, we see how to compute the measures $\pi_{i+1}^r(u,v)$, for any $\pi_{i+1} \in \Pi_{i+1}$, and the range $[\alpha_{i+1}, \beta_{i+1}]$ after the elimination of z_i in G_i, such that $\pi_{i+1}^r(u,v)$ is a linear function of r for $r \in [\alpha_{i+1}, \beta_{i+1}]$ and $r^* \in [\alpha_{i+1}, \beta_{i+1}]$.

As proved in Lemma 37-39, we get the corresponding lemma for each $\pi_{i+1} \in \Pi_{i+1}$

Lemma 44. *For any pair (u,v) corresponding to an edge $uv \in E(G_{i+1})$, given linear functions $\pi_i^r(u,v)$ of r in $[\alpha_i, \beta_i]$, for any $\pi_i \in \Pi_i$ and $r^* \in [\alpha_i, \beta_i]$, each $\pi_{i+1}^r(u,v)$ can be computed as follows:*

- *If u,v are not the two neighbors of z_i in G_i, $\pi^r_{i+1}(u,v) = \pi^r_i(u,v)$, for each $\pi_{i+1} \in \Pi_{i+1}$ and $\pi_i \in \Pi_i$;*

- *Otherwise, we have:*

  - $st^r_{i+1}(u,v) = \min\{st^r_i(u,v) + yn^r_i(u,z_i) + ny^r_i(z_i,v) + y^r_u + y^r_v,$
    $st^r_i(u,v) + st^r_i(u,z_i) + dt^r_i(z_i,v) + y^r_u + y^r_{z_i} + y^r_v,$
    $st^r_i(u,v) + dt^r_i(u,z_i) + st^r_i(z_i,v) + y^r_u + y^r_{z_i} + y^r_v,$
    $dt^r_i(u,v) + st^r_i(u,z_i) + st^r_i(z_i,v) + y^r_u + y^r_{z_i} + y^r_v\};$
  - $dt^r_{i+1}(u,v) = \min\{dt^r_i(u,v) + yn^r_i(u,z_i) + ny^r_i(z_i,v) + y^r_u + y^r_v,$
    $dt^r_i(u,v) + st^r_i(u,z_i) + dt^r_i(z_i,v) + y^r_u + y^r_{z_i} + y^r_v,$
    $dt^r_i(u,v) + dt^r_i(u,z_i) + st^r_i(z_i,v) + y^r_u + y^r_{z_i} + y^r_v,\},$
  - $yn^r_{i+1}(u,v) = \min\{yn^r_i(u,v) + yn^r_i(u,z_i) + y^r_u,$
    $yn^r_i(u,v) + st^r_i(u,z_i) + yn^r_i(z_i,v) + y^r_u + y^r_{z_i}\};$
  - $ny^r_{i+1}(u,v) = \min\{ny^r_i(u,v) + ny^r_i(z_i,v) + y^r_v,$
    $ny^r_i(u,v) + ny^r_i(u,z_i) + st^r_i(z_i,v) + y^r_{z_i} + y^r_v\};$
  - $nn^r_{i+1}(u,v) = \min\{nn^r_i(u,v),\ nn^r_i(u,z_i),\ nn^r_i(z_i,v),\ ny^r_i(u,z_i) + yn^r_i(z_i,v) + y^r_{z_i}\}.$

From Lemma 44, for $\pi_{i+1} \in \Pi_{i+1}$, we see that if u,v are not the two neighbors of z_i in G_i, then $\pi^r_{i+1}(u,v) = \pi^r_i(u,v)$ is a linear in function in $[\alpha_i,\beta_i]$. Otherwise, $\pi^r_{i+1}(u,v)$ is the minimum of at most four linear functions of r in $[\alpha_i,\beta_i]$. So $\pi^r_{i+1}(u,v)$ is a piecewise linear function of r in $[\alpha_i,\beta_i]$ with at most three non-differentiable points. So there are at most 15 non-differentiable points in all the five measures. In the following lemma, we compute an interval containing r^*, $[\alpha_{i+1},\beta_{i+1}] \subseteq [\alpha_i,\beta_i]$, in which each $\pi^r_{i+1}(u,v)$ is linear.

Lemma 45. *For any pair (u,v) corresponding to an edge $uv \in E(G_{i+1})$, let each $\pi^r_{i+1}(u,v)$, for any $\pi_{i+1} \in \Pi_{i+1}$, be a piecewise linear function of r in $[\alpha_i,\beta_i]$ with at most three non-differentiable points. Moreover $r^* \in [\alpha_i,\beta_i]$. Let $A = \{\alpha_i,\beta_i\} \cup \{$ the non-differentiable points of all $\pi^r_{i+1}(u,v)$, for any $\pi_{i+1} \in \Pi_{i+1}\}$.*

Let $\alpha_{i+1} = \max_{r\in A}\{r : \nu(T^r,x^r,y^r) \geq \mathsf{B}\}$ and $\beta_{i+1} = \min_{r\in A}\{r : \nu(T^r,x^r,y^r) \leq \mathsf{B}\}$. Then $\pi^r_{i+1}(u,v)$, for any $\pi_{i+1} \in \Pi_{i+1}\}$, is a linear function of r in $[\alpha_{i+1},\beta_{i+1}]$ and $r^* \in [\alpha_{i+1},\beta_{i+1}]$.

Proof. From Lemma 42, $\nu(T^{r^*},x^{r^*},y^{r^*}) = \mathsf{B}$. So $\nu(T^{\alpha_{i+1}},x^{\alpha_{i+1}},y^{\alpha_{i+1}}) \geq \nu(T^{r^*},x^{r^*},y^{r^*}) \geq \nu(T^{\beta_{i+1}},x^{\beta_{i+1}},y^{\beta_{i+1}})$. From Lemma 41, $\alpha_{i+1} \leq r^* \leq \beta_{i+1}$, i.e. $r^* \in [\alpha_{i+1},\beta_{i+1}]$.

To prove that $\pi^r_{i+1}(u,v)$, for any $\pi_{i+1} \in \Pi_{i+1}$, is a linear function of r in $[\alpha_{i+1},\beta_{i+1}]$, it is sufficient to prove that there is no non-differentiable points in $(\alpha_{i+1},\beta_{i+1})$ of any $\pi^r_{i+1}(u,v)$. Suppose there is a non-differentiable point $r \in (\alpha_{i+1},\beta_{i+1})$ of some $\pi^r_{i+1}(u,v)$. If $\nu(T^r,x^r,y^r) \leq \mathsf{B} \leq \nu(T^{\alpha_{i+1}},x^{\alpha_{i+1}},y^{\alpha_{i+1}})$, then $r \geq \beta_{i+1}$ from Lemma 41; otherwise, we have $r < \alpha_{i+1}$. It is a contradiction. □

Remark 5. *If $\alpha_{i+1} = \beta_{i+1}$, then $r^* = \alpha_{i+1} = \beta_{i+1}$. Then it is not necessary to compute the measures π^r_j, for any $\pi_j \in \Pi_j$ for $i+2 \leq j \leq n-2$.*

Theorem 16. *The MMR_PCST problem in 2-trees of n vertices can be solved in $O(n^2)$ time.*

Proof. From Lemma 44 and 45, for $1 \leq i \leq n-2$ and any pair (u,v) corresponding to an edge $uv \in E(G_i)$, we can compute the measures $\pi_i^r(u,v)$, for any $\pi_i \in \Pi_i$, and interval $[\alpha_i,\beta_i]$ such that $\pi_i^r(u,v)$ is a linear function of r in $[\alpha_i,\beta_i]$ and $r^* \in [\alpha_i,\beta_i]$.

For each $i = 1,\ldots,n-2$, to compute the measures as in Lemma 44, we need to compute the minimum of at most four linear functions. In [Meg79], there is an $O(k\log k)$ algorithm for computing the minimum of k linear functions. So each $\pi_i^r(u,v)$ can be computed in constant time. To compute $[\alpha_i,\beta_i]$ as in Lemma 45, we need to compute $\nu(T^r,x^r,y^r)$ for r belongs to all the non-differentiable points, at most 15 non-differentiable points, of all measures $\pi_i^r(u,v)$, for any $\pi_i \in \Pi_i$, in $[\alpha_{i-1},\beta_{i-1}]$. From Theorem 15, this can be done in $O(n)$ time. So it costs $O(n^2)$ time to compute all the measures $\pi_i^r(u,v)$ and interval $[\alpha_i,\beta_i]$ for $1 \leq i \leq n-2$, any $\pi_i \in \Pi_i$, and any pair (u,v) corresponding to an edge $uv \in E(G_i)$.

From Lemma 43, to compute $\nu(T^r,x^r,y^r)$, we only need to compute the minimum of the four linear functions of r in $[\alpha_{n-2},\beta_{n-2}]$, $st_{n-2}^r(z_{n-1},z_n)$, $yn_{n-2}^r(z_{n-1},z_n)$, $ny_{n-2}^r(z_{n-1},z_n)$ and $nn_{n-2}^r(z_{n-1},z_n)$, which takes constant time. So $\nu(T^r,x^r,y^r)$ is a piecewise linear functions of r in $[\alpha_{n-2},\beta_{n-2}]$ with at most three non-differentiable points. Then as in Lemma 45, we can compute an interval $[\alpha,\beta] \subseteq [\alpha_{n-2},\beta_{n-2}]$ such that $\nu(T^r,x^r,y^r)$ is linear in $[\alpha,\beta]$ and $r^* \in [\alpha,\beta]$. It take $O(n)$ time.

Finally, we solve the linear equation $\nu(T^r,x^r,y^r) = \mathsf{B}$ for $r \in [\alpha,\beta]$. Its solution is the optimal value r^* of the MMR_PCST problem. Then T^{r^*} is an optimal solution of the MMR_PCST problem, which can be computed by Algorithm 6 in $O(n)$ time.

So the MMR_PCST problem in 2-trees of n vertices can be solved in $O(n^2)$ time □

5.3.2 PCST Problem under Min-Sum Risk Model

The PCST problem under min-sum risk model, denoted by MSR_PCST, is to find a tree T in given graph $G = (V,E)$ along with payment $x \in \mathbb{R}_+^{E(T)}$ and prize $y \in \mathbb{R}_+^{V(T)}$ such that the sum of risks at edges and vertices in T:

$$r_s(T,x,y) \equiv \sum_{e\in E(T)} \frac{c_e^+ - x_e}{c_e^+ - c_e^-} + \sum_{v\in V(T)} \frac{y_v - p_v^-}{p_v^+ - p_v^-} = \sum_{e\in E(T)} r(x_e) + \sum_{v\in V(T)} r(y_v)$$

is minimized and the value $\nu(T,x,y)$ is no greater than the given budget bound B. This problem can be formulated as follows:

$$\begin{aligned} \text{(MSR\_PCST)} \quad \min_{T\in\mathscr{T}} \Big(& \sum_{e\in E(T)} \frac{c_e^+ - x_e}{c_e^+ - c_e^-} + \sum_{v\in V(T)} \frac{y_v - p_v^-}{p_v^+ - p_v^-} \Big) \\ \text{s.t.} \quad & \nu(T,x,y) \leq \mathsf{B}; \\ & x_e \in [c_e^-, c_e^+],\ \forall\, e \in E(T); \\ & y_v \in [p_v^-, p_v^+],\ \forall\, v \in V(T). \end{aligned}$$

For example, given a graph G with cost intervals and prize interval as shown in Fig. 5.1 (Robust PCST), let the budget B is -12. The optimal solution of the MMR_PCST problem (T^*,x^*,y^*) is also an optimal solution of the MSR_PCST problem, where T^* is the subtree induced by $\{s,w,z,t\}$, $x^* = (7,8,2)$ is the cost vector of the edges (sw,wz,zt) on T^* and $y^* = (10,6,3,10)$ is the prize vector of the vertices (s,w,z,t) on T^*. The optimal value

$r_s(T^*,x^*,y^*) = 2.5$. In addition, the solution (T^*,x,y) for $x = (8,9,1)$ and $y = (10,7,3,10)$ is also an optimal solution of the MSR_PCST problem.

Now let us see an easy lemma satisfied by any optimal solution of the MSR_PCST problem, which is similar to the property of Lemma 42 for the MRR_PCST problem

Lemma 46. *Let (T^*,x^*,y^*) be an optimal solution of the MSR_PCST problem. If $r_s(T^*,x^*,y^*) > 0$, then $\nu(T^*,x^*,y^*) = \mathsf{B}$.*

Proof. Since $r_s(T^*,x^*,y^*) > 0$, there exist an edge or a vertex with risk bigger than 0. W.l.o.g, suppose $u \in V(T)$ has risk $r(y_u^*) > 0$, i.e. $y_u^* \in (p_u^-, p_u^+)$. If $\nu(T^*,x^*,y^*) < \mathsf{B}$, then let $\delta = \min\{p_u^+ - y_u^*, \mathsf{B} - \nu(T^*,x^*,y^*)\}$. Let change y_u^* to be $y_u^* + \delta$. Then (T^*,x^*,y^*) is still a feasible solution, but $r_s(T^*,x^*,y^*)0$ strictly decreases. It is a contradiction. So $\nu(T^*,x^*,y^*) = \mathsf{B}$. □

The MSR_PCST problem has an optimal solution that satisfies the following extremeness property – the edge payments and vertex prizes hit lower or upper limits with at most one exceptional edge and at most one exceptional vertex. This property is contrary to the property in Lemma 40 for the MRR_PCST, where all price and cost are fixed when r is fixed.

Lemma 47 (Extremeness property)**.** *There exists an optimal solution (T^*,x^*,y^*) of the MSR_PCST problem which contains an edge $f \in E(T^*)$ if $E(T^*) \neq \emptyset$ and a vertex $u \in V(T^*)$ such that*

$$\begin{aligned} &x_e^* \in \{c_e^-, c_e^+\} \text{ for all } e \in E(T^*) \setminus \{f\} \text{ if } E(T^*) \neq \emptyset; \text{ and} \\ &y_v^* \in \{P_v^-, p_v^+\} \text{ for all } v \in V(T^*) \setminus \{u\}. \end{aligned} \tag{7}$$

Proof. Let (T^*,x^*,y^*) be an optimal solution to the MSR_PCST problem such that the union of two sets $\mathcal{E}(T^*,x^*,y^*) \equiv \{e : e \in E(T^*), x_e^* \in (c_e^-, c_e^+)\}$ and $\mathcal{V}(T^*,x^*,y^*) \equiv \{v : v \in V(T^*), y_v^* \in (p_v^-, p_v^+)\}$ contains as few elements as possible. If $|\mathcal{E}(T^*,x^*,y^*)| \leq 1$ and $|\mathcal{V}(T^*,x^*,y^*)| \leq 1$, then the lemma is true.

If $|\mathcal{E}(T^*,x^*,y^*)| > 1$, then there exist distinct edges $g, f \in \mathcal{E}(T^*,x^*,y^*)$ with $c_g^+ - c_g^- \leq c_f^+ - c_f^-$. Take $\delta = \min\{c_g^+ - x_g^*, x_f^* - c_f^-\}$ and define $x' \in \mathbb{R}_+^{E(T^*)}$ by setting $x'_g = x_g^* + \delta$, $x'_f = x_f^* - \delta$ and $x'_e = x_e^*$ for every $e \in E(T^*) \setminus \{g,f\}$. Then $\nu(T^*,x',y^*) = \nu(T^*,x^*,y^*) \leq \mathsf{B}$ and $r_s(T^*,x',y^*) \leq r_s(T^*,x^*,y^*)$. So (T^*,x',y^*) is also an optimal solution to the MSR_PCST with $\mathcal{E}(T^*,x',y^*) \subset \mathcal{E}(T^*,x^*,y^*)$ and $\mathcal{V}(T^*,x',y^*) = \mathcal{V}(T^*,x^*,y^*)$. It is a contradiction.

If $|\mathcal{V}(T^*,x^*,y^*)| > 1$, then there exist distinct vertices $u,z \in \mathcal{V}(T^*,x^*,y^*)$ with $p_u^+ - p_u^- \leq p_z^+ - p_z^-$. Take $\delta = \min\{y_u^* - p_u^-, p_v^+ - y_v^*\}$ and define $y' \in \mathbb{R}_+^{V(T^*)}$ by setting $y'_u = y_u^* - \delta$, $y'_z = y_z^* + \delta$ and $y'_v = y_v^*$ for every $v \in V(T^*) \setminus \{u,z\}$. Then $\nu(T^*,x^*,y') = \nu(T^*,x^*,y^*) \leq \mathsf{B}$ and $r_s(T^*,x^*,y') \leq r_s(T^*,x^*,y^*)$. So (T^*,x^*,y') is also an optimal solution to the MSR_PCST with $\mathcal{V}(T^*,x^*,y') \subset \mathcal{V}(T^*,x^*,y^*)$ and $\mathcal{E}(T^*,x^*,y') = \mathcal{E}(T^*,x^*,y^*)$. It is a contradiction. □

For easy description, we give the following definition.

Definition 7. *An optimal solution (T^*,x^*,y^*) of the MSR_PCST problem is called an extreme optimal solution if there are an edge $f \in E(T^*)$ unless $E(T^*) = \emptyset$ and a vertex*

$u \in V(T^*)$ such that (T^*,x^*,y^*) satisfies (7); moreover, f (resp. u) is called the extreme edge (resp. extreme vertex) of (T^*,x^*,y^*).

In an extreme optimal solution (T^*,x^*,y^*) with an extreme edge f and an extreme vertex u, each edge e (resp. vertex v) in T^* except f (resp. u) has two possible costs, either c_e^+ (resp. p_v^+) or c_e^- (resp. p_v^-), so it has risk either 0 (resp. 1) or 1 (resp. 0). Intuitively, if we change the cost of f to be c_f^+ and the prize of u to be p_u^- in (T^*,x^*,y^*), then its value should be the minimum among all solutions (T,x,y), where T containing f,u, $x_f = c_f^+$, $y_u = p_u^-$, all other edges and vertices have risk either 0s or 1s, and $r_s(T,x,y) \leq r_s(T^*,x^*,y^*) - r(x_f^*) - r(y_u^*)$. In the following, we see how to get an extreme optimal solution of the MSR_PCST problem. It is divided into two problems $(BWCC)$ and (LP) as presented in the following lemma:

Lemma 48. *Let (T^*,x^*,y^*) be an extreme optimal solution with extreme edge $f \in E(T^*)$ and extreme vertex $u \in V(T^*)$ of the MSR_PCST problem. Assume that $r(T^*,x^*,y^*) > 0$. Let (T,x,y) be an optimal solution of the following problem $(BWCC)$:*

$$
\begin{aligned}
(BWCC) \qquad & \min \nu(T,x,y) \\
s.t. \quad & r_s(T,x,y) \leq r_s(T^*,x^*,y^*) - r(x_f^*) - r(y_u^*); \\
& f \in E(T), u \in V(T), x_f = c_f^+, y_u = p_u^-; \\
& x_e \in \{c_e^+, c_e^-\},\ \forall\, e \in E(T) \setminus \{f\}; \\
& y_v \in \{p_v^+, p_v^-\},\ \forall\, v \in V(T) \setminus \{u\}.
\end{aligned}
$$

Let (x_0,y_0) be an optimal solution of the following linear programming (LP):

$$
\begin{aligned}
(LP) \qquad & \min \frac{c_f^+ - x_0}{c_f^+ - c_f^-} + \frac{y_0 - p_u^-}{p_u^+ - p_u^-} \\
s.t. \quad & \nu(T,x,y) - x_f + y_u + x_0 - y_0 \leq \mathsf{B}; \\
& x_0 \in [c_f^-, c_f^+], \\
& y_0 \in [p_u^-, p_u^+].
\end{aligned}
$$

Then (T,x',y') is also an extreme optimal solution with extreme edge $f \in E(T)$ and extreme vertex $u \in V(T)$ of the MSR_PCST problem, where $x'_f = x_0, y'_u = y_0$, $x'_e = x_e$ for any edge $e \in E(T) \setminus \{f\}$ and $y'_v = y_v$ for any $v \in V(T) \setminus \{u\}$.

Proof. It is sufficient to prove that $\nu(T,x',y') \leq \mathsf{B}$ and $r_s(T,x',y') \leq r_s(T^*,x^*,y^*)$, since $x'_e = x_e \in \{c_e^-, c_e^+\}$ for any $e \in E(T) \setminus \{f\}$ and $y'_v = y_v \in [p_v^-, p_v^+]$ for any $v \in V(T) \setminus \{u\}$ and $f \in E(T), u \in V(T)$ from $(BWCC)$ and (LP). Since $\nu(T,x',y') = \nu(T,x,y) - x_f + y_u + x_0 - y_0$, then $\nu(T,x',y') \leq \mathsf{B}$ from (LP). In the following, we show that $r_s(T,x',y') \leq r_s(T^*,x^*,y^*)$.

One see that $r_s(T,x',y') = r_s(T,x,y) - r(x_f) - r(y_u) + r(x'_f) + r(y'_u)$. Since $x_f = c_f^+$ and $y_u = p_u^-$, $r(x_f) = r(y_u) = 0$. So $r_s(T,x',y') = r_s(T,x,y) + r(x_0) + r(y_0)$. From the first condition of $(BWCC)$, $r_s(T,x',y') \leq r_s(T^*,x^*,y^*) - r(x_f^*) - r(y_u^*) + r(x_0) + r(y_0)$. If $r(x_0) + r(y_0) \leq r(x_f^*) + r(y_u^*)$, then the lemma is proved.

To prove $r(x_0) + r(y_0) \leq r(x_f^*) + r(y_u^*)$, it is sufficient to prove that (x_f^*, y_u^*) is a feasible solution of (LP). So we only need to prove that $\nu(T,x,y) - x_f + y_u + x_f^* - y_u^* \leq \mathsf{B}$. From

Lemma 46, it is enough to prove that $\nu(T,x,y) - x_f + y_u + x^*_f - y^*_u \leq \nu(T^*,x^*,y^*)$, i.e. $\nu(T,x,y) - x_f + y_u \leq \nu(T^*,x^*,y^*) - x^*_f + y^*_u$. Suppose it is not true, i.e., $\nu(T,x,y) - x_f + y_u > \nu(T^*,x^*,y^*) - x^*_f + y^*_u$. Then let $\bar{x}_f = c^+_f$, $\bar{y}_u = p^-_u$, $\bar{x}_e = x^*_e$ for any edge $e \in E(T^*) \setminus \{f\}$ and $\bar{y}_v = y^*_v$ for any $v \in V(T^*) \setminus \{u\}$. Then $\nu(T^*,\bar{x},\bar{y}) = \nu(T^*,x^*,y^*) - x^*_f + y^*_u + c^+_f - p^-_u < \nu(T,x,y) - x_f + y_u + c^+_f - p^-_u = \nu(T,x,y)$, i.e. $\nu(T^*,\bar{x},\bar{y}) < \nu(T,x,y)$. If $(T^*,\bar{x},\bar{y})$ is a feasible solution of $(BWCC)$, then it is a contradiction. So the lemma is proved.

Finally, we only need to prove that $(T^*,\bar{x},\bar{y})$ is a feasible solution of $(BWCC)$. Since $\bar{x}_f = c^+_f$, $\bar{y}_u = p^-_u$, $\bar{x}_e = x^*_e$ for any edge $e \in E(T^*) \setminus \{f\}$ and $\bar{y}_v = y^*_v$ for any $v \in V(T^*) \setminus \{u\}$, we only need to prove $r_s(T^*,\bar{x},\bar{y}) \leq r_s(T^*,x^*,y^*) - r(x^*_f) - r(y^*_u)$. We see that $r_s(T^*,\bar{x},\bar{y}) = r_s(T^*,x^*,y^*) - r(x^*_f) - r(y^*_u) + r(\bar{x}_f) + r(\bar{y}_u)$. Since $\bar{x}_f = c^+_f$ and $\bar{y}_u = p^-_u$, $r(\bar{x}_f) = r(\bar{y}_u) = 0$. So $r_s(T^*,\bar{x},\bar{y}) = r_s(T^*,x^*,y^*) - r(x^*_f) - r(y^*_u)$. The lemma is proved. □

Note that the optimal solution (T,x,y) of $(BWCC)$ has the minimum value of any tree containing f,u with both cost risk of f and prize risk u being 0 and any other vertices and edges on the tree with risks either 1 or 0; and moreover, the sum of all the risks of edges and vertices on the tree is at most $r^* - r(x^*_f) - r(y^*_u)$. This lemma plays an important role in our algorithm for the MSR_PCST problem in 2-trees, which is going to be presented in the rest of this subsection.

5.3.2.1 Polynomial Algorithm for the MSR_PCST problem in 2-Trees

In this subsection, we present the polynomial algorithm for the MSR_PCST problem in 2-Trees.

Given a 2-tree $G = (V,E)$, each edge $e \in E$ is associated with a cost interval $[c^-_e, c^+_e]$, and each vertex $v \in V$ is associated with a prize interval $[p^-_v, p^+_v]$, where $0 \leq c^-_e \leq c^+_e$ and $0 \leq p^-_v \leq p^+_v$. Let $|V| = n$. We denote r^* as the optimal value of the MSR_PCST problem $(G, c^+, c^-, p^+, p^-, \mathsf{B})$.

It is easy to check whether $r^* = 0$. We only need to apply the Algorithm 6 with input (G,x^0,y^0) to see whether $\nu(T^0,x^0,y^0) \leq \mathsf{B}$. If yes, then the $r^* = 0$ and $\nu(T^0,x^0,y^0)$ is an optimal solution. Otherwise, $r^* > 0$. In the following, we always assume $\nu(T^0,x^0,y^0) > \mathsf{B}$, i.e., $r^* > 0$.

We are going to find an extreme optimal solution. Assume that f,u are the extreme edge and extreme vertex of some extreme optimal solution (T^*,x^*,y^*), and moreover the risk $\Gamma \equiv \mathbf{r_s}(\mathbf{T}^*,\mathbf{x}^*,\mathbf{y}^*) - \mathbf{r}(\mathbf{x}^*_\mathbf{f}) - \mathbf{r}(\mathbf{y}^*_\mathbf{u})$ is known. Then from Lemma 48, we just need to solve the two problems $(BWCC)$ and (LP). After solving $(BWCC)$, we see that (LP) is just a linear programming with only 2 variables and 5 constraints, which can be solved in constant time. So the key point is to compute Γ and solve the problem $(BWCC)$. The main idea of our algorithm is to compute a bound of the risk Γ. Note that Γ is an integer. From the computed bound of Γ, which is going to be presented later, Γ can be three possible integers. On the other hand, since it is not sure which edge and vertex can be the extreme edge f and extreme vertex u, we are going to consider each pair of edge and vertex in G as f and u respectively in the $(BWCC)$ problem. Finally, we solve the $(BWCC)$ and (LP) problems with three possibilities of Γ and $O(n^2)$ possibilities of f,u. Then we choose a solution with the minimum sum of all risks among all the $O(n^2)$ solutions we get as in

Lemma 48. In the following, we see how to compute the bound for Γ and how to solve the problem $(BWCC)$. It turns out that both of these two problems are closed related with the *Binary Weight Constrained Prize Collecting Steiner Tree (BWC_PCST)* problem (see the definition in the next paragraph).

Bound of $\Gamma = r_s(T^*,x^*,y^*) - r(x_f^*) - r(y_u^*)$

First let us compute the bound for Γ by solving the *Binary Weight Constrained Prize Collecting Steiner Tree (BWC_PCST)* problem, defined as follows for any non-negative integer ζ:

$$\begin{aligned} (BWC\_PCST) \qquad & \min_{T\in\mathscr{T}} \nu(T,x,y) \\ \text{s.t.} \quad & r_s(T,x,y) \le \zeta; \\ & x_e \in \{c_e^-, c_e^+\},\ \forall\, e \in E(T); \\ & y_v \in \{p_v^-, p_v^+\},\ \forall\, v \in V(T). \end{aligned}$$

For easy description, we denote the optimal value as ν_ζ. One sees that in the (BWC_PCST) problem, the costs and prizes of any edges and vertices are either the upper bounds or lower bounds of the given intervals. So their risks are either 1s or 0s. Note that $r_s(T,x,y) \le 2n-1$ for any (T,x,y). So we always assume that $0 \le \zeta \le 2n-1$.

Lemma 49. *Let β be the minimum non-negative integer such that $\nu_\beta \le \mathsf{B}$. Then $\beta - 2 \le \Gamma \le \beta$, i.e., $\Gamma \in \{\beta-2, \beta-1, \beta\}$.*

Proof. Recall that $\Gamma = r_s(T^*,x^*,y^*) - r(x_f^*) - r(y_u^*)$ for an extreme optimal solution (T^*,x^*,y^*) of the MSR_PCST problem with extreme edge $f \in E(T^*)$ and extreme vertex $u \in V(T^*)$. If $\beta = 0$, then $\nu_0 \le \mathsf{B}$. So $r^* = 0$. Then any optimal solution of the MSR_PCST problem has risk 0 of each edge and vertex. So $\Gamma = 0$. The lemma is proved.

Otherwise, $\beta \ge 1$ and $\nu_{\beta-1} > \mathsf{B}$. Since $\nu_\beta \le \mathsf{B}$, the optimal solution (T,x,y) of the (BWC_PCST) problem for $\zeta = \beta$ is a feasible solution of the MSR_PCST problem. Then $r_s(T^*,x^*,y^*) \le \beta$, since $r_s(T,x,y) \le \beta$ from the first condition of (BWC_PCST).

If $r^* = 0$, then $\beta = 0$. So we have $r^* > 0$. From Lemma 46, $\nu(T^*,x^*,y^*) = \mathsf{B}$. Since $\nu_{\beta-1} > \mathsf{B}$, then $\nu_{\beta-1} > \nu(T^*,x^*,y^*)$. So (T^*,x^*,y^*) is not a feasible solution of the (BWC_PCST) problem for $\zeta = \beta - 1$, i.e., $r_s(T^*,x^*,y^*) > \beta - 1$. So $\Gamma = r_s(T^*,x^*,y^*) - r(x_f^*) - r(y_u^*) \ge r_s(T^*,x^*,y^*) - 2 > \beta - 3$. □

From Lemma 49, the desired bound of Γ can be obtained if we can solve the (BWC_PCST) problem for any integer $\zeta \in [0, 2n-1]$.

Solving the (BWC_PCST) Problem for $\zeta \in [0, 2n-1]$

To solve the (BWC_PCST) problem, we are going to follow the idea of Algorithm 6 and add the risk into the measures. Recall that in Section 5.2.1, we order all vertices in V as $z_1, \ldots, z_n$ such that each vertex z_i, for $1 \le i \le n-2$, has degree 2 and is contained in a triangle in the induced subgraph $G_i \equiv G[\{z_i, \ldots, z_n\}]$. Moreover, $H_i(u,v)$ is the same as defined in Definition 4 for $i = 1, \ldots, n-2$ and any pair (u,v) corresponding to an $uv \in E(G_i)$. In the following, we give the definitions of the measures with risk.

Definition 8. *For each $1 \le i \le n-2$, for $\tau, \theta \in \{+,-\}$, for each pair (u^τ, v^θ) corresponding to an edge $uv \in G_i$ and each integer $\xi \in [0, \zeta]$, we will associate five measures defined as follows, which summarize the values and sum-risks[6] incurred in the subgraph $H_i(u,v)$; moreover, u (resp. v) has prize p_u^τ (resp. p_v^θ).*

(i) $st_i(u^\tau, v^\theta, \xi)$ is the minimum value of a tree in $H_i(u,v)$ containing both u and v with sum-risks at most ξ; $(T^{st_i(u^\tau,v^\theta,\xi)}, x^{st_i(u^\tau,v^\theta,\xi)}, y^{st_i(u^\tau,v^\theta,\xi)})$ denotes such a solution;

(ii) $dt_i(u^\tau, v^\theta, \xi)$ is the minimum value of the forest consisting of two vertex-disjoint trees in $H_i(u,v)$, one containing u and the other containing v, with sum-risks at most ξ; $(T^{dt_i(u^\tau,v^\theta,\xi)}, x^{dt_i(u^\tau,v^\theta,\xi)}, y^{dt_i(u^\tau,v^\theta,\xi)})$ denotes such a solution;

(iii) $yn_i(u^\tau, v^\theta, \xi)$ is the minimum value of a tree in $H_i(u,v)$ containing u but not v with sum-risks at most ξ; $(T^{yn_i(u^\tau,v^\theta,\xi)}, x^{yn_i(u^\tau,v^\theta,\xi)}, y^{yn_i(u^\tau,v^\theta,\xi)})$ denotes such a solution;

(iv) $ny_i(u^\tau, v^\theta, \xi)$ is the minimum value of a tree in $H_i(u,v)$ containing v but not u with sum-risks at most ξ; $(T^{ny_i(u^\tau,v^\theta,\xi)}, x^{ny_i(u^\tau,v^\theta,\xi)}, y^{ny_i(u^\tau,v^\theta,\xi)})$ denotes such a solution;

(v) $nn_i(u^\tau, v^\theta, \xi)$ is the minimum value of a tree in $H_i(u,v)$ containing neither u nor v with sum-risks at most ξ; $(T^{nn_i(u^\tau,v^\theta,\xi)}, x^{nn_i(u^\tau,v^\theta,\xi)}, y^{nn_i(u^\tau,v^\theta,\xi)})$ denotes such a solution.

We can always omit the subscript τ (resp. θ) if u (resp. v) is not contained in the forest. So we also use $yn_i(u^\tau, v, \xi)$, $ny_i(u, v^\theta, \xi)$ and $nn_i(u, v, \xi)$

Similarly to Lemma 36, we have the following lemma for the (BWC_PCST) problem.

Lemma 50. *For any $\zeta \in [0, 2n-1]$, we have:*

$$\nu_\zeta = \min\{st_{n-2}(z_{n-1}^\tau, z_n^\theta, \zeta), yn_{n-2}(z_{n-1}^\tau, z_n, \zeta), ny_{n-2}(z_{n-1}, z_n^\theta, \zeta), nn_{n-2}(z_{n-1}, z_n, \zeta) : \tau, \theta \in \{+,-\}\}.$$

Moreover, the corresponding solution achieving the value ν_ζ is an optimal solution of the (BWC_PCST) problem.

First let us define the initial measures for $i = 1$, $\xi = 0, \ldots, \zeta$ and any pair (u,v) corresponding to an edge $uv \in G$.

$$\begin{array}{ll}
st_1(u^-, v^-, 0) = c_{uv}^+ - p_u^- - p_v^-, & T_{st_1(u^-,v^-,0)} = (\{u,v\}, uv); \\
dt_1(u^-, v^-, 0) = -p_u^- - p_v^-, & T_{dt_1(u^-,v^-,0)} = (\{u,v\}, \emptyset); \\
yn_1(u^-, v, 0) = -p_u^-, & T_{yn_1(u^-,v,0)} = (\{u\}, \emptyset); \\
ny_1(u, v^-, 0) = -p_v^-, & T_{ny_1(u,v^-,0)} = (\{v\}, \emptyset); \\
nn_1(u, v, 0) = 0, & T_{nn_1(u,v,0)} = (\emptyset, \emptyset).
\end{array} \tag{8}$$

The cost vectors and prize vectors are not presented explicitly, but one can see it clearly from the formula of the corresponding measures. For example, from $st_1(u^-, v^-, 0) = c_{uv}^+ - p_u^- - p_v^-$, we know that $x_{uv}^{st_1(u^-,v^-,0)} = c_{uv}^+$, $y_u^{st_1(u^-,v^-,0)} = p_u^-$ and $y_v^{st_1(u^-,v^-,0)} = p_v^-$.

If $r(y_u) + r(y_v) > \xi$, then set $st_1(u^\tau, v^\theta, \xi) = dt_1(u^\tau, v^\theta, \xi) = \infty$. For example, $st_1(u^+, v^-, 0) = \infty$ since $r(y_u) = 1 > 0$.

If $r(y_u) > \xi$ (resp. $r(y_v) > \xi$), then set $yn_1(u^\tau, v, \xi) = \infty$ (resp. $ny_1(u, v^\theta, \xi) = \infty$).

[6] The sum of all risks of edges and vertices.

So any other measure for $\xi = 0$ and $i = 1$ except the ones in (8) has value ∞.

For any $\xi \geq 1$, $T_{\pi_1(u^\tau, v^\theta, \xi)} = T_{\pi_1(u^\tau, v^\theta, 0)}$ for any $\pi_1 \in \Pi_1$. So in the following, we only present the measures $\pi_1(u, v, \xi)$.

$$
\begin{aligned}
&st_1(u^-, v^-, 1) = c^-_{uv} - p^-_u - p^-_v,\\
&st_1(u^+, v^-, 1) = c^+_{uv} - p^+_u - p^-_v,\\
&st_1(u^-, v^+, 1) = c^+_{uv} - p^-_u - p^+_v;\\
&st_1(u^+, v^+, 2) = c^-_{uv} - p^+_u - p^+_v,\\
&st_1(u^\tau, v^\theta, \xi) = c^-_{uv} - p^\tau_u - p^\theta_v, \text{ for any } 2 \leq \xi \leq \zeta \text{ and } r(y_u) + r(y_v) < 2;\\
&st_1(u^\tau, v^\theta, \xi) = c^-_{uv} - p^\tau_u - p^\theta_v, \text{ for any } 3 \leq \xi \leq \zeta;\\
&dt_1(u^\tau, v^\theta, 1) = -p^\tau_u - p^\theta_v, \text{ for any } (\tau, \theta) \neq (+, +)\\
&dt_1(u^\tau, v^\theta, \xi) = -p^\tau_u - p^\theta_v, \text{ for any } 2 \leq \xi \leq \zeta;\\
&yn_1(u^\tau, v, \xi) = -p^\tau_u, \text{ for any } 1 \leq \xi \leq \zeta;\\
&ny_1(u, v^\theta, \xi) = -p^\theta_v, \text{ for any } 1 \leq \xi \leq \zeta;\\
&nn_1(u, v, \xi) = 0, \text{ for any } 1 \leq \xi \leq \zeta.
\end{aligned}
$$

Any other measure for $\xi \geq 1$ and $i = 1$ has value ∞.

We update the measures recursively. Assume that, for $2 \leq i \leq n-3$, and any pair (u, v) corresponding to an edge $uv \in E(G_i)$, any measure $\pi_i(u^\tau, v^\theta, \xi)$ for $\pi_i \in \Pi_i$, $\xi = 0, \ldots, \zeta$ and $\tau, \theta \in \{+, -\}$ are computed already after the elimination of $\{z_1, \ldots, z_{i-1}\}$. In the following, we see how to compute the measures $\pi_{i+1}(u^\tau, v^\theta, \xi)$ for $\pi_i \in \Pi_i$, $\xi = 0, \ldots, \zeta$ and $\tau, \theta \in \{+, -\}$.

As proved in Lemma 37-39, we get the corresponding lemma:

Lemma 51. *For $2 \leq i \leq n-3$, and any pair (u, v) corresponding to an edge $uv \in E(G_{i+1})$, any $0 \leq \xi \leq \zeta$ and $\tau, \theta \in \{+, -\}$, each $\pi_{i+1}(u^\tau, v^\theta, \xi)$ can be computed as follows:*

- *If u, v are not the two neighbors of z_i in G_i, $\pi_{i+1}(u^\tau, v^\theta, \xi) = \pi_i(u^\tau, v^\theta, \xi)$, for each $\pi_{i+1} \in \Pi_{i+1}$ and $\pi_i \in \Pi_i$;*
- *Otherwise, we have:*
  - $$
    \begin{aligned}
    st_{i+1}(u^\tau, v^\theta, \xi) = \min\{&st_i(u^\tau, v^\theta, \xi_1) + yn_i(u^\tau, z_i, \xi_2) + ny^r_i(z_i, v^\theta, \xi_3) + y^\tau_u + y^\theta_v,\\
    &st_i(u^\tau, v^\theta, \xi_4) + st_i(u^\tau, z^+_i, \xi_5) + dt^r_i(z^+_i, v^\theta, \xi_6) + y^\tau_u + p^+_{z_i} + y^\theta_v,\\
    &st_i(u^\tau, v^\theta, \xi_7) + st_i(u^\tau, z^-_i, \xi_8) + dt^r_i(z^-_i, v^\theta, \xi_9) + y^\tau_u + p^-_{z_i} + y^\theta_v,\\
    &st_i(u^\tau, v^\theta, \xi_{10}) + dt_i(u^\tau, z^+_i, \xi_{11}) + st^r_i(z^+_i, v^\theta, \xi_{12}) + y^\tau_u + p^+_{z_i} + y^\theta_v,\\
    &st_i(u^\tau, v^\theta, \xi_{13}) + dt_i(u^\tau, z^-_i, \xi_{14}) + st^r_i(z^-_i, v^\theta, \xi_{15}) + y^\tau_u + p^-_{z_i} + y^\theta_v,\\
    &dt_i(u^\tau, v^\theta, \xi_{16}) + st_i(u^\tau, z^+_i, \xi_{17}) + st^r_i(z^+_i, v^\theta, \xi_{18}) + y^\tau_u + p^+_{z_i} + y^\theta_v,\\
    &dt_i(u^\tau, v^\theta, \xi_{19}) + st_i(u^\tau, z^-_i, \xi_{20}) + st^r_i(z^-_i, v^\theta, \xi_{21}) + y^\tau_u + p^-_{z_i} + y^\theta_v\\
    &\mid \text{for any } \xi_j \geq 0, j = 1, \ldots, 21 \text{ and } \xi_1 + \xi_2 + \xi_3 = \cdots = \xi_{19} + \xi_{20} + \xi_{21} = \xi\};
    \end{aligned}
    $$
  - $$
    \begin{aligned}
    dt_{i+1}(u^\tau, v^\theta, \xi) = \min\{&dt_i(u^\tau, v^\theta, \xi_1) + yn_i(u^\tau, z_i, \xi_2) + ny^r_i(z_i, v^\theta, \xi_3) + y^\tau_u + y^\theta_v,\\
    &dt_i(u^\tau, v^\theta, \xi_4) + st_i(u^\tau, z^+_i, \xi_5) + dt^r_i(z^+_i, v^\theta, \xi_6) + y^\tau_u + p^+_{z_i} + y^\theta_v,\\
    &dt_i(u^\tau, v^\theta, \xi_7) + st_i(u^\tau, z^-_i, \xi_8) + dt^r_i(z^-_i, v^\theta, \xi_9) + y^\tau_u + p^-_{z_i} + y^\theta_v,\\
    &dt_i(u^\tau, v^\theta, \xi_{10}) + dt_i(u^\tau, z^+_i, \xi_{11}) + st^r_i(z^+_i, v^\theta, \xi_{12}) + y^\tau_u + p^+_{z_i} + y^\theta_v,\\
    &dt_i(u^\tau, v^\theta, \xi_{13}) + dt_i(u^\tau, z^-_i, \xi_{14}) + st^r_i(z^-_i, v^\theta, \xi_{15}) + y^\tau_u + p^-_{z_i} + y^\theta_v\\
    &\mid \text{for any } \xi_j \geq 0, j = 1, \ldots, 21 \text{ and } \xi_1 + \xi_2 + \xi_3 = \cdots = \xi_{13} + \xi_{14} + \xi_{15} = \xi\};
    \end{aligned}
    $$

– $yn_{i+1}(u^\tau, v, \xi) = \min\{yn_i(u^\tau, v, \xi_1) + yn_i(u^\tau, z_i, \xi_2) + y_u^\tau,$
$yn_i(u^\tau, v, \xi_3) + st_i(u^\tau, z_i^+, \xi_4) + yn_i(z_i^+, v, \xi_5) + y_u^\tau + p_{z_i}^+,$
$yn_i(u^\tau, v, \xi_6) + st_i(u^\tau, z_i^-, \xi_7) + yn_i(z_i^-, v, \xi_8) + y_u^\tau + p_{z_i}^-$
$|\, for\ any\ \xi_j \geq 0, j = 1, \ldots, 8\ and\ \xi_1 + \xi_2 = \xi_3 + \xi_4 + x_5 = \xi_6 + \xi_7 + \xi_8 = \xi\};$

– $ny_{i+1}(u, v^\theta, \xi) = \min\{ny_i(u, v^\theta, \xi_1) + ny_i(z_i, v^\theta, \xi_2) + y_v^\theta,$
$ny_i(u, v^\theta, \xi_3) + ny_i(u, z_i^+, \xi_4) + st_i(z_i^+, v^\theta, \xi_5) + p_{z_i}^+ + y_v^\theta,$
$ny_i(u, v^\theta, \xi_3) + ny_i(u, z_i^-, \xi_4) + st_i(z_i^-, v^\theta, \xi_5) + p_{z_i}^- + y_v^\theta$
$|\, for\ any\ \xi_j \geq 0, j = 1, \ldots, 8\ and\ \xi_1 + \xi_2 = \xi_3 + \xi_4 + x_5 = \xi_6 + \xi_7 + \xi_8 = \xi\};$

– $nn_{i+1}(u, v) = \min\{nn_i(u, v, \xi),\ nn_i^r(u, z_i, \xi),\ nn_i^r(z_i, v, \xi),$
$ny_i^r(u, z_i^+, \xi_1) + yn_i^r(z_i^+, v, \xi_2) + y_{z_i}^+,$
$ny_i^r(u, z_i^-, \xi_3) + yn_i^r(z_i^-, v, \xi_4) + y_{z_i}^-$
$|\, for\ any\ \xi_j \geq 0, j = 1, \ldots, 4\ and\ \xi_1 + \xi_2 = \xi_3 + \xi_4 = \xi\}.$

Moreover, the corresponding solution $(T^{\pi_{i+1}(u^\tau, v^\theta, \xi)}, x^{\pi_{i+1}(u^\tau, v^\theta, \xi)}, y^{\pi_{i+1}(u^\tau, v^\theta, \xi)})$ *can also be computed.*

Theorem 17. *The* (BWC_PCST) *problem can be solved in* $O(n\zeta^3)$ *time in 2-trees.*

Proof. From Lemma 50, the (BWC_PCST) problem can be solved as long as we can compute all the measures $st_{n-2}(z_{n-1}^\tau, z_n^\theta, \zeta)$, $yn_{n-2}(z_{n-1}^\tau, z_n, \zeta)$, $ny_{n-2}(z_{n-1}, z_n^\theta, \zeta)$, $nn_{n-2}(z_{n-1}, z_n, \zeta)$ for any $\tau, \theta \in \{+, -\}$. From Lemma 51, for any $0 \leq \xi \leq \zeta$, each update takes $O(\xi^2)$ time, so it takes $O(\zeta^3)$ in all. We need to update $n-2$ times, so the time complexity is $O(n\zeta^3)$. □

Solving the Problem $(BWCC)$ in Lemma 48

Recall the problem $(BWCC)$ in Lemma 48 as follows:

$$
\begin{aligned}
(BWCC) \qquad & \min \nu(T, x, y) \\
\text{s.t.} \quad & r_s(T, x, y) \leq \Gamma; \\
& f \in E(T), u \in V(T), x_f = c_f^+, y_u = p_u^-; \\
& x_e \in \{c_e^+, c_e^-\},\ \forall\ e \in E(T) \setminus \{f\}; \\
& y_v \in \{p_v^+, p_v^-\},\ \forall\ v \in V(T) \setminus \{u\}.
\end{aligned}
$$

Remark 6. *The problem* $(BWCC)$ *is a constrained* (BWC_PCST) *problem, which requires that the a special edge* f *and special vertex* u *are contained in the tree and their risks are both 0s, i.e.* $x_f = c_f^+$ *and* $y_u = p_u^-$*; moreover, the sum-risks is at most* Γ*, i.e.* $\zeta = \Gamma$ *in the* (BWC_PCST) *problem.*

So to solve the $(BWCC)$, we just need to solve the constrained (BWC_PCST) problem, which has solutions containing f, u for $f \in E(G)$ and $u \in V(G)$ with $x_f = c_f^+$ and $y_u = p_u^-$. As proved in Lemma 32, we can assign very big prizes for u and the two endpoints of edge f, and assign small cost for f, then any optimal solution of (BWC_PCST) problem contains u and f. We describe this formally as follows.

Lemma 52. *Given $G=(V,E)$ with cost vectors $c^+\in\mathbb{R}_+^E, c^-\in\mathbb{R}_+^E$ and prize vectors $p^+\in\mathbb{R}_+^V, p^-\in\mathbb{R}_+^V$, and a real number $M>c^+(E)+p^+(V)$, an edge $f=ab\in E$ and a vertex $u\in V$, let p'^+ be defined by $p'^+_v=M$ for every $v\in\{u,a,b\}$, and $p'^+_v=p^+_v$ for every $v\in V\setminus\{u,a,b\}$; let p'^- be defined by $p'^-_v=M$ for every $v\in\{u,a,b\}$, and $p'^-_v=p^-_v$ for every $v\in V\setminus\{u,a,b\}$; let c'^+ be defined by $c'^+_f=0$, and $c'^+_e=c^+_e$ for every $e\in E\setminus\{f\}$; let c'^- be defined by $c'^-_f=0$, and $c'^-_e=c^-_e$ for every $e\in E\setminus\{f\}$.*

If (T,x',y') is an optimal solution of the (BWC_PCST) problem with input $(G,c'^+,c'^-,p'^+,p'^-,\Gamma)$, then define $x_f=c^+_f$, $x_e=x'_e$ for every $e\in E(T)\setminus\{f\}$ and $y_u=p^-_u$, $y_v=y'_v$ for every $v\in V(T)\setminus\{u\}$. Then (T,x,y) is an optimal solution of the $(BWCC)$.

So we have the following result:

Lemma 53. *For any edge $f\in E$ and vertex $u\in V$, the problem $(BWCC)$ can be solved in $O(n\Gamma^3)$ time.*

Summing up the Lemma 48, Lemma 49, Theorem 17 and Lemma 53, we solve the MSR_PCST.

Theorem 18. *The MSR_PCST problem in 2-trees can be solved in $O(n^6)$ time.*

Proof. First, for each $\zeta=0,\ldots,2n-1$, solve the (BWC_PCST) problem, in $O(n\zeta^3)$ time from Theorem 17, to compute its optimal value v_ζ. If for any $\zeta=0,\ldots,2n-1$, we have that $v_\zeta>\mathsf{B}$, then the MSR_PCST problem has no solution. Otherwise, let β be the minimum $\zeta\in\{0,\ldots,2n-1\}$ such that $v_\zeta\le\mathsf{B}$.

If $\beta=0$, then an optimal solution of the (BWC_PCST) problem with $\zeta=0$ is an optimal solution of the MSR_PCST problem; moreover, its optimal value $r^*=0$. So the problem is solved in $O(n^4)$ since $\zeta\le 2n-1$.

Otherwise, $\beta>0$. Then $v_\zeta>\mathsf{B}$. So we have $r^*>0$.

For each vertex $v\in V$, if $p^+_v\ge-\mathsf{B}\ge p^-_v$, then let $y_v=-\mathsf{B}$. So the tree, containing only one vertex v, together with its prize $y_v=-\mathsf{B}$ is the solution of the MSR_PCST problem containing only one vertex v with the minimum sum-risks. Let $\mathcal{A}_1$ be the set of all such solutions containing only one vertex. So it takes $O(n)$ time to obtain $\mathcal{A}_1$.

For each vertex $u\in V$ and each edge $f\in E$, and each $\Gamma\in\{\beta,\beta-1,\beta-2\}\cap\mathbb{R}_+$, solve the programming $(BWCC)$ in $O(n\Gamma^3)$ time from Lemma 53, to obtain an optimal solution (T,x,y) of $(BWCC)$. Then solve the linear programming (LP), presented in Lemma 48, to obtain an optimal solution (x_0,y_0) of (LP). Define the vectors $x'\in\mathbb{R}_+^{E(T)}$ and $y'\in\mathbb{R}_+^{V(T)}$ as: $x'_f=x_0, y'_u=y_0$, $x'_e=x_e$ for any edge $e\in E(T)\setminus\{f\}$ and $y'_v=y_v$ for any $v\in V(T)\setminus\{u\}$. So we get such a solution (T,x',y') for each vertex $u\in V$ and each edge $f\in E$, and each $\Gamma\in\{\beta,\beta-1,\beta-2\}\cap\mathbb{R}_+$. Let $\mathcal{A}_2$ be the set of all these $O(n^2)$ solutions. So it takes $O(n^6)$ time to obtain $\mathcal{A}_2$.

Choose the $(T,x,y)\in(\mathcal{A}_1\cup\mathcal{A}_2)$, which has the minimum $r_s(T,x,y)$. In the following, we prove that (T,x,y) is an optimal solution of the MSR_PCST problem.

If any optimal solution of the MSR_PCST problem contains only one vertex, then $\mathcal{A}_1$ contains all the optimal solutions. So (T,x,y) is optimal.

Otherwise, let (T^*,x^*,y^*) be an extreme optimal solution with extreme edge $f\in E(T^*)$ and extreme vertex $u\in V(T^*)$ of the MSR_PCST problem. Then from Lemma 49,

$r_s(T^*,x^*,y^*)-r(x_f^*)-r(y^*u)\in\{\beta,\beta-1,\beta-2\}$. So the extreme optimal solution (T,x',y') as in Lemma 48 is contained in $\mathcal{A}$. So (T,x,y) is optimal.

So the MSR_PCST problem in 2-trees is solved in $O(n^6)$ time. □

Remark 7. *Recall that any graph of treewidth at most 2 can be transformed to a 2-tree by adding some edges (seen in Lemma 33). Given any graph of treewidth at most 2, we add some edges to get a 2-tree in $O(n)$ time. Assign a cost interval $[M,M]$ for each new added edge, where M is very huge number, e.g. the sum of all upper bounds of the costs and prizes. Then solve the MMR_PCST and MSR_PCST problems in the obtained 2-trees. As proved in Lemma 34, the new added edges are not contained in the optimal solutions. So the MMR_PCST and MSR_PCST problems in graphs of treewidth at most 2 can also be solved in polynomial time.*

5.3.3 Discussion

In the preceding subsections we propose two risk models, the MSR_PCST and the MMR_PCST, for the PCST problem with interval data. These two models with the same data setting may yield different solutions as shown by the example depicted in Fig.5.4.

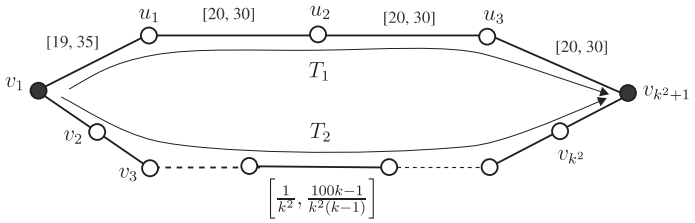


Figure 5.4: Min-max risk model and Min-sum model are different.

Both models investigate a cycle on k^2+4 vertices, where $k\geq 8$ is an integer. The cost interval of every edge is specified beside the edge. In particular, $c_e^- = 1/k^2$ and $c_e^+ = (100k-1)/(k^2(k-1))$, for edge $e = v_iv_{i+1}$, $i = 1,2,\cdots,k^2$. Every vertex has prize interval $[0,0]$. The target set consists of two black vertices v_1 and v_{k^2+1}. The tree in any solution to the MSR_PCST problem or to the MMR_PCST problem must be one of the two paths $T_1 = v_1u_1u_2u_3v_{k^2+1}$ and $T_2 = v_1v_2v_3\ldots v_{k^2+1}$ between v_1 and v_{k^2+1}. Given budget bound $\mathsf{B} = 100$, using Lemmas 40 and 41, one can easily verify that the optimal solution to the MMR_PCST is $(T_2, \frac{100}{k^2}\mathbf{1}, \mathbf{0})$ with maximum risk $1/k$, because every feasible solution $(T_1,x,\mathbf{0})$ must have some $e\in E(T_1)$ with $x_e\leq 25$, which has risk at least $\min\{\frac{25-20}{30-20}, \frac{25-19}{35-19}\} = \frac{3}{8} > \frac{1}{k}$. On the other hand, the optimal solution to the MSR_PCST problem is $(T_1,x,\mathbf{0})$ with $x = (20,30,30,20)$ that has risk sum $\frac{17}{16}$, because every feasible solution on T_2 has risk sum at least $k\geq 8$. In this example, the optimal solution to the MMR_PCST problem is very inefficient for the MSR_PCST problem in case of large k, and vice versa.

However, in practice, we simulate the MMR_PCST and the MSR_PCST models in various of network situations to investigate the solution behaviors and average performance of both models. We discover that in average the optimal solution of the MMR_PCST

model is not a very bad solution for the MSR_PCST model. But the optimal solution of the MSR_PCST model may be very far from the optimal solution of the MSR_PCST model.

5.4 Perspectives

In this chapter, we have presented a dynamic programming algorithm for the PCST problem in graphs of treewidth at most 2. This algorithm has been generalized in [CMZ11] to solve the PCST problem in graphs of treewidth at most k, for any fixed integer k, by using more general dynamic programming base on a tree decomposition of bounded width.

Moreover, we have proposed two risk models for the PCST problem with interval data and solved them in graphs of treewidth at most 2. Compared with other models for uncertain optimization problems, their advantages lie in not only keeping computational complexity unchanged but also providing flexibility for decision makers. So far risk models have been applied in dealing with several polynomial-time solvable problems. In the future, it is worthwhile studying how to apply risk models to some other optimization problems with interval data, particularly NP-hard optimization problems. Moreover, it is interesting to see if our models and approaches can be extended to the problems where uncertainty is not described using intervals.

CHAPTER 6

Data Gathering and Personalized Broadcasting in Radio Grids

Contents

In this chapter, we study a class of routing problem that was motivated by designing efficient strategies to provide Internet access using wireless devices [BBS05]. Typically, several houses in a village need access to a gateway (for example a satellite antenna) to transmit and receive data over the Internet. To reduce the cost of the transceivers, multi-hop wireless relay routing is used. We formulate this problem as gathering information in a Base Station (denoted by BS) of a wireless multi-hop network when interferences constraints are present. This problem is also known as data collection.

The results of this chapter is a collaboration with J. C. Bermond, N. Nisse, M.-L. Yu and H. Rivano. The work is submitted to the journal Theoretical Computer Science [s-BLN+13].

6.1 Introduction

6.1.1 Problem, model and assumptions

Gathering problem is very important in sensor networks and access networks. Depending on the functions of the devices in the networks, the problem have different models and assumptions.

Transmission model We adopt the network model considered in [BGK$^+$06, BGR10, BKK$^+$10, FFM04, RS08]. The network is represented by a node-weighted symmetric digraph $G = (V, E)$, where V is the set of nodes and E is the set of arcs. More specifically, each node in V represents a *device* (sensor, station, ...) that can transmit and receive data. There is a special node $BS \in V$ called the *Base Station (BS)*, which is the final destination of all data possessed by the various nodes of the network. Each node may have any number of pieces of information, or *messages*, to transmit, including none. There is an arc from u to v if u can transmit a message to v. We suppose that the digraph is symmetric; so if u can transmit a message to v, then v can also transmit a message to u. Therefore G represents the graph of possible communications. Some authors use an undirected graph (replacing the two arcs (u,v) and (v,u) by an edge $\{u,v\}$). However *calls* (transmissions) are directed: a call (s,r) is defined as the transmission from the node s to node r, in which s is the *sender* and r is the *receiver* and s and r are adjacent in G. The distinction of sender and receiver will be important for our interference model.

Here we will consider grids as they model well both access networks and also random networks [KLNP09]. The network is assumed to be synchronous and the time is slotted into *steps*. During each step, a transmission (or a *call*) between two nodes can transport at most one message. That is, a step is a unit of time during which several calls can be done as long as they do not interfere with each other. We suppose that each device is equipped with an half duplex interface: a node cannot both receive and transmit during a step. This models the near-far effect of antennas: when one is transmitting, it's own power prevents any other signal to be properly received. Moreover, we assume that a node can transmit or receive at most one message per step.

Following [FFM04, GR09, RS07, RS08, ZTG05] we assume that no buffering is done at intermediate nodes and each node forwards a message as soon as it receives it. One of the rationales behind this assumption is that it frees intermediate nodes from the need to maintain costly state information and message storage.

Interference model We use a binary asymmetric model of interference based on the distance in the communication graph. Let $d(u,v)$ denote the distance, that is the length of a shortest directed path, from u to v in G and d_I be an nonnegative integer. We assume that when a node u transmits, all nodes v such that $d(u,v) \leq d_I$ are subject to the interference from u's transmission. We assume that all nodes of G have the same interference range d_I. Two calls (s,r) and (s',r') do not interfere if and only if $d(s,r') > d_I$ and $d(s',r) > d_I$. Otherwise calls interfere (or there is a collision). We will focus on the cases when $d_I \leq 2$.

Note that we suppose in this chapter $d_I \geq 0$. It implies that a node cannot receive and send simultaneously.

The binary interference model is a simplified version of the reality, where the Signal-to-Noise-and-Interferences Ratio (the ratio of the received power from the source of the transmission to the sum of the thermic noise and the received powers of all other simultaneously transmitting nodes) has to be above a given threshold for a transmission to be successful. However, the values of the completion times that we obtain will lead to lower bounds on the corresponding real life values. Stated differently, if the value of the completion time is fixed, then our results will lead to upper bounds on the maximum possible number of messages that can be transmitted in the network.

Gathering and Personalized broadcasting Our goal is to design protocols that will efficiently, i.e., quickly, gather all messages to the base station BS subject to these interference constraints. More formally, let $G = (V, E)$ be a connected symmetric digraph, $BS \in V$ and and $d_I \geq 0$ be an integer. Each node in $V \setminus BS$ is assigned a set (possibly empty) of messages that must be sent to BS. A multi-hop schedule for a message consists of the path it must follow to reach BS together with the starting step (because no buffering is allowed, the starting step defines the whole schedule). The *gathering problem* consists in computing a multi-hop schedule for each message to arrive the BS under the constraint that during any step any two calls do not interfere within the interference range d_I. The completion time or *makespan* of the schedule is the number of steps used for all messages to reach BS. We are interested in computing the schedule with minimum makespan.

Actually, we will describe the gathering schedule by illustrating the schedule for the equivalent *personalized broadcasting problem* since this formulation allows us to use a simpler notation and simplify the proofs. In this problem, the base station BS has initially a set of *personalized* messages and they must be sent to their destinations, i.e., each message has a personalized destination in V, and possibly several messages may have the same destination. The problem is to find a multi-hop schedule for each message to reach its corresponding destination node under the same constraints as the gathering problem. The completion time or *makespan* of the schedule is the number of steps used for all messages to reach their destination and the problem aims at computing a schedule with minimum makespan. To see that these two problems are equivalent, from any personalized broadcasting schedule, we can always build a gathering schedule with the same makespan, and the other way around. Indeed, consider a personalized broadcasting schedule with makespan $\mathcal{T}$. Any call (s, r) occurring at step k corresponds to a call (r, s) scheduled at step $\mathcal{T} + 1 - k$ in the corresponding gathering schedule. Furthermore, as the digraph is symmetric, if two calls (s, r) and (s', r') do not interfere, then $d(s, r') > d_I$ and $d(s', r) > d_I$; so the reverse calls do not interfere. Hence, if there is an (optimal) personalized broadcasting schedule from BS, then there exists an (optimal) solution for gathering at BS with the same makespan. The reverse also holds. Therefore, in the sequel, we consider the personalized broadcasting problem.

6.1.2 Related Work

Gathering problems like the one that we study in this chapter have received much recent attention. The papers most closely related to our results are [BGP$^+$11, BGR10, FFM04, GR09, RS07, RS08]. Paper [FFM04] firstly introduced the data gathering problem in a model for sensor networks similar to the one adopted in this chapter. It deals with $d_I = 0$ and gives optimal gathering schedules for trees. Optimal algorithms for star networks are given in [RS08] find the optimal schedule minimizing both the completion time and the average delivery time for all the messages. Under the same hypothesis, an optimal algorithm for general networks is presented in [GR09] in the case each node has exactly one message to deliver. In [BGR10] (resp [BGP$^+$11]) optimal gathering algorithms for tree networks in the same model considered in this chapter, are given when $d_I = 1$ (resp.,$d_I \geq 2$). In [BGP$^+$11] it is also shown that the Gathering Problem is NP-complete if the process must be performed along the edges of a *routing tree* for $d_I \geq 2$ (otherwise the complexity is not determined). Furthermore, for $d_I \geq 1$ a simple $(1 + \frac{2}{d_I})$ factor approximation algorithm is given for general networks. In slightly different settings, in particular the assumption of directional antennas, the problem has been proved NP-hard in general networks [RSY10]. The case of *open-grid* where *BS* stands at a corner and no messages have destinations in the first row or first column, called axis in the following, is considered in [RS07], where a 1.5-approximation algorithm is presented.

Other related results can be found in [BCY09, BGK$^+$06, BP12, BY10, BKSS08] (see [BKK$^+$10] for a survey). In these articles data buffering is allowed at intermediate nodes, achieving a smaller makespan. In [BGK$^+$06], a 4-approximation algorithm is given for any graph. In particular the case of grids is considered in [BP12], but with exactly one message per node. Another related model can be found in [GPRR08], where steady-state (continuous) flow demands between each pair of nodes have to be satisfied, in particular, the authors also study the gathering in radio grid networks.

6.1.3 Main results

In this chapter, we propose algorithms to solve the personalized broadcasting problem (and so the equivalent gathering problem) in a grid with the model described above (synchronous, no buffering, one message transmission per step, with an interference parameter d_I). Initially all messages stand at the base station *BS* and each message has a particular destination node (possibly several messages may be sent to the same node). Our algorithms compute in linear time (in the number of messages) schedules with no calls interfering, with a makespan differing from the lower bound by a small additive constant. We first study the basic instance consisting of an open grid where no messages have destination on an axis, with a BS in the corner of the grid and with $d_I = 0$. This is exactly the same case as that considered in [RS07]. In Section 6.2 we give a simple lower bound *LB*. Then in Section 6.3 we design for this basic instance a linear time algorithm with a makespan at most $LB + 2$ steps, so obtaining a $+2$-approximation algorithm for the open grid, which greatly improves the multiplicative 1.5 approximation algorithm of [RS07] . Such an algorithm has already been given in the extended abstract [BNRR09]; but the one given here

is simpler and we can refine it to obtain for the basic instance a $+1$-approximation algorithm. Then we prove in Section 6.4 that the $+2$-approximation algorithm works also for a general grid where messages can have destinations on the axis again with *BS* in the corner and $d_I = 0$. Then we consider in Section 6.5 the cases $d_I = 1$ and 2. We give lower bounds $LB_c(1)$ (when *BS* is in the corner) and $LB(2)$ and show how to use the $+1$-approximation algorithm given in Section 6.3 to design algorithms with a makespan at most $LB_c(1)+3$ when $d_I = 1$ and *BS* is in the corner , and at most $LB(2)+4$ when $d_I = 2$; however the coordinates of the destinations have in both cases to be at least 2. In Section 6.6, we extend our results to the case where BS is in a general position in the grid. In addition, we point out that our algorithms are 2-approximations if the buffering is allowed, which improves the result of [BGK$^+$06] in the case of grids with $d_I \leq 2$. Finally, we conclude this chapter in Section 6.7. The main results are summarized in Table 6.1.

Interference	**Additional hypothesis**	**Performances**	
		no buffering	buffering
$d_I = 0$		+2-approximation	
	no messages on axes	+1-approximation	
$d_I = 1$	BS in a corner and no messages "close" to the axes (see Def. 10)	+3-approx.	$\times$1.5-approx.
	no messages at distance ≤ 1 from an axis	$\times$1.5-approximation	
$d_I = 2$	no messages at distance ≤ 1 from an axis	+4-approx.	$\times$2-approx.

Table 6.1: Performances of the algorithms designed in this chapter. Our algorithms deal with the gathering and personalized broadcasting problems in a grid with arbitrary base station (unless stated otherwise). In this table, $+c$-approximation means that our algorithm achieves an optimal makespan up to an additive constant c. Similarly, $\times c$-approximation means that our algorithm achieves an optimal makespan up to an multiplicative constant c.

6.2 Notations and Lower bound

In the following, we consider a grid $G = (V,E)$ with a particular node, the base station *BS*, also called *the source*. A node v is represented by its coordinates (x,y). The source *BS* has coordinates $(0,0)$. We define the *axis* of the grid with respect to *BS*, as the set of nodes $\{(x,y) : x = 0\}$ or $\{(x,y) : y = 0\}$. The distance between two nodes u and v is the length of a shortest directed path in the grid and will be denoted by $d(u,v)$. In particular, $d(BS,v) = |x| + |y|$.

We consider a set of $M > 0$ messages that must be sent from the source *BS* to some destination nodes. Note that *BS* is not a destination node. Let $dest(m) \in V$ denote the destination of the message m. We use $d(m) > 0$ to denote the distance $d(BS, dest(m))$. We suppose that the messages are ordered by non-increasing distance from *BS* to their destina-

tion nodes, and we denote this ordered set $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ where $d(m_1) \geq d(m_2) \geq \cdots \geq d(m_M)$. The input of all our algorithms is the ordered sequence $\mathcal{M}$ of messages. For simplicity we suppose that the grid is infinite; however it suffices to consider a grid slightly greater than the one containing all the destinations of messages. Note that our work does not include the case of the paths, already considered in [BCY09, FFM04, RS07].

We will use the name of *open grid* to mean that no messages have destination on an axis that is when all messages have destination nodes in the set $\{(x,y) : x \neq 0 \text{ and } y \neq 0\}$.

Note that in our model the source can send at most one message per step. Given a set of messages that must be sent by the source, a *broadcasting scheme* consists in indicating for each message m the time at which the source sends the message m and the directed path followed by this message. More precisely a broadcasting scheme will be represented by an *ordered sequence* of messages $\mathcal{S} = (s_1, \cdots, s_k)$, where furthermore for each s_i we give the directed path P_i followed by s_i and the time t_i at which the source sends the message s_i. The sequence is ordered in such a way message s_{i+1} is sent after message s_i, that is we have $t_{i+1} > t_i$.

As we suppose there is no buffering, a message m sent at step t_m is received at step $t'_m = l_m + t_m - 1$, where l_m is the length of the directed path followed by the message m. In particular $t'_m \geq d(m) + t_m - 1$. The *completion time or makespan* of a broadcasting scheme is the step where all the messages have arrived at their destinations. Its value is $\max_{m \in \mathcal{M}} l_m + t_m - 1$. In the next proposition we give a lower bound of the makespan:

Proposition 1. *Given the set of messages $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ ordered by non-increasing distance from BS, the makespan of any broadcasting scheme is greater than or equal to $LB = \max_{i \leq M} d(m_i) + i - 1$.*

Proof. Consider any personalized broadcasting scheme. For $i \leq M$, let t_i be the step where the last message in $(m_1, m_2, \cdots, m_i)$ is sent; therefore $t_i \geq i$. This last message denoted m is received at step $t'_i \geq d(m) + t_i - 1 \geq d(m_i) + t_i - 1 \geq d(m_i) + i - 1$. So the makespan is at least $LB = \max_{i \leq M} d(m_i) + i - 1$.

□

Note that this result is valid for any topology (not only grids) since it uses only the fact that the source sends at most one message per step. If there are no interference constraints, in particular if a node can send and receive messages simultaneously, then the bound is achieved by the greedy algorithm where at step i the source sends the message m_i of the ordered sequence $\mathcal{M}$ through a shortest directed path from BS to $dest(m_i)$, i.e. the makespan LB is attained by BS sending all the messages through the shortest paths to their destinations according to the non-increasing distance ordering.

If there are interferences and $d_I > 0$, we will design in Section 6.5 some better lower bounds. If $d_I = 0$, we will design in the next two sections linear time algorithms with a makespan at most $LB + 2$ in the grid with the base station in the corner and a makespan at most $LB + 1$ when furthermore there is no message with a destination node on the axis (open-grid). In case $d_I = 0$ in open grid, our algorithms are simple in the sense that they use only very simple shortest directed paths and that BS never waits.

Example 1. *Here, we exhibit examples for which the optimal makespan is strictly larger than LB. In particular, in the case of general grids, $LB+2$ can be optimal. On the other hand, results of this chapter show that the optimal makespan is at most $LB+1$ in the case of open-grids for $d_I = 0$ (Theorem 22) and at most $LB+2$ in general grids for $d_I = 0$ (Theorem 25). In case $d_I = 0$ and in open-grids, our algorithms use shortest paths and the BS sends a message at each step. We also give examples for which optimal makespan cannot be achieved in this setting.*

Let us remark that there exist configurations for which no personalized broadcasting protocol can achieve better makespan than $LB+1$. Figure 6.1(a) represents such a configuration. Indeed, in Figure 6.1(a), message m_i has a destination node v_i for $i = 1,2,3$ and $LB = 7$. However, to achieve the makespan $LB = 7$ for $d_I = 0$, BS must send the message m_1 to v_1 at step 1 (because v_1 is at distance 7 from BS) and must send message m_2 to v_2 at step 2 (because the message starts after the first step and must be sent to the destination node at distance 6) and these messages should be sent along shortest directed paths. To avoid interferences, the only possibility is that BS sends the first message to node $(0,1)$, and the second one to the node $(1,0)$. Intuitively, this is because otherwise the shortest paths followed by first two messages would intersect in such a way that interference cannot be avoided. A formal proof can be obtained from Fact 6 in Section 6.3.2. But then, if we want to achieve the makespan of 7, BS has to send the message m_3 via node $(0,1)$ and it will reach v_3 at step 7; but the directed paths followed by m_2 and m_3 need to cross and at this crossing point m_3 arrives at a step where m_2 leaves and so the messages interfere. So BS has to wait one step and sends m_3 only at step 4. Then the makespan is $8 = LB+1$.

In addition, there are also examples in which BS has to wait for some steps after sending one message in order to reach the lower bound LB for $d_I = 0$. Figure 6.1(b) represents such an example. To achieve the lower bound 7, BS has to send messages using shortest directed paths firstly to v_1 via $(3,0)$ and then consecutively sends messages to v_2 via $(0,4)$ and v_3 via $(2,0)$. If BS sends message m_4 at step 4, then m_4 will interfere with m_3. But, to avoid this interference, BS can send message m_4 at step 5 and will reach v_4 at step 7.

There are also examples in which no schedule using only shortest directed paths achieves the optimal makespan[1]*. For instance, consider the grid with four messages to be sent to $(0,4),(0,3),(0,2)$ and $(0,1)$ (all on the first column) and let $d_I = 0$ (a more elaborate example with an open-grid is given in Example 6.6(a)). Clearly, sending all messages through shortest directed paths implies that BS sends messages every two steps. Therefore, it requires 7 steps. On the other hand, the following scheme has makespan 6: send the message to $(0,4)$ through the unique shortest directed path at step 1; send the message to $(0,3)$ at step 2 via nodes $(1,0),(1,1),(1,2)(1,3)$; send the message to $(0,2)$ through the shortest directed path at step 3 and, finally, send the message to $(0,1)$ at step 4 via nodes $(1,0),(1,1)$. Note that the optimal makespan is in this example $LB+2$.*

[1]The authors would like to thanks Prof. Frédéric Guinand who raised this question.

(a) Configuration when the trivial lower bound cannot be achieved.

(b) BS has to wait for one step to achieve the trivial lower bound.

Figure 6.1: Two particular configurations

6.3 Basic instance: $d_I = 0$, open-grid, and BS in the corner

In this section we study simple configurations called *basic instances*. A basic instance is a configuration where $d_I = 0$, messages are sent in the open grid (no destinations on the axis) and BS is in the corner (a node with degree 2 in the grid). We will see that we can find personalized broadcasting algorithms using a basic scheme, where each message is sent via a simple shortest directed path (with one horizontal and one vertical segment) and where the source sends a message at each step (it never waits) and achieving a makespan of at most $LB + 1$.

6.3.1 Basic schemes

A message is said to be sent *horizontally* to its destination $v = (x, y)$ $(x > 0, y > 0)$, if it goes first horizontally then vertically, that is if it follows the shortest directed path from *BS* to v passing through $(x, 0)$. Correspondingly, the message is sent *vertically* to its destination $v = (x, y)$, if it goes first vertically then horizontally, that is if it follows the shortest directed path from *BS* to v passing through $(0, y)$. We will use the notation **a message is sent in direction** D, where $D = H$ (for horizontally) (resp. $D = V$ (for vertically)) if it is sent horizontally (resp. vertically). Also, $\bar{D}$ **will denote the direction different from** D that is $\bar{D} = V$ (resp. $\bar{D} = H$) if $D = H$ (resp. $D = V$).

Definition 9. [basic scheme] *A basic scheme is a broadcasting scheme where BS sends a message at each step alternating horizontal and vertical sendings. Therefore it is represented by an ordered sequence* $\mathcal{S} = (s_1, s_2, \ldots, s_M)$ *of the M messages with the properties: message* s_i *is sent at step i and furthermore, if* s_i *is sent in direction D, then* s_{i+1} *is sent in direction* $\bar{D}$.

Notation: Note that, by definition of horizontal and vertical sendings, the basic scheme defined below uses shortest paths. Moreover, as soon as we fix $\mathcal{S}$ and the sending direction D of the first or last message, the directed paths used in the scheme are uniquely determined. Hence, the scheme is characterized by the sequence $\mathcal{S}$ and the direction D. We will use when needed, the notation $(\mathcal{S}, first = D)$ to indicate a basic scheme where the first

message is sent in direction D, and the notation $(\mathcal{S}, last = D)$ when the last message is sent in direction D.

6.3.2 Interference of messages

Our aim is to design an *admissible* basic scheme in which the messages are broadcasted without any collisions. The following simple fact shows that we only need to take care of consecutive sendings. In the following, we say that two messages are *consecutive* if the source sends them consecutively (one at step t and the other at step $t+1$)

Fact 5. *When $d_I = 0$, in any broadcast scheme using only shortest paths (in particular in a basic scheme), only consecutive messages may interfere.*

Proof. By definition, a basic scheme uses only shortest paths. Let the message m be sent at step t and the message m' at step $t' \geq t+2$. Let $t'+h$ ($h \geq 0$) be a step such that the two messages have not reached their destinations. As we use shortest directed paths the message m is sent on an arc (u,v) with $d(v,BS) = d(u,BS)+1 = t'+h-t+1$, while message m' is sent on an arc (u',v') with $d(v',BS) = d(u',BS)+1 = h+1$. Therefore, $d(u,v') \geq t'-t-1 \geq 1 > 0 = d_I$ and $d(u',v) \geq t'-t+1 \geq 3 > d_I$. □

We now characterize the situations when two consecutive messages interfere in a basic scheme. For that we use the following notation:

Notation: In the case $d_I = 0$, if BS sends in direction $D \in \{V,H\}$ the message m at step t and sends the message m' in the other direction $\bar{D}$, at step $t' = t+1$, we will write $(m,m') \in D\bar{D}$ if they do not interfere and $(m,m') \notin D\bar{D}$ if they interfere.

(a) $(m,m') \notin HV$ (b) $(m,m') \notin VH$

Figure 6.2: Cases of interferences

Fact 6. *Let m and m' be two consecutive messages in a basic scheme. Then, $(m,m') \notin D\bar{D}$ if and only if the paths followed by the messages in the basic scheme intersect in a vertex which is not the destination of m.*

Proof. Suppose the directed paths intersect in a node v that is not the destination of m. The message m sent at step t has not reached its destination and so leaves the node v at step

$t+d(v,BS)$; but the message m' sent at step $t+1$ arrives at node v at step $t+d(v,BS)$ and therefore the two messages interfere.

Conversely if the two directed paths used for m and m' do not cross then the messages do not interfere. If the paths intersect only in the destination $dest(m)$ of m, then m' arrives in $dest(m)$ one step after m has stopped in $dest(m)$ and so the two messages do not interfere. □

Remark 8. *Note that Fact 6 does not hold if we do not impose basic schemes (i.e., this is not true if any shortest paths are considered). Moreover, we emphasize that the two paths may intersect, but the corresponding messages do not necessarily interfere.*

In some proofs throughout this chapter, we will need to use the coordinates of the messages. Therefore, the following equivalent statement of Fact 6 will be of interest. Let $dest(m)=(x,y)$ *and* $dest(m')=(x',y')$. *Then*

- $(m,m') \notin HV$ *if and only if* $\{x' \geq x$ *and* $y' < y\}$;
- $(m,m') \notin VH$ *if and only if* $\{x' < x$ *and* $y' \geq y\}$;

Figure 6.2 shows when there are interferences and also illustrates Fact 6 for $D=H$ (resp. V) in case (a) (resp. (b)).

6.3.3 Basic lemmas

We now prove some simple but useful lemmata.

Lemma 54. *If* $(m,m') \notin D\bar{D}$, *then* $(m,m') \in \bar{D}D$ *and* $(m',m) \in D\bar{D}$.

Proof. By Fact 6, if $(m,m') \notin D\bar{D}$, then the two directed paths followed by m and m' in the basic scheme (in directions D and $\bar{D}$ respectively) intersect in a node different from $dest(m)$. Then, the two directed paths followed by m and m' in the basic scheme (in directions $\bar{D}$ and D respectively) do not intersect. Hence, by Fact 6, $(m,m') \in \bar{D}D$. Similarly, the two directed paths followed by m' and m in the basic scheme (in directions D and $\bar{D}$ respectively) do not intersect. Hence, by Fact 6, $(m',m) \in D\bar{D}$.

□

Note that this lemma is enough to prove the multiplicative $\frac{3}{2}$ approximation obtained in [RS07]. Indeed the source can send at least two messages every three steps, in the order of $\mathcal{M}$. More precisely, *BS* sends any pair of messages m_{2i-1} and m_{2i} consecutively by sending the first one horizontally and the second one vertically if $(m_{2i-1},m_{2i}) \in HV$, otherwise sending the first one vertically and the second one horizontally if $(m_{2i-1},m_{2i}) \notin HV$ (since this implies that $(m_{2i-1},m_{2i}) \in VH$). Then the source does not send anything during the third step. So we can send $2q$ messages in $3q$ steps. Such a scheme has makespan at most $\frac{3}{2}LB$.

Note that in general, $(m,m') \in D\bar{D}$ does not imply $(m',m) \in \bar{D}D$, namely when the directed paths intersect only in the destination of m which is not the destination of m'.

Lemma 55. *If* $(m,m') \in D\bar{D}$ *and* $(m',m'') \notin \bar{D}D$, *then* $(m,m'') \in D\bar{D}$.

Proof. By Fact 6, $(m,m') \in D\bar{D}$ implies that the paths followed by m and m' (in directions D and $\bar{D}$ respectively) in the basic scheme may intersect only in $dest(m)$. Moreover, $(m',m'') \notin \bar{D}D$ implies that the paths followed by m' and m'' (in directions $\bar{D}$ and D respectively) intersect in a node which is not $dest(m')$. Simple check shows that the paths followed by m and m'' (in directions D and $\bar{D}$ respectively) may intersect only in $dest(m)$. Therefore, by Fact 6, $(m,m'') \in D\bar{D}$.

□

Lemma 56. *If $(m,m') \notin D\bar{D}$ and $(m,m'') \notin \bar{D}D$, then $(m',m'') \in D\bar{D}$.*

Proof. By Lemma 54 $(m,m') \notin D\bar{D}$ implies $(m',m) \in D\bar{D}$. Then we can apply the preceding Lemma 55 with m',m,m'' in this order to get the result. The second claim is obtained similarly. □

6.3.4 Makespan can be approximated up to additive constant 2

Recall that $\mathcal{M} = (m_1,\ldots,m_M)$ is the set of messages ordered by non-increasing distance from *BS*. Throughout this chapter, $S \odot S'$ denotes the sequence obtained by the concatenation of two sequences S and S'.

In [BNRR09], we use a basic scheme to design an algorithm for broadcasting the messages in the basic instance with a makespan at most $LB+2$. We give here a different algorithm with similar properties, but easier to prove and which presents two improvements: it can be adapted to the case where the destinations of the messages may be on the axes (i.e. for general grid) (see Section 6.4) and it can be refined to give in the basic instance a makespan at most $LB+1$. We denote the algorithm by $TwoApprox[d_I = 0, last = D](\mathcal{M})$; for an input set of messages $\mathcal{M}$ ordered by non-increasing distances from *BS*, and a direction $D \in \{H,V\}$, it gives as output an ordered sequence $\mathcal{S}$ of the messages such that the basic scheme $(\mathcal{S}, last = D)$ has makespan at most $LB+2$. Recall that D is the direction of the last sent message in $\mathcal{S}$ in Definition 9.

The algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ is given in Figure 6.3. It uses a basic scheme, where the non-increasing order is kept, if there are no interferences; otherwise we change the order a little bit. To do that, we apply dynamic programming. We examine the messages in their order and at a given step we add to the current ordered sequence the two next unconsidered messages. We show that we can avoid interferences, only by reordering these two messages and the last one in the current sequence.

Remark 9. *Notice that, there are instances (see examples below) for which Algorithm TwoApprox computes an optimal makespan only for one direction. Hence, it may sometimes be interesting to apply the algorithm for each direction and take the better resulting schedule.*

Because of the behavior of a basic scheme, the direction of the final message and of the first one are simply linked via the parity of the number of messages. Hence, we can also derive an algorithm $TwoApprox[d_I = 0, first = D](\mathcal{M})$ that has the first direction D of the message as an input.

Input: $\mathcal{M} = (m_1, \cdots, m_M)$, the set of messages ordered by non-increasing distances from *BS* and the direction $D \in \{H, V\}$ of the last message.
Output: $\mathcal{S} = (s_1, \cdots, s_M)$ an ordered sequence of the M messages satisfying (i) and (ii) (See in Theorem 19)
begin
1 **Case** $M = 1$**: return** $\mathcal{S} = (m_1)$
2 **Case** $M = 2$**:**
3 **if** $(m_1, m_2) \in \bar{D}D$ **return** $\mathcal{S} = (m_1, m_2)$
4 **else return** $\mathcal{S} = (m_2, m_1)$
5 **Case** $M > 2$**:**
6 let $\mathcal{O} \odot p = TwoApprox[d_I = 0, last = D](m_1, \cdots, m_{M-2})$
(p is the last message in the obtained sequence)
7 **Case 1: if** $(p, m_{M-1}) \in D\bar{D}$ and $(m_{M-1}, m_M) \in \bar{D}D$ **return** $\mathcal{O} \odot (p, m_{M-1}, m_M)$
8 **Case 2: if**$(p, m_{M-1}) \in D\bar{D}$ and $(m_{M-1}, m_M) \notin \bar{D}D$ **return** $\mathcal{O} \odot (p, m_M, m_{M-1})$
9 **Case 3: if**$(p, m_{M-1}) \notin D\bar{D}$ and $(p, m_M) \in \bar{D}D$ **return** $\mathcal{O} \odot (m_{M-1}, p, m_M)$
10 **Case 4: if**$(p, m_{M-1}) \notin D\bar{D}$ and $(p, m_M) \notin \bar{D}D$ **return** $\mathcal{O} \odot (p, m_M, m_{M-1})$

end

Figure 6.3: Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$

Example 2. *Here, we give examples that illustrate the execution of Algorithm TwoApprox. Moreover, we describe instances for which it is not optimal.*

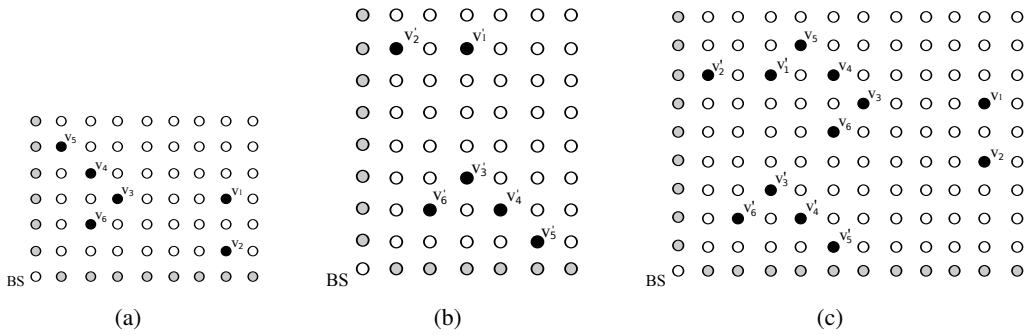
Consider the example of Figure 6.4(a). The destinations of the messages m_i ($1 \leq i \leq 6$) are $v_1 = (7,3)$, $v_2 = (7,1)$, $v_3 = (3,3)$, $v_4 = (2,4)$, $v_5 = (1,5)$ and $v_6 = (2,2)$. Here $LB = 10$. Let us apply the Algorithm $TwoApprox[d_I = 0, last = V](\mathcal{M})$. First we apply the algorithm for m_1, m_2. As $(m_1, m_2) \notin HV$, we are at line 4 and $\mathcal{S} = (m_2, m_1)$. Then we consider m_3, m_4. The value of p (line 6) is m_1 and as $(m_1, m_3) \notin VH$ and $(m_1, m_4) \in HV$, we get (line 9, case 3) $\mathcal{S} = (m_2, m_3, m_1, m_4)$. We now apply the algorithm with m_5, m_6. The value of p (line 6) is m_4 and as $(m_4, m_5) \notin VH$ and $(m_4, m_6) \notin HV$, we get (line 10, case 4) $\mathcal{S} = (m_2, m_3, m_1, m_4, m_6, m_5)$. The makespan of the algorithm is $LB + 2 = 12 = d(m_1) + 2$ achieved for $s_3 = m_1$.

But, if we apply to this example the Algorithm $TwoApprox[d_I = 0, last = H](\mathcal{M})$, we get a makespan of 10. Indeed $(m_1, m_2) \in VH$ and we get (line 3) $\mathcal{S} = (m_1, m_2)$. Then as $p = m_2$, $(m_2, m_3) \in HV$ and $(m_3, m_4) \notin VH$, we get (line 8, case 2) $\mathcal{S} = (m_1, m_2, m_4, m_3)$. Finally, with $p = m_3$, $(m_3, m_5) \in HV$ and $(m_5, m_6) \in VH$ we get (line 7, case 1) the final sequence $\mathcal{S} = (m_1, m_2, m_4, m_3, m_5, m_6)$ with makespan $10 = LB$.

Consider the example of Figure 6.4(b). The destinations of the messages m'_i ($1 \leq i \leq 6$) are v'_i, which are placed in symmetric positions with respect to the diagonal as v_i in Figure 6.4(a). So $v'_1 = (3,7)$, $v'_2 = (1,7), \ldots$, $v'_6 = (2,2)$. So we can apply the algorithm by exchanging the x and y, V and H. By the Algorithm $TwoApprox[d_I = 0, last = V](\mathcal{M})$, we get $\mathcal{S} = (m'_1, m'_2, m'_4, m'_3, m'_5, m'_6)$ with makespan 10; by the Algorithm $TwoApprox[d_I = 0, last = H](\mathcal{M})$, we get $\mathcal{S} = (m'_2, m'_3, m'_1, m'_4, m'_6, m'_5)$ with makespan 12.

However there are sequences $\mathcal{M}$ such that both Algorithms $TwoApprox[d_I = 0, last = V](\mathcal{M})$ and $TwoApprox[d_I = 0, last = H](\mathcal{M})$ give a makespan $LB + 2$. Consider the example of Figure 6.4(c) with $\mathcal{M} = (m_1, \ldots, m_6, m'_1, \ldots, m'_6)$. The destinations of $m_1, \ldots, m_6$ are obtained from the destination nodes in Figure 6.4(a) by translating them along a vector $(3,3)$, i.e. we move $v_i = (x_i, y_i)$ to $(x_i + 3, y_i + 3)$. So $LB = 16$ and Algorithm $TwoApprox[d_I = 0, last = V](m_1, \ldots, m_6)$ gives the sequence $\mathcal{S}_V = (m_2, m_3, m_1, m_4, m_6, m_5)$ with makespan 18 and Algorithm $TwoApprox[d_I = 0, last = H](m_1, \ldots, m_6)$ gives the sequence $\mathcal{S}_H = (m_1, m_2, m_4, m_3, m_5, m_6)$ with makespan 16. Note that the destinations of $m'_1, \ldots, m'_6$ are in the same configuration as those of Figure 6.4(b). Now, if we run the Algorithm $TwoApprox[d_I = 0, last = V](\mathcal{M})$ on the sequence $\mathcal{M} = (m_1, \ldots, m_6, m'_1, \ldots, m'_6)$, we get as $(m_5, m'_1) \in VH$ and $(m'_1, m'_2) \in HV$, the sequence $\mathcal{S}_V \odot \mathcal{S}'_V = (m_2, m_3, m_1, m_4, m_6, m_5, m'_1, m'_2, m'_4, m'_3, m'_5, m'_6)$ with makespan 18 achieved for $s_3 = m_1$. If we run Algorithm $TwoApprox[d_I = 0, last = H](\mathcal{M})$ on the sequence $\mathcal{M} = (m_1, \ldots, m_{12})$, we get as $(m_6, m'_1) \in HV$ and $(m'_1, m'_2) \notin VH$ the sequence $\mathcal{S}_H \odot \mathcal{S}'_H = (m_1, m_2, m_4, m_3, m_5, m_6, m'_2, m'_3, m'_1, m'_4, m'_6, m'_5)$ with makespan 18 achieved for $s_9 = m'_1$.

However we can find a sequence with a makespan 16 achieving the lower bound with a basic scheme namely $\mathcal{S}^* = (m_1, m_5, m_2, m_4, m_3, m'_1, m_6, m'_2, m'_5, m'_3, m'_4, m'_6)$ with the first message sent horizontally.



(a) (b) (c)

Figure 6.4: Examples for Algorithms $TwoApprox[d_I = 0, last = D](\mathcal{M})$ and $OneApprox[d_I = 0, last = D](\mathcal{M})$

Theorem 19. *Given a basic instance and the set of messages ordered by non-increasing distances from BS, $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ and a direction $D \in \{H, V\}$, Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ computes in linear-time an ordering $\mathcal{S} = (s_1, \cdots, s_M)$ of the messages satisfying the following properties:*

(i) the basic scheme$(\mathcal{S}, last = D)$ broadcasts the messages without collisions;

(ii) $s_1 \in \{m_1, m_2\}$, $s_2 \in \{m_1, m_2, m_3\}$ and $s_i \in \{m_{i-2}, m_{i-1}, m_i, m_{i+1}, m_{i+2}\}$ for any $3 \leq i \leq M-2$ and $s_{M-1} \in \{m_{M-3}, m_{M-2}, m_{M-1}, m_M\}$, $s_M \in \{m_{M-1}, m_M\}$.

Proof. The proof is by induction on M. If $M = 1$, we send m_1 in direction D (line 1). So the theorem is true. If $M = 2$, either $(m_1, m_2) \in \bar{D}D$ and $\mathcal{S} = (m_1, m_2)$ satisfies all properties or $(m_1, m_2) \notin \bar{D}D$ and by Lemma 54 $(m_2, m_1) \in \bar{D}D$ and $\mathcal{S} = (m_2, m_1)$ satisfies all properties .

If $M > 2$, let $\mathcal{O} \odot p = TwoApprox[d_I = 0, last = D](m_1, \cdots, m_{M-2})$ be the sequence computed by the algorithm for $(m_1, m_2, \cdots, m_{M-2})$. By the induction hypothesis, we may assume that $\mathcal{O} \odot p$ satisfies properties (i) and (ii). In particular p is sent in direction D and $p \in \{m_{M-3}, m_{M-2}\}$. Now we will prove that the sequence $S = \{s_1, \ldots, s_M\}$ satisfies properties (i) and (ii). Property (ii) is satisfied in all cases: for $s_i, 1 \leq i \leq M-3$, as it is verified by induction in $\mathcal{O}$; for s_{M-2}, as either $s_{M-2} = p \in \{m_{M-3}, m_{M-2}\}$ or $s_{M-2} = m_{M-1}$; for s_{M-1}, as either $s_{M-1} = p \in \{m_{M-3}, m_{M-2}\}$ or $s_{M-1} = m_{M-1}$ or $s_{M-1} = m_M$ and finally for s_M, as $s_M \in \{m_{M-1}, m_M\}$. For property (i) we consider the four cases of the algorithm (lines 7-10). Obviously, the last message is sent in direction D in all cases. In the following we prove that there are no interferences in any case. For cases 1, 2 and 4, $\mathcal{O} \odot p$ is by induction a scheme that results in no collision.

In case 1, by hypothesis, $(p, m_{M-1}) \in D\bar{D}$ and $(m_{M-1}, m_M) \in \bar{D}D$.

In case 2, since $(p, m_{M-1}) \in D\bar{D}$ and $(m_{M-1}, m_M) \notin \bar{D}D$, Lemma 55 with p, m_{M-1}, m_M in this order implies that $(p, m_M) \in D\bar{D}$. Furthermore, by Lemma 54, $(m_{M-1}, m_M) \notin \bar{D}D$ implies $(m_M, m_{M-1}) \in \bar{D}D$.

For case 4, by Lemma 54 $(p, m_M) \notin \bar{D}D$ implies $(p, m_M) \in D\bar{D}$. Furthermore Lemma 56, applied with p, m_M, m_{M-1} in this order and direction $\bar{D}$, implies $(m_M, m_{M-1}) \in \bar{D}D$

For case 3, $(p, m_M) \in \bar{D}D$; furthermore by Lemma 54, $(p, m_{M-1}) \notin D\bar{D}$ implies $(m_{M-1}, p) \in D\bar{D}$. It remains to verify that if q is the last message of $\mathcal{O}$, $(q, m_{M-1}) \in \bar{D}D$. As $\mathcal{O} \odot p$ is an admissible scheme we have $(q, p) \in \bar{D}D$ and since also $(p, m_{M-1}) \notin D\bar{D}$, by Lemma 55 applied with q, p, m_{M-1} in this order and direction $\bar{D}$, we get $(q, m_{M-1}) \in \bar{D}D$. □

As corollary we get by property (ii) and definition of LB that the basic scheme $(\mathcal{S}, last = D)$ achieves a makespan at most $LB + 2$. We emphasize this result as a Theorem and note that in view of Example 2 it is the best possible for the algorithm.

Theorem 20. *In the basic instance, the basic scheme $(\mathcal{S}, last = D)$ obtained by the Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ achieves a makespan at most $LB + 2$.*

Proof. It is sufficient to consider the arrival time of each message. Because Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ uses a basic scheme, each message follows a shortest path. By Property (ii) of Theorem 19, the message s_1 arrives at its destination at step $d(s_1) \leq d(m_1) \leq LB$ and the message s_2 arrives at step $d(s_2) + 1 \leq d(m_1) + 1 \leq LB + 1$; for any $2 < i \leq M$, the message s_i arrives at its destination at step $d(s_i) + i - 1 \leq d(m_{i-2}) + i - 1 = d(m_{i-2}) + (i-2) - 1 + 2 \leq LB + 2$. □

6.3.5 Makespan can be approximated up to additive constant 1

In this subsection, we show how to improve Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ in the basic instance (open grid with BS in the corner) to achieve makespan at most $LB + 1$. For that we will distinguish two cases according to the value of last term s_M which can be either m_M or m_{M-1}. In the later case, $s_M = m_{M-1}$ we will also maintain another ordered admissible sequence $\mathcal{S}'$ of the $M-1$ messages $(m_1, \cdots, m_{M-1})$ which can be extended in

the induction step when $\mathcal{S}$ cannot be extended. Both sequences $\mathcal{S}$ and $\mathcal{S}'$should satisfy some technical properties (see Theorem 21).

We denote the algorithm as $OneApprox[d_I = 0, last = D](\mathcal{M})$. For an ordered input sequence $\mathcal{M}$ of messages and the direction $D \in \{H,V\}$, it gives as output an ordered sequence $\mathcal{S}$ of the messages such that the basic scheme $(\mathcal{S}, last = D)$ has makespan at most $LB+1$. Algorithm $OneApprox[d_I = 0, last = D](\mathcal{M})$ is depicted in Figure 6.5. As we explain in Remark 9, we can also obtain algorithms with the first message sent in direction D.

Input: $\mathcal{M} = (m_1, \cdots, m_M)$, the set of messages ordered by non-increasing distances from *BS* and the direction $D \in \{V,H\}$ of the last message.
Output: $\mathcal{S} = (s_1, \cdots, s_M)$ an ordered sequence of $\mathcal{M}$ satisfying properties (a) and (b) and, only when $s_M = m_{M-1}$, an ordering $\mathcal{S}' = (s'_1, \cdots, s'_{M-1})$ of the messages $(m_1, \cdots, m_{M-1})$ satisfying properties (a'), (b') and (c') (See in Theorem 21). When $\mathcal{S}'$ is not specified below, it means $\mathcal{S}' = \emptyset$.
begin
1 **Case** $M = 1$**: return** $\mathcal{S} = (m_1)$
2 **Case** $M = 2$**:**
3 **if** $(m_1, m_2) \in \bar{D}D$ **return** $\mathcal{S} = (m_1, m_2)$
4 **else return** $\mathcal{S} = (m_2, m_1)$ and $\mathcal{S}' = (m_1)$
5 **Case** $M > 2$**:**
6 let $\mathcal{O} \odot p = OneApprox[d_I = 0, last = D](m_1, \cdots, m_{M-2})$ and when $p = m_{M-3}$, let $\mathcal{O}'$ be the ordering of $\{m_1, \cdots, m_{M-3}\}$ satisfying (a')(b')(c').
7 **Case 1: if** $(p, m_{M-1}) \in D\bar{D}$ and $(m_{M-1}, m_M) \in \bar{D}D$ **return** $\mathcal{S} = \mathcal{O} \odot (p, m_{M-1}, m_M)$
8 **Case 2: if** $(p, m_{M-1}) \in D\bar{D}$ and $(m_{M-1}, m_M) \notin \bar{D}D$ **return** $\mathcal{S} = \mathcal{O} \odot (p, m_M, m_{M-1})$ and $\mathcal{S}' = \mathcal{O} \odot (p, m_{M-1})$
9 **Case 3: if** $(p, m_{M-1}) \notin D\bar{D}$ and $(m_{M-2}, m_M) \in \bar{D}D$
10 **Case 3.1: if** $p = m_{M-2}$ **return** $\mathcal{S} = \mathcal{O} \odot (m_{M-1}, m_{M-2}, m_M)$
11 **Case 3.2: if** $p = m_{M-3}$ **return** $\mathcal{S} = \mathcal{O}' \odot (m_{M-1}, m_{M-2}, m_M)$
12 **Case 4: if** $(p, m_{M-1}) \notin D\bar{D}$ and $(m_{M-2}, m_M) \notin \bar{D}D$
13 **Case 4.1: if** $p = m_{M-2}$ **return** $\mathcal{S} = \mathcal{O} \odot (m_{M-2}, m_M, m_{M-1})$ and $\mathcal{S}' = \mathcal{O} \odot (m_{M-1}, m_{M-2})$
14 **Case 4.2: if** $p = m_{M-3}$ **return** $\mathcal{S} = \mathcal{O} \odot (m_{M-3}, m_M, m_{M-1})$ and $\mathcal{S}' = \mathcal{O}' \odot (m_{M-1}, m_{M-2})$
end

Figure 6.5: Algorithm $OneApprox[d_I = 0, last = D](\mathcal{M})$

Example 3. *Here, we give examples that illustrate the execution of Algorithm OneApprox. Moreover, we describe instances for which it is not optimal.*

Consider again the Example of Figure 6.4(a) (see Example 2). Let us apply the Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M})$. *First we apply the algorithm for* m_1, m_2*;* $(m_1, m_2) \notin HV$*, we are at line* 4 *and* $\mathcal{S} = (m_2, m_1)$ *and* $\mathcal{S}' = (m_1)$*. Then we consider* m_3, m_4*; the value of p (line 6) is* m_1*; as* $(m_1, m_3) \notin VH$ *and* $(m_2, m_4) \in HV$*, we are in case 3.2 line*

11 ($p = m_{M-3}$). So we get, as $\mathcal{O}' = (m_1)$, $S = (m_1, m_3, m_2, m_4)$. We now apply the algorithm with m_5, m_6; the value of p (line 6) is m_4; as $(m_4, m_5) \notin VH$ and $(m_4, m_6) \notin HV$, we are in case 4.1 line 13. So we get $S = (m_1, m_3, m_2, m_4, m_6, m_5)$. The makespan of the algorithm is $LB + 1 = 11 = d(m_5) + 5$ achieved for $s_6 = m_5$.

But, if we apply to this example the Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M})$, we get a makespan of 10. Indeed $(m_1, m_2) \in HV$ and so the algorithm applied to (m_1, m_2) gives $S = (m_1, m_2)$. Then as $p = m_2$, $(m_2, m_3) \in HV$ and $(m_3, m_4) \notin VH$, we are in case 2 line 8. So we get $S = (m_1, m_2, m_4, m_3)$ and $\mathcal{S}' = (m_1, m_2, m_3)$. Finally, with $p = m_3$, $(m_3, m_5) \in HV$ and $(m_5, m_6) \in VH$ we get (line 7 case 1) the final sequence $S = (m_1, m_2, m_4, m_3, m_5, m_6)$ with makespan $10 = LB$.

However there are sequences $\mathcal{M}$ such that both Algorithms $OneApprox[d_I = 0, last = V](\mathcal{M})$ and
$OneApprox[d_I = 0, last = H](\mathcal{M})$ give a makespan $LB + 1$. Consider the example of Figure 6.4(c). Like in Example 2, $LB = 16$; furthermore, for the messages $m_1, \ldots, m_6$ Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M})$ gives the sequence $(m_1, m_3, m_2, m_4, m_6, m_5)$ denoted by S_V with makespan 17 and Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M})$ gives the sequence $(m_1, m_2, m_4, m_3, m_5, m_6)$ denoted by $S_H$with makespan 16. For the messages $m'_1, \ldots, m'_6$, we get (similarly as in Example 2) by applying Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M})$ the sequence $S'_V = (m'_1, m'_2, m'_4, m'_3, m'_5, m'_6)$ with makespan 10 and by applying the Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M})$ the sequence $S'_H = (m'_1, m'_3, m'_2, m'_4, m'_6, m'_5)$ with makespan 11 achieved for $s'_6 = m'_5$. Now if we run the Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M})$ on the global sequence $\mathcal{M} = (m_1, \ldots, m_6, m'_1, \ldots, m'_6)$, we get as $(m_5, m'_1) \in VH$ and $(m'_1, m'_2) \in HV$, the sequence $S_V \odot S'_V = (m_1, m_3, m_2, m_4, m_6, m_5, m'_1, m'_2, m'_4, m'_3, m'_5, m'_6)$ with makespan 17 achieved for $s_6 = m_5$. If we run Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M})$ on the global sequence $\mathcal{M} = (m_1, \ldots, m_6, m'_1, \ldots, m'_6)$, we get as $(m_6, m'_1) \in HV$ and $(m'_1, m'_2) \notin VH$, the sequence $S_H \odot S'_H = (m_1, m_2, m_4, m_3, m_5, m_6, m'_1, m'_3, m'_2, m'_4, m'_6, m'_5)$ with makespan 17 achieved for $s_{12} = m'_5$.

However, we know that the sequence S^* (defined in Example 2) achieves a makespan 16.

Theorem 21. *Given a basic instance and the set of messages ordered by non-increasing distances from BS, $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ and a direction $D \in \{H, V\}$, Algorithm $OneApprox[d_I = 0, last = D](\mathcal{M})$ computes in linear-time an ordering $\mathcal{S} = (s_1, \cdots, s_M)$ of the messages satisfying the following properties:*

(a) the basic scheme $(\mathcal{S}, last = D)$ broadcasts the messages without collisions;

(b) $s_1 \in \{m_1, m_2\}$ and $s_i \in \{m_{i-1}, m_i, m_{i+1}\}$ for any $1 < i \leq M-1$, and $s_M \in \{m_{M-1}, m_M\}$.

When $s_M = m_{M-1}$, it also computes an ordering $\mathcal{S}' = (s'_1, \cdots, s'_{M-1})$ of the messages $(m_1, \cdots, m_{M-1})$ satisfying properties (a')-(c').

(a') the scheme$(\mathcal{S}', last = \bar{D})$ broadcasts the messages without collisions;

(b') $s'_1 \in \{m_1, m_2\}$, *and* $s'_i \in \{m_{i-1}, m_i, m_{i+1}\}$ *for any* $1 < i \leq M-2$, *and* $s'_{M-1} \in \{m_{M-2}, m_{M-1}\}$.

(c') $(s'_{M-1}, m_M) \notin \bar{D}D$ *and if* $s'_{M-1} = m_{M-2}$, $(m_{M-2}, m_{M-1}) \notin D\bar{D}$

Proof. The proof is by induction. If $M = 1$, the result is correct as we send $m_1$in direction D (line 1). If $M = 2$, either $(m_1, m_2) \in \bar{D}D$ and $\mathcal{S} = (m_1, m_2)$ satisfies properties (a) and (b) or $(m_1, m_2) \notin \bar{D}D$ and by Lemma 54 $(m_2, m_1) \in \bar{D}D$ and $\mathcal{S} = (m_2, m_1)$ satisfies properties (a) and (b) and $\mathcal{S}' = (m_1)$ satisfies all properties (a'), (b') and (c').

Now, let $M > 2$ and let $\mathcal{O} \odot p = OneApprox[d_I = 0, last = D](m_1, \cdots, m_{M-2})$ be the sequence computed by the algorithm for $(m_1, m_2, \cdots, m_{M-2})$. By the induction hypothesis, we may assume that $\mathcal{O} \odot p$ satisfies properties (a) and (b). In particular p is sent in direction D and $p \in \{m_{M-3}, m_{M-2}\}$. We have also that, if $p = m_{M-3}$, $\mathcal{O}'$ satisfies properties (a'), (b') and (c').

Property (b) is also satisfied for $s_i, 1 \leq i \leq M-3$ as it is verified by induction either in $\mathcal{O}$ or in case 3.2 in $\mathcal{O}'$. Furthermore, either $s_{M-2} = p \in \{m_{M-3}, m_{M-2}\}$ or $s_{M-2} = m_{M-1}$ in case 3. Similarly, $s_{M-1} \in \{m_{M-2}, m_{M-1}, m_M\}$ and $s_M \in \{m_{M-1}, m_M\}$. Hence, Property (b) is satisfied. Property (b') is also satisfied for $s'_i, 1 \leq i \leq M-3$, as it is verified by induction in $\mathcal{O}$ or for case 4.2 in $\mathcal{O}'$. Furthermore $s'_{M-2} \in \{m_{M-3}, m_{M-2}, m_{M-1}\}$ and $s'_{M-1} \in \{m_{M-2}, m_{M-1}\}$. Hence, Property (b') is satisfied.

Now let us prove that $\mathcal{S}$ satisfies property (a) and $\mathcal{S}'$ properties (a') and (c') in the six cases of the algorithm (lines 7-14). Obviously the last message in $\mathcal{S}$ (resp. $\mathcal{S}'$) is sent in direction D (resp. $\bar{D}$).

In cases 1, 2, 3.1, 4.1 the hypothesis and sequence $\mathcal{S}$ are exactly the same as that given by Algorithm
$TwoApprox[d_I = 0, last = D](\mathcal{M})$. Therefore, by the proof of Theorem 19, $\mathcal{S}$ satisfies property (a) and so the proof is complete for cases 1 and 3.1 as there are no sequences $\mathcal{S}'$.

In case 2, $\mathcal{S}'$ satisfies (a') as by hypothesis (line 8) $(p, m_{M-1}) \in D\bar{D}$. Property (c') is also satisfied as $s'_{M-1} = m_{M-1}$ and by hypothesis (line 8) $(m_{M-1}, m_M) \notin \bar{D}D$.

In case 4.1 ($p = m_{M-2}$), let q be the last element of $\mathcal{O}$; $(q, m_{M-2}) \in \bar{D}D$ as $\mathcal{O} \odot p$ is admissible. By hypothesis (line 12), $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ and then by Lemma 55 applied with q, m_{M-2}, m_{M-1} in this order, we get $(q, m_{M-1}) \in \bar{D}D$; furthermore, by Lemma 54 , $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ implies $(m_{M-1}, m_{M-2}) \in D\bar{D}$. So, $\mathcal{S}'$ satisfies Property (a'). Finally $s'_{M-1} = m_{M-2}$ and by hypothesis (line 12) $(m_{M-2}, m_M) \notin \bar{D}D$ and $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ and therefore $\mathcal{S}'$ satisfies property (c').

The following claims will be useful to conclude the proof in cases 3.2 and 4.2. In these cases $p = m_{M-3}$ and let p' be the last element of $\mathcal{O}'$. By induction on $\mathcal{O}'$, and by property (b'), $p' \in \{m_{M-4}, m_{M-3}\}$.

Claim 4. *: In cases 3.2 and 4.2,* $(m_{M-2}, m_{M-1}) \notin D\bar{D}$

Proof. To write a convincing proof, we use coordinates and the expression of Fact 6 in terms of coordinates (see Remark 8). We use $dest(m_{M-i}) = (x_{M-i}, y_{M-i})$. Let us suppose $D = V$ (the claim can be proved for $D = H$ by exchanging H and V and exchanging x and y).

By hypothesis (lines 9 and 12) $(m_{M-3}, m_{M-1}) \notin VH$.

- If $p' = m_{M-3}$, by induction hypothesis (c') applied to $\mathcal{O}'$, we have $(p', m_{M-2}) \notin HV$. Then $(m_{M-3}, m_{M-1}) \notin VH$ and $(m_{M-3}, m_{M-2}) \notin HV$ imply by Fact 6: $\{x_{M-1} < x_{M-3}$ and $y_{M-1} \geq y_{M-3}\}$ and $\{x_{M-2} \geq x_{M-3}$ and $y_{M-2} < y_{M-3}\}$.

  So we have $x_{M-1} < x_{M-3} \leq x_{M-2}$ implying $x_{M-1} < x_{M-2}$ and $y_{M-1} \geq y_{M-3} > y_{M-2}$ implying $y_{M-1} > y_{M-2}$. These conditions imply by Fact 6 that $(m_{M-2}, m_{M-1}) \notin VH$.

- If $p' = m_{M-4}$, by induction hypothesis (c') applied to $\mathcal{O}'$, we have $(p', m_{M-2}) \notin HV$ and $(m_{M-4}, m_{M-3}) \notin VH$. So $(m_{M-3}, m_{M-1}) \notin VH$, $(m_{M-4}, m_{M-2}) \notin HV$ and $(m_{M-4}, m_{M-3}) \notin VH$ imply respectively by Fact 6: $\{x_{M-1} < x_{M-3}$ and $y_{M-1} \geq y_{M-3}\}$; $\{x_{M-2} \geq x_{M-4}$ and $y_{M-2} < y_{M-4}\}$ and $\{x_{M-3} < x_{M-4}$ and $y_{M-3} \geq y_{M-4}\}$.

  So we have $x_{M-1} < x_{M-3} < x_{M-4} \leq x_{M-2}$ implying $x_{M-1} < x_{M-2}$ and $y_{M-1} \geq y_{M-3} \geq y_{M-4} > y_{M-2}$ implying $y_{M-1} > y_{M-2}$. These conditions imply by Fact 6 that $(m_{M-2}, m_{M-1}) \notin VH$.

□

Claim 5. *: In cases 3.2 and 4.2, $(p', m_{M-1}) \in \bar{D}D$.*

Proof. If $p' = m_{M-3}$ by hypothesis lines 9 and 12 $(m_{M-3}, m_{M-1}) \notin D\bar{D}$ and by Lemma 54 $(m_{M-3}, m_{M-1}) \in \bar{D}D$. If $p' = m_{M-4}$, by induction hypothesis (c') applied to $\mathcal{O}'$, $(m_{M-4}, m_{M-3}) \notin D\bar{D}$ and so by Lemma 54 $(m_{M-4}, m_{M-3}) \in \bar{D}D$; furthermore by hypothesis $(m_{M-3}, m_{M-1}) \notin D\bar{D}$ and so by Lemma 55 applied with $m_{M-4}, m_{M-3}, m_{M-1}$ in this order, we get $(m_{M-4}, m_{M-1}) \in \bar{D}D$.

□

In case 3.2, by hypothesis (line 9) $(m_{M-2}, m_M) \in \bar{D}D$; by the claim 1 $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ and so by Lemma 54 $(m_{M-1}, m_{M-2}) \in D\bar{D}$; and by claim 2, $(p', m_{M-1}) \in \bar{D}D$. So the theorem is proved in case 3.2.

Finally it remains to deal with the case 4.2. Let us first prove that $\mathcal{S}$ satisfies (a). By hypothesis line 12 $(m_{M-2}, m_M) \notin \bar{D}D$ and by the claim $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ and so by Lemma 56 applied with m_{M-2}, m_M, m_{M-1} in this order we get $(m_M, m_{M-1}) \in \bar{D}D$. We claim that $(m_{M-3}, m_{M-2}) \in D\bar{D}$; indeed, if $p' = m_{M-3}$, by induction hypothesis (c') applied to $\mathcal{O}'$, we have $(m_{M-3}, m_{M-2}) \notin \bar{D}D$ and so $(m_{M-3}, m_{M-2}) \in D\bar{D}$. If $p' = m_{M-4}$, by induction hypothesis (c') applied to $\mathcal{O}'$, we have $(m_{M-4}, m_{M-2}) \notin \bar{D}D$ and $(m_{M-4}, m_{M-3}) \notin D\bar{D}$ and so by Lemma 56 applied with $m_{M-4}, m_{M-3}, m_{M-2}$ in this order we get $(m_{M-3}, m_{M-2}) \in D\bar{D}$. Now the property $(m_{M-3}, m_{M-2}) \in D\bar{D}$ combined with the hypothesis line 12 $(m_{M-2}, m_M) \notin \bar{D}D$ gives by Lemma 55 applied with m_{M-3}, m_{M-2}, m_M in this order $(m_{M-3}, m_M) \in D\bar{D}$.

Finally, by claim 1, $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ and so by Lemma 54 $(m_{M-1}, m_{M-2}) \in D\bar{D}$. By claim 2, $(p', m_{M-1}) \in \bar{D}D$ and so $\mathcal{S}'$ satisfies Property (a'). $\mathcal{S}'$ satisfies also Property (c') as $(m_{M-2}, m_M) \notin \bar{D}D$ by hypothesis and $(m_{M-2}, m_{M-1}) \notin D\bar{D}$ by claim 1.

□

As corollary we get by property (b) and definition of LB that the basic scheme $(\mathcal{S}, last = D)$ achieves a makespan at most $LB + 1$. We emphasize this result as a Theorem and note

that in view of Example 3 it is the best possible for the algorithm. The proof is similar to that Theorem 20.

Theorem 22. *In the basic instance, the basic scheme $(\mathcal{S}, last = D)$ obtained by the Algorithm $OneApprox[d_I = 0, last = D](\mathcal{M})$ achieves a makespan at most $LB + 1$.*

As we have seen in Example 3, Algorithms $OneApprox[d_I = 0, last = V](\mathcal{M})$ and $OneApprox[d_I = 0, last = H](\mathcal{M})$ are not always optimal since there are instances for which the optimal makespan equals LB while our algorithms only achieves $LB + 1$. However there are other cases where Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M})$ or Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M})$ can be used to obtain an optimal makespan LB. The next theorem might appear as specific, but it includes the case where each node in a finite grid receives exactly one message (case considered in many papers in the literature, such as in [BP12] for the grid when buffering is allowed).

Theorem 23. *Let $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ be an ordered sequence of messages (i.e., by decreasing distance), if the bound $LB = \max_{i \leq M} d(m_i) + i - 1$ is reached for an unique value of i, then we can design an algorithm with optimal makespan $= LB$.*

Proof. Let k be the value for which LB is achieved that is $d(m_k) + k - 1 = LB$ and $d(m_i) + i - 1 < LB$ for $i \neq k$. We divide $\mathcal{M} = (m_1, \cdots, m_M)$ into two ordered subsequences $\mathcal{M}_k = (m_1, \ldots, m_k)$ and $\mathcal{M}'_k = (m_{k+1}, \ldots, m_M)$. So $|\mathcal{M}_k| = k$ and $|\mathcal{M}'_k| = M - k$. Let $\mathcal{S}_V$ (resp., $\mathcal{S}_H$) be the sequence obtained by applying Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M}_k)$ (resp., Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M}_k)$) to the sequence $\mathcal{M}_k$. The makespan is equal to LB; indeed if the sequence is $(s_1, \ldots, s_k)$, then the makespan is $\max_{i \leq k} d(s_i) + i - 1$. But we have $s_i \in \{m_{i-1}, m_i, m_{i+1}\}$ for any $i \leq k - 1$, and so $d(s_i) + i - 1 \leq d(m_{i-1}) + (i - 1) \leq LB$ (as $d(m_{i-1}) + (i - 1) - 1 < LB$); we also have $s_k \in \{m_{k-1}, m_k\}$ and so either $d(s_k) + (k - 1) = d(m_{k-1}) + (k - 1) \leq LB$ or $d(s_k) + (k - 1) = d(m_k) + k - 1 = LB$.

Suppose $k > 1$, then the destination of m_{k-1} is at the same distance of that of m_k; indeed if $d(m_{k-1}) > d(m_k)$, then $d(m_{k-1}) + k - 2 \geq d(m_k) + k - 1 = LB$ and LB will also be achieved for $k - 1$ contradicting the hypothesis. Consider the set D_k of all the messages with destinations at the same distance as that of m_k (so if $k > 1$ $|D_k| \geq 2$) and let m_u (resp.,m_ℓ) be the uppermost message (resp., lowest message) of D_k, that is the message in D_k with destination the node with the highest y (resp., the lowest y); (in case there are many such messages with this property, i.e. they have the same destination node, we choose one of them).

We claim that there exists a basic scheme for $\mathcal{M}_k$, such that if the last message is sent vertically (resp., horizontally) it is m_u (resp. m_ℓ). Indeed, suppose we want the last message sent vertically to be m_u it suffices to order the messages in $\mathcal{M}_k$ such that the last one $m_k = m_u$; then if we apply Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M}_k)$ we get a sequence where $s_k \in \{m_{k-1}, m_k\}$. Either $s_k = m_k = m_u$ and we are done or $s_k = m_{k-1}$ and $s_{k-1} = m_u$; but in that case $(s_{k-1}, s_k) \in HV$ implies, by Fact 6, that $x_{k-1} < x_u$ or $y_{k-1} \geq y_u$, where (x_u, y_u) and (x_{k-1}, y_{k-1}) are the destinations of m_u and m_{k-1}. But $m_u, m_{k-1} \in D_k$ and m_u being the uppermost vertex, $y_{k-1} \leq y_u$ and $x_{k-1} \geq x_u$. Therefore, s_{k-1} and s_k have the same

destination. So, we can interchange them. Similarly using Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M}_k)$ we can obtain an HV-scheme denoted $\mathcal{S}_H$ with the last message sent horizontally being m_ℓ.

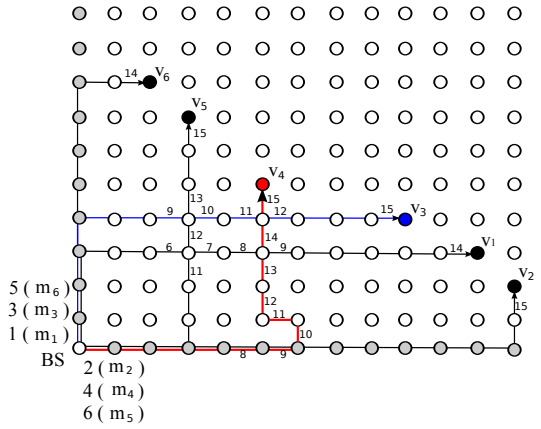
If k=1, $\mathcal{M}_k$ is reduced to one message m_1 and the claims are satisfied with $m_u = m_\ell = m_1$ and $\mathcal{S}_V = \mathcal{S}_H = m_1$.

Now, we consider the sequence $\mathcal{M}'_k$; the lower bound is $LB' = \max_{k<i\leq M} d(m_i) + i - k - 1 < LB - k$ as LB is not achieved for any $i \neq k$. Let $\mathcal{S}'_H$ be the sequence obtained by applying Algorithm $OneApprox[d_I = 0, first = H](\mathcal{M}'_k)$ with the first element of $\mathcal{S}'_H$ sent horizontally and let s'_h be this first element. (We obtain this algorithm from Algorithm $OneApprox[d_I = 0, last = V](\mathcal{M}'_k)$ if $|M'_k| = M - k$ is even or Algorithm $OneApprox[d_I = 0, last = H](\mathcal{M}'_k)$ if $|M'_k|$ is odd). Similarly, let $\mathcal{S}'_V$ be the sequence obtained by applying Algorithm $OneApprox[d_I = 0, first = V](\mathcal{M}'_k)$ with the first element of $\mathcal{S}'_V$ sent vertically and let s'_v be this first element. In all the cases the makespan is at most $LB' + 1 \leq LB - k$.

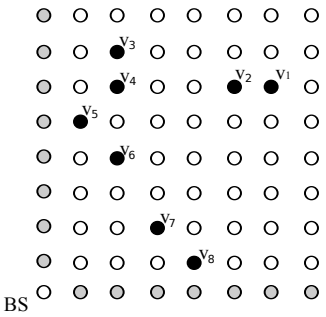
Finally, we consider the concatenation of the sequences $\mathcal{S}_V \odot \mathcal{S}'_H$ and $\mathcal{S}_H \odot \mathcal{S}'_V$. We claim that one of these two sequences has no interferences. If the claim is true, then the theorem is proved as the makespan will be LB for the first k messages and $LB' + 1 + k \leq LB$ for the last $M - k$ messages. In what follows, let as usual (x_u, y_u), (x_l, y_l), (x'_h, y'_h) and (x'_v, y'_v) denote respectively the destinations of messages m_u, m_l, s'_h and s'_v. Now, suppose the claim is not true, that is $(m_u, s'_h) \notin VH$ and $(m_\ell, s'_v) \notin HV$. That implies by Fact 6 that $x'_h < x_u$ and $y'_h \geq y_u$ and $x'_v \geq x_\ell$ and $y'_v < y_\ell$. But we choose the destination of m_u (resp.,m_ℓ) to be the uppermost one (resp., the lowest one) in D_k. So, $x_u \leq x_l$ and $y_u \geq y_l$. Therefore $x'_h < x'_v$ and $y'_h > y'_v$ which imply first that $s'_h \neq s'_v$ and by Fact 6 that $(s'_v, s'_h) \notin VH$ and $(s'_h, s'_v) \notin HV$.

Note that, by the property of Algorithm $OneApprox[d_I = 0, last = D](\mathcal{M})$, $s'_h \in \{m_{k+1}, m_{k+2}\}$ and $s'_v \in \{m_{k+1}, m_{k+2}\}$; thus, as they are different, one of s'_h, s'_v is m_{k+1} and the other m_{k+2}. Suppose that $s'_h = m_{k+1}$ and $s'_v = m_{k+2}$; then in the sequence $\mathcal{S}'_V$ the first message is $s'_v = m_{k+2}$ and from property (c) in Theorem 21, the second message is necessarily $m_{k+1} = s'_h$, but that implies $(s'_v, s'_h) \in VH$ a contradiction. The case $s'_h = m_{k+2}$ and $s'_v = m_{k+1}$ implies similarly in the sequence $\mathcal{S}'_H$ that $(s'_h, s'_v) \in HV$, a contradiction. So the claim and the theorem are proved. □

Example 4. *As mentioned above, Algorithm OneApprox is not always optimal. The design of a polynomial-time optimal algorithm seems challenging because of some reasons that we discuss now. First, the first example below shows that there are open-grid instances for which any broadcast scheme using shortest paths in not optimal (a general grid with such property was already given in Example 1). In this example described in Figure 6.6(a), we have* 6 *messages* m_i *(*$1 \leq i \leq 6$*) with destinations at distance* d *for* m_1 *and* m_2*,* $d - 1$ *for* m_3 *and* $d - 4$ *for* m_4, m_5, m_6*. Here* $LB = d + 1$*, achieved for* m_2*,* m_3 *and* m_6*. In the Figure 6.6(a),* $d = 14$*,* $v_1 = (11,3)$*,* $v_2 = (12,2)$*,* $v_3 = (9,4)$*,* $v_4 = (5,5)$*,* $v_5 = (3,7)$ *and* $v_6 = (2,8)$ *and* $LB = 15$*. If we apply* $OneApprox[d_I = 0, last = V](\mathcal{M})$ *we get the sequence* $(m_1, m_3, m_2, m_5, m_4, m_6)$ *with a makespan* 16 *attained for* $s_3 = m_2$*. If we apply*



(a) Example for optimal schedule with not shortest directed path.



(b) Example for optimal schedule sending the m_6 at step 3.

Figure 6.6: Examples for optimal schedules are difficult to obtain.

OneApprox$[d_I = 0, last = H](\mathcal{M})$ *we get the sequence* $(m_1, m_2, m_4, m_3, m_6, m_5)$ *also with a makespan* 16 *attained for* $s_4 = m_3$. *Consider any algorithm where the messages are sent via shortest directed paths. If the makespan is LB then* m_1 *and* m_2 *should be sent in the first two steps and to avoid interferences the source should send* m_1 *via* $(0,1)$ *and* m_2 *via* $(1,0)$. m_3 *should be sent at step* 3. *If* m_2 *was sent at step* 1 *and so* m_1 *at step* 2, *then* m_3 *should be sent at step* 3 *via* $(1,0)$ *and will interfere with* m_1. *Therefore, the only possibility is to send* m_1 *at step* 1 *via* $(0,1)$, m_2 *at step* 2 *via* $(1,0)$ *and* m_3 *at step* 3 *via* $(0,1)$. *But then at step* 4, *we cannot send any of* m_4, m_5, m_6 *without interference. So the source does no sending at step* 4, *but the last sent message will be sent at step* 7 *and the makespan will be* $d+2 = LB+1$. *However there exists a tricky schedule with makespan LB, but not with shortest directed paths routing. We sent* m_1 *vertically,* m_2 *horizontally,* m_3 *vertically but* m_4 *with a detour to introduce a delay of* 2. *More precisely, if* $v_4 = (x_4, y_4)$, *we send* m_4 *horizontally till* $(x_4+1, 0)$, *then to* $(x_4+1, 1)$ *and* $(x_4, 1)$ *(the detour) and then vertically till* (x_4, y_4). *Finally we send* m_6 *vertically at step* 5 *and* m_5 *horizontally at step* 6. m_4 *has been delayed by two but the message arrives at time LB and there is no interference between the messages.*

Secondly, even if we restrict ourselves to use shortest paths, the computation of an optimal schedule seems difficult. Indeed, the second example below illustrates the fact that optimal schedule may be very different compared to the non-increasing distance schedule. The example is decribed in Figure 6.6(b). We have 8 *messages* m_i $(1 \leq i \leq 8)$ *with destinations at* $v_1 = (6,6)$, $v_2 = (5,6)$, $v_3 = (2,7)$, $v_4 = (2,6)$, $v_5 = (1,5)$, $v_6 = (2,4)$, $v_7 = (3,2)$ *and* $v_8 = (4,1)$. *Here* $LB = 12$, *achieved for* m_1, m_2 *and* m_8. *We will prove that there is a unique sequence of messages reaching the bound LB which is the ordered sequence* $(m_1, m_2, m_6, m_3, m_4, m_5, m_8, m_7)$ *with the first message sent horizontally. Indeed to reach the makespan LB,* m_1 *and* m_2 *have to be sent first and second because their distances are 12 and 11 and in order they do not interfere* m_1 *has to be sent horizontally and* m_2 *verti-*

cally.The next message to be sent cannot be m_3 nor m_4 as they will interfere with m_2. If the third message sent is m_i for some $i \in 5,7,8$, then the fourth and fifth messages have to be m_3 vertically then m_4 horizontally since their distances are 9 and 8. Now only message m_5 can be sent vertically at step 6, otherwise there will an interference with m_4. Then message m_6 has to be sent horizontally at step 7 since its distance is 6. But then the last message m_7 or m_8 (the one not sent at the third step) can not be sent vertically as it will interfere with m_6. So the only possibility consists in sending m_6 at the third step and then the ordered sequence is forced.

In the last example, some specific message (m_6) has to be chosen to be sent early (while being close to *BS* compared with other messages) to achieve the optimal solution. Deciding of such "critical" message seems to be not easy. Hence it shows that the complexity of determining the value of the minimum makespan might be a difficult problem (even when considering only shortest path schedules).

6.4 Case $d_I = 0$; general grid, and BS in the corner

We will see in this section that, by generalizing the notion of basic scheme, Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ also achieves a makespan at most $LB+2$ in the case of a general grid, that is when the destinations of the messages can be on one or both axes and with BS in the corner. First we have to generalize the notions of horizontal sendings for a destination node on Y-axis and vertical sendings for a destination node on the X-axis. However the proofs of the basic lemmata are more complicated as Lemma 55 is not fully valid in this case. Furthermore, we cannot present the conditions only in simple terms like in Fact 6 and so to be precise we need to use coordinates.

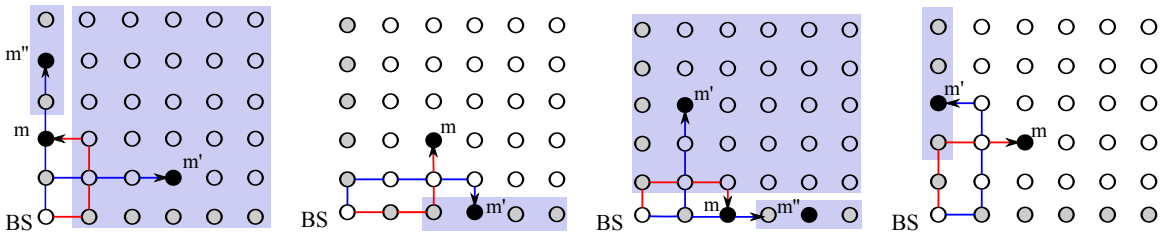
The following definitions are illustrated on Figure 6.7. We will say that a message is sent "horizontally to reach the Y axis", denoted by H_Y*-sending*, if the destination of m is on the Y axis, i.e., $dest(m) = (0,y)$, and the message is sent first horizontally from BS to (1,0) then it follows the vertical directed path from $(1,0)$ till $(1,y)$ and finally the horizontal arc $((1,y),(0,y))$. For instance, an H_Y-sending of message m is illustrated in Figure 6.7(a) and of message m' in Figure 6.7(d).

Similarly a message is sent "vertically to reach the X axis", denoted by V_X*-sending*, if the destination of m is on the X axis, i.e., $dest(m) = (x,0)$, and the message is sent first vertically from BS to (0,1) then it follows the horizontal directed path from $(0,1)$ till $(x,1)$ and finally the vertical arc $((x,1),(x,0))$. For instance, an V_X-sending of message m' is illustrated in Figure 6.7(b) and of message m in Figure 6.7(c).

Notations. Definition 1 of basic scheme in Section 6.3.1 is generalized by allowing H_Y (resp., V_X)-sendings as horizontal (resp., vertical) sendings. For emphasis, we call it *modified basic scheme*. We will also generalize the notation HV (resp., VH) by including H_Y (resp., V_X)-sendings.

Note that we cannot have an H_Y-sending followed by a V_X-sending (or a V_X-sending followed by an H_Y-sending) as there will be interference in $(1,1)$.

Fact 7. *Let $dest(m) = (x,y), dest(m') = (x',y')$ and suppose at least one of $dest(m)$ and $dest(m')$ is on an axis. Then*



(a) $(m,m') \notin HV$ Case 7.1; $(m,m'') \notin HV$ Case 7.2 (b) $(m,m') \notin HV$ Case 7.3 (c) $(m,m') \notin VH$ Case 8.1; $(m,m'') \notin VH$ Case 8.2 (d) $(m,m') \notin VH$ Case 8.3

Figure 6.7: Cases of interferences with destinations on the axis.

- $(m,m') \notin HV$ *if and only if we are in one of the following cases, see in Fig. 6.7(a) and 6.7(b)*

  7.1: $x = 0$ *and* $x' > 0$

  7.2: $x = 0$, $x' = 0$ *and* $y' > y$

  7.3: $x > 0$, $y' = 0$, $x \leq x'$ *and* $y \geq 2$

  or equivalently

- $(m,m') \in HV$ *if and only if we are in one of the following cases*

  7.4: $y = 0$

  7.5: $x = 0$, $x' = 0$ *and* $y' \leq y$

  7.6: $x > 0, y > 0$, $x' = 0$

  7.7: $x > 0, y > 0$, $y' = 0$, *and either* $y = 1$ *or* $x' < x$

Proof. First suppose dest(m) is on one of the axis. If, $y = 0$ there is no interference (7.4). If $x = 0$ and $y' > y$ message m arrives at its destination $(0,y)$ at step $y+2$, but message m' leaves $(0,y)$ at step $y+2$ and so they interfere (7.2 and 7.1 with $y' > y$). If $x = 0$ and $y' \leq y$, either $x' = 0$ and the directed paths followed by the messages do not cross (7.5), or $x' > 0$, but then message m leaves $(1,y')$ at step $y'+2$, while message m' arrives at $(1,y')$ at step $y'+2$ and so they interfere (7.1 with $y' \leq y$).

Suppose now that dest(m) is not on one of the axis, that is $x > 0$ and $y > 0$. If $x' = 0$, the directed paths followed by the messages do not cross (7.6). If $y' = 0$, then either $x' < x$ and the messages do not interfere (7.7) or $x' \geq x$, and the directed paths cross at $(x,1)$ and there either $y = 1$ and the messages do not interfere (7.7) or $y \geq 2$, but then message m leaves $(x,1)$ at step $x+2$, while message m' arrives at $(x,1)$ at step $x+2$ and so they interfere (7.3). □

Fact 8. *Let* $dest(m) = (x,y), dest(m') = (x',y')$ *and suppose at least one of* $dest(m)$ *and* $dest(m')$ *is on an axis. Then*

- $(m,m') \notin VH$ *if and only if we are in one of the following cases, see in Fig. 6.7(c) and 6.7(d)*

8.1: $y = 0$ *and* $y' > 0$

8.2: $y = 0$, $y' = 0$ *and* $x' > x$

8.3: $y > 0$, $x' = 0$, $y \leq y'$ *and* $x \geq 2$

or equivalently

- $(m,m') \in VH$ *if and only if we are in one of the following cases*

8.4: $x = 0$

8.5: $y = 0$, $y' = 0$ *and* $x' \leq x$

8.6: $x > 0, y > 0$, $y' = 0$

8.7: $x > 0, y > 0$, $x' = 0$, *and either* $x = 1$ *or* $y' < y$

Lemma 57. *If* $(m,m') \notin D\bar{D}$, *then* $(m,m') \in \bar{D}D$ *and* $(m',m) \in D\bar{D}$.

Proof. We prove that if $(m,m') \notin HV$ (case $D = H$), then $(m,m') \in VH$ in the following. Other results are proved similarly. If none of the destinations of m and m' are on the axis, the result holds by Lemma 54. If at least one destination is on an axis, suppose that $(m,m') \notin HV$. If conditions of Fact 7.1 or 7.2 are satisfied, then $x = 0$ but then by Fact 8.4 $(m,m') \in VH$. If condition of Fact 7.3 is satisfied , so $x > 0$, $y' = 0$ and $y \geq 2$ which implies by Fact 8.6 that $(m,m') \in VH$.

□

However Lemma 55 is no more valid in its full generality.

Lemma 58. *Let* $dest(m) = (x,y)$, $dest(m') = (x',y')$ *and* $dest(m'') = (x'',y'')$.
If $(m,m') \in D\bar{D}$ *and* $(m',m'') \notin \bar{D}D$, *then* $(m,m'') \in D\bar{D}$ *except if:*

- *Case* $D = H$: $y' = 0$ *(V_X-sending is used for* m'*), and* $y \geq max(2, y''+1)$, *and* $0 < x' < x \leq x''$.
- *Case* $D = V$: $x' = 0$ *(H_Y-sending is used for* m'*), and* $x \geq max(2, x''+1)$, *and* $0 < y' < y \leq y''$.

Proof. Let us prove the case $D = H$. If none of the destinations of m, m', m'' are on an axis the result holds by Lemma 55. If $y = 0$, then $(m,m'') \in HV$ by Fact 7.4. By Fact 8, $(m',m'') \notin VH$ implies $x' > 0$. If $x = 0$, then by Fact 7.5, $(m,m') \in HV$ implies $x' = 0$ a contradiction with the preceding assertion. Therefore $x > 0$ and dest(m) is not on an axis. If $x'' = 0$, then by Fact 7.6 $(m,m'') \in HV$. If $y' > 0$, then $(m',m'') \notin VH$ implies $x'' = 0$ by Fact 8.3, where we already know that by Fact 7.6 $(m,m'') \in HV$. So $y' = 0$, $x > 0$, $y > 0$ and by Fact 7.7 $(m,m') \in HV$ implies that either $y = 1$ or $x' < x$.

If $y'' = 0$, by Fact 7.3, $(m,m'') \notin HV$ if and only if $y \geq 2$ and $x \leq x''$. If $y'' > 0$, none of the destinations of m and m'' are on the axis and so by Fact 6, $(m,m'') \notin HV$, if and only if $x'' \geq x$ and $y'' < y$. So again $y \geq 2$ and $x \leq x''$. In summary $(m,m'') \notin HV$, if and only if $y \geq 2$ and when $y'' > 0$, $y > y''$ and $0 < x' < x \leq x''$

The case $D = V$ is obtained similarly. □

We give the following useful corollary for the proof of the next theorem.

Corollary 12. *If $d(m') \geq d(m'')$ then: If $(m,m') \in D\bar{D}$ and $(m',m'') \notin \bar{D}D$, then $(m,m'') \in D\bar{D}$.*

We now show that:

Lemma 59. *Lemma 56 is still valid in general grid.*

Proof. We prove it for $D = H$. The case $D = V$ can be obtained similarly.

If none of the destinations of m,m',m'' are on an axis the result holds by Lemma 56. Suppose first dest(m'') is on an axis; by Fact 8 $(m,m'') \notin VH$ implies $x > 0$. If furthermore dest(m) or dest(m') are on an axis, by Fact 7.3 $(m,m') \notin HV$ implies $y' = 0$ and so by Fact 7.4 $(m',m'') \in HV$. Otherwise if none of dest(m) and dest(m') are on an axis, $y > 0$ and by Fact 8.3 $(m,m'') \notin VH$ implies $x'' = 0$, and with $x' > 0$ and $y' > 0$ Fact 7.6 implies $(m',m'') \in HV$.

If dest(m'') is not on an axis, then one of dest(m) and dest(m') is on an axis and $(m,m') \notin HV$ implies $y > 0$. We cannot have $x = 0$ otherwise it contradicts $(m,m'') \notin VH$. If $x > 0$, then by Fact 7.3 $(m,m') \notin HV$ implies $y' = 0$, but then Fact 7.4 implies $(m',m'') \in HV$.

□

Theorem 24. *Let $d_I = 0$, and BS be in the corner of the general grid. Given the set of messages ordered by non-increasing distances from BS, $\mathcal{M} = (m_1,m_2,\cdots,m_M)$ and a direction D, Algorithm TwoApprox$[d_I = 0, last = D](\mathcal{M})$ computes in linear-time an ordering $\mathcal{S}$ of the messages satisfying following properties*

(i) the modified basic scheme$(\mathcal{S}, last = D)$ broadcasts the messages without collisions;

(ii) $s_1 \in \{m_1,m_2,m_3\}$, $s_2 \in \{m_1,m_2,m_3,m_4\}$ and $s_i \in \{m_{i-2},m_{i-1},m_i,m_{i+1},m_{i+2}\}$ for any $2 < i \leq M-2$, and $s_{M-1} \in \{m_{M-3},m_{M-2},m_{M-1},m_M\}$ and $s_M \in \{m_{M-1},m_M\}$;

(iii) for any $i \leq M$, if s_i is an H_Y (resp., V_X) sending with destination on column 0 (resp., on line 0), then either $s_i \in \{m_i,m_{i+1},m_{i+2}\}$ if $i < M-1$, or $s_i \in \{m_i,m_{i+1}\}$ if $i = M-1$, or $s_i = m_i$ if $i = M$.

Proof. We prove the theorem for $D = V$. The case $D = H$ can be proved similarly. The proof is by induction on M and follows the proof of Theorem 19. We have to verify the new property (iii) and property (i) when one of p,q,m_{M-1},m_M has its destination on one of the axis. Recall that q is the last message in $\mathcal{O}$. We will denote $dest(p) = (x_p,y_p)$,and as usual $dest(m_{M-1}) = (x_{M-1},y_{M-1})$ and $dest(m_M) = (x_M,y_M)$.

For property (i) the proof of Theorem 19 works if, when using Lemma 55, we are in a case where it is still valid, that is when Lemma 58 is valid. We use Lemma 55 to prove case 2 of the Algorithm $TwoApprox[d_I = 0, last = V](\mathcal{M})$ with p,m_{M-1},m_M in this order. The order on the messages implies $d(m_{M-1}) \geq d(m_M)$ and so by Corollary 12, Lemma 58 is valid. We also use Lemma 55 to prove the case 3 of the algorithm with q,p,m_{M-1} in this order. The order on the messages implies $d(p) \geq d(m_{M-1})$ and so by Corollary 12, Lemma 58 is valid. Note that to prove case 4 of the algorithm we use Lemma 56 which is still valid (Lemma 59).

It remains to verify property (iii). In case 2 of the algorithm, we have to show that $s_M = m_{M-1}$ is not using V_X-sending because we use induction for $(m_1,\ldots,m_{M-2})$. So it is sufficient to prove $y_{M-1} > 0$. Indeed, by Fact 7, $(m_{M-1},m_M) \notin HV$ implies $y_{M-1} > 0$.

In case 3 of the algorithm, to verify property (iii) we have to show that $s_{M-1} = p$ is not using H_Y-sending because we use induction for $(m_1,\ldots,m_{M-2})$. So it is sufficient to prove $x_p > 0$. Indeed, by Fact 8, $(p,m_{M-1}) \notin VH$ implies $x_p > 0$.

In case 4 of the algorithm, to verify property (iii) we have to show that $s_M = m_{M-1}$ is not using V_X-sending. Suppose it is not the case i.e. $y_{M-1} = 0$; as $(p,m_{M-1}) \notin VH$, we have by Fact 8.2 $y_p = 0$ and $x_{M-1} > x_p$. But then $d(p) < d(m_{M-1})$ contradicts the order of the messages. □

As corollary we get by properties (ii) and (iii) and the definition of *LB*, that the modified basic scheme$(\mathcal{S}, last = D)$ achieves a makespan at most $LB+2$. We emphasize this result as a Theorem and note that in view of Example 2 or the example given at the end of Section 6.2 it is the best possible.

Theorem 25. *In the general grid with BS in the corner and $d_I = 0$, the modified basic scheme $(\mathcal{S}, last = D)$ obtained by the Algorithm $TwoApprox[d_I = 0, last = D](\mathcal{M})$ achieves a makespan at most $LB+2$.*

6.5 d_I-Open Grid when $d_I \in \{1,2\}$

In this section, we use the Algorithm $OneApprox[d_I = 0, last = D](\mathcal{M})$ and the detour similar with the one in Example 4 to solve the personalized broadcasting problem for $d_I \in \{1,2\}$ in d_I-open grids, defined as follows:

Definition 10. *A grid with $BS(0,0)$ in the corner is called 1-open grid if at least one of the following conditions is satisfied: (1) All messages have destination nodes in the set $\{(x,y) : x \geq 2$ and $y \geq 1\}$; (2) All messages have destination nodes in the set $\{(x,y) : x \geq 1$ and $y \geq 2\}$.*

The 1-open grid differs from the open grid only by excluding destinations of messages either on the line $x = 1$ (condition (1)) or on the column $y = 1$ (condition (2)). For $d_I \geq 2$ the definition is simpler.

Definition 11. *For $d_I \geq 2$, a grid with $BS(0,0)$ in the corner is called d_I-open grid if all messages have destination nodes in the set $\{(x,y) : x \geq d_I$ and $y \geq d_I\}$.*

6.5.1 Lower bounds

In this subsection, we give the lower bounds of the makespan for $d_I \in \{1,2\}$ in d_I-open grids:

Proposition 2. *Let G be a grid with BS in the corner, $d_I = 1$ and the set of messages, $\mathcal{M} = (m_1,m_2,\cdots,m_M)$, ordered by non-increasing distances from BS, with all the destinations at distance at least 3 ($d(m_M) \geq 3$), then the makespan of any broadcasting scheme is greater than or equal to $LB_c(1) = max_{i \leq M} d(m_i) + \lceil 3i/2 \rceil - 2$.*

Proof. First we claim that if the source sends two messages in two concecutive steps t and $t+1$, then it cannot send at step $t+2$. Indeed, suppose that the source sends a message m at step t on one axis; then at step $t+1$ it must send the message m' on the other axis. But then at step $t+2$, both the two neighbors of the source are at distance at most 1 from the sender of messages m or m'. So if the source sends m'' at step $t+2$, m'' will interfere with m or m'.

Let t_i be the step where the last message in $(m_1, m_2, \cdots, m_i)$ is sent; therefore $t_i \geq \lceil 3i/2 \rceil - 1$. This last message denoted m is received at step $t'_i \geq d(m) + t_i - 1 \geq d(m_i) + t_i - 1 \geq d(m_i) + \lceil 3i/2 \rceil - 2$ and for every $i \leq M$, $LB_c(1) \geq d(m_i) + \lceil 3i/2 \rceil - 2$. □

Remark 10. *(A): Obviously, this bound is valid for 1-open grid according to Definition 10.*

(B): This bound is valid for $d_I = 1$ only when the source has a degree 2 (case BS in the corner of the grid). If BS is in a general position in the grid we have no better bound than LB.

(C): One can check that the bound is still valid if at most one message has a destination at distance 1 or 2. But if two or more messages have such destinations ($d(m_{M-1}) \leq 2$), then the bound is no more valid. As an example, let $dest(m_i) = v_i$, with $v_1 = (1,2)$, $v_2 = (2,1)$, $v_3 = (1,2)$ and $v_4 = v_5 = (1,1)$, then $d(m_1) = d(m_2) = d(m_3) = 3$ and $d(m_4) = d(m_5) = 2$ and $LB_c(1) = d(m_5) + 6 = 8$. However we can achieve a makespan of 7 by sending m_4 horizontally at step 1, then m_1 vertically at step 2 and m_2 horizontally at step 3, then the source sends m_3 vertically at step 5 and m_5 horizontally at step 6. m_3 and m_5 reach their destinations at step 7.

(D): Finally let us also remark that there exist configurations for which no gathering protocol can achieve better makespan than $LB_c(1) + 1$. Let $dest(m_1) = v_1 = (x,y)$, with $x + y = d$, $dest(m_2) = v_2 = (x, y-1)$ and $dest(m_3) = v_3 = (x-1, y-2)$. To achieve a makespan of $LB_c(1) = d$, m_1 should be sent at step 1 via a shortest directed path; m_2 should be sent at step 2 via a shortest directed path; and m_3 should be sent at step 4 via a shortest directed path. But, at step d, the sender of m_2 (either $(x, y-2)$ or $(x-1, y-1)$) is at distance 1 from $v_3 = dest(m_3)$ and so m_2 and m_3 interfere.

For $d_I \geq 2$, we have the following lower bound.

Proposition 3. *Let $d_I \geq 2$ and suppose we are in d_I-open grid. Let $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ be the set of messages ordered by non-increasing distances from BS, then the makespan of any broadcasting scheme is greater than or equal to $LB(d_I) = max_{i \leq M} d(m_i) + (i-1)d_I$.*

Proof. Indeed if a source sends a message at some step the next message has to be sent at least d_I steps after.

□

Remark 11. *For $d_I = 2$, there exist configurations for which no gathering protocol can achieve a better makespan than $LB(2) + 2$. Let $dest(m_1) = v_1 = (x,y)$, with $x + y = d$ and $dest(m_2) = v_2 = (x-1, y-1)$. Note that $LB(2) = d$. Let s_1, s_2 be the sequence obtained*

by some algorithm ; to avoid interferences s_1 being sent at step 1 , s_2 should be sent at step ≥ 3. If $s_2 = m_1$, the makespan is at least $d+2$; Furthermore, if m_1 is not sent via a shortest directed path again the makespan is at least $d+2$. So $s_1 = m_1$ is sent at step 1 via a shortest directed path. At step d the sender of m_1 (either $(x,y-1)$ or $(x-1,y)$ is at distance 1 from v_2. Therefore, if m_2 is sent at step 3 (resp., 4) it arrives at v_2 (resp.,at a neighbor of v_2) at step d and so m_2 interferes with m_1. Thus, m_2 can be sent in the best case at step 5 and arrives at step $d+2$. In all the cases, the makespan of any algorithm is $LB(2)+2$.

6.5.2 Routing with ε-detours

To design the algorithms for $d_I \in \{1,2\}$, we will use the sequence $\mathcal{S}$ obtained by Algorithm $OneApprox[d_I = 0, first = D](\mathcal{M})$. First, as seen in the proof of lower bounds, the source will no more send a message at each step. Second, we need to send the messages via directed paths more complicated than horizontal or vertical sendings; however we will see that we can use relatively simple directed paths with at most 2 turns and simple detours. Let us define precisely such sendings.

Definition 12. *We say that a message to be sent to node (x,y) is sent vertically with an ε-detour, if it follows the directed path from $BS(0,0)$ to $(0,y+\varepsilon)$, then from $(0,y+\varepsilon)$ to $(x,y+\varepsilon)$ and finally from $(x,y+\varepsilon)$ to (x,y). Similarly a message to be sent to node (x,y) is sent horizontally with an ε-detour, if it follows the directed path from $BS(0,0)$ to $(x+\varepsilon,0)$, then from $(x+\varepsilon,0)$ to $(x+\varepsilon,y)$ and finally from $(x+\varepsilon,y)$ to (x,y).*

Note that $\varepsilon = 0$ corresponds to a message sent horizontally (or vertically) as defined earlier (in that case we will also say that the message is sent without detour). Note also that in the previous section we use directed paths with 1-detour but only to reach vertices on the axes which are now excluded, since we are in open grid. A message sent at step t with an ε-detour reaches its destination at step $t+d(m)+2\varepsilon-1$. We also note that the detours introduced here are slightly different from the one used in Example 4. They are simpler in the sense that they are doing only two turns and for the case $\varepsilon = 1$ (1-detour) going backward only at the last step.

We will design algorithms using the sequence obtained by Algorithm $OneApprox[d_I = 0, first = D](\mathcal{M})$ but we will have to send some of the messages with a 1-detour. We will first give some lemmata which characterize when two messages m and m' interfere when $d_I = 1$, but not interfere in the basic scheme that is when $d_I = 0$, according to the detours of their sendings. For that the following fact which gives precisely the arcs used by the messages will be useful.

Fact 9.

- *If $dest(m) = (x,y)$ and m is sent horizontally at step t with an ε-detour ($\varepsilon = 0$ or 1) then it uses at step $t+h$ the following arc*

  case 1: $((h,0),(h+1,0))$ *for* $0 \leq h < x+\varepsilon$

  case 2: $((x+\varepsilon,h-(x+\varepsilon)),(x+\varepsilon,h+1-(x+\varepsilon))$ *for* $x+\varepsilon \leq h < x+y+\varepsilon$

  case 3: if $\varepsilon = 1$ $((x+1,y),(x,y))$ *for* $h = x+y+1$

- *If $dest(m') = (x',y')$ and m' is sent vertically with an ε'-detour ($\varepsilon' = 0$ or 1) at step t', then it uses at step $t'+h'$ the following arc*

case 1': $((0,h'),(0,h'+1))$ *for* $0 \le h' < y' + \varepsilon'$

case 2': $((h'-(y'+\varepsilon'),y'+\varepsilon'),(h'+1-(y'+\varepsilon'),y'+\varepsilon'))$ *for* $y'+\varepsilon' \le h' < x'+y'+\varepsilon'$

case 3': *if* $\varepsilon' = 1$ $((x',y'+1),(x',y'))$ *for* $h' = x'+y'+1$

Lemma 60. *Let G be an open grid. Let $dest(m) = (x,y)$ and m be sent at step t horizontally without detour, i.e. $\varepsilon = 0$. Let $dest(m') = (x',y')$ and m' be sent vertically with an ε'-detour ($\varepsilon' = 0$ or 1) at step $t' = t+1$. Let furthermore $\{x' < x$ or $y' \ge y\}$ (i.e. $(m,m') \in HV$ in the basic scheme). Then for $d_I = 1$, m and m' do not interfere.*

Proof. To prove that the two messages do not interfere, we will prove that at any step for any pair of messages sent but not arrived at destination, the distance between the sender of one and the receiver of the other is ≥ 2. Consider a step $t+h = t'+h'$ where $h' = h-1$ and $1 \le h < min\{x+y, x'+y'+1+2\varepsilon'\}$. By Fact 9 we have to consider 6 cases. We label them as case *i-j*' if we are in case *i* for *m* and in case *j*' for m', $i = 1,2$ and $1 \le j \le 3$:

case 1-1': $1 \le h < x$ and $0 \le h-1 < y'+\varepsilon'$. Then, the distance between a sender and a receiver is at least $2h \ge 2$.

case 1-2': $1 \le h < x$ and $y'+\varepsilon' \le h-1 < x'+y'+\varepsilon'$. Then, the distance between a sender and a receiver is at least $2(y'+\varepsilon') \ge 2$, as $y' \ge 1$.

case 1-3': $1 \le h < x$ and $\varepsilon' = 1$ $h-1 = x'+y'+1$. Then, the distance between a sender and a receiver is at least $h-x'+y' = 2y'+2 \ge 4$.

case 2-1': $x \le h < x+y$ and $0 \le h-1 < y'+\varepsilon$. Then, the distance between a sender and a receiver is at least $|x|+|x-2| \ge 2$.

case 2-2': $x \le h < x+y$ and $y'+\varepsilon' \le h-1 < x'+y'+\varepsilon'$. Recall that $(m,m') \in HV$; so, by Fact 6, $x' < x$ or $y' \ge y$. If $x' < x$, as $h \le x'+(y'+\varepsilon')$, we get $h \le x+(y'+\varepsilon')-1$. If $y' \ge y$, $h < x+y$ implies $h \le x+y'-1$. But, the distance between a sender and a receiver is at least $2(x+(y'+\varepsilon')-h) \ge 2$ in both cases.

case 2-3': $x \le h < x+y$ and $\varepsilon' = 1$ $h-1 = x'+y'+1$. Then, the distance between a sender and a receiver is at least $|x'-x|+|x'-x+2| \ge 2$.

□

The next lemma will be used partly for proving the correctness of algorithm for $d_I = 1$ (since the last case in the lemma will not happen in the algorithm) and fully for the algorithm for $d_I = 2$.

Lemma 61. *Let G be an open-grid. Let $dest(m) = (x,y)$ with $x \ge 2$ and m be sent horizontally at step t with an ε-detour ($\varepsilon = 0$ or 1). Let $dest(m') = (x',y')$ and m' be sent vertically with an ε'-detour ($\varepsilon' = 0$ or 1) at step $t' = t+2$. Let furthermore $\{x' < x$ or $y' \ge y\}$ (i.e. $(m,m') \in HV$ in the basic scheme). Then, for $d_I = 1$ or 2, m and m' interfere if and only if*

case 00. $\varepsilon = 0, \varepsilon' = 0$*:* $x' = x-1$ *and* $y' \leq y-1$

case 01. $\varepsilon = 0, \varepsilon' = 1$*:* $x' = x-1$ *and* $y' \leq y-2$

case 10. $\varepsilon = 1, \varepsilon' = 0$*:* $x' \geq x$ *and* $y' = y$

case 11. $\varepsilon = 1, \varepsilon' = 1$*:* $x' = x-1$ *and* $y' = y-1$

Proof. Consider a step $t+h = t'+h'$ so $h' = h-2$. By Fact 9 we have to consider 9 cases according the 3 possibilities for an arc used by m and the 3 possibilities for an arc used by m'. We label them as case *i-j*' if we are in case i for m and in case j' for m', $1 \leq i, j \leq 3$. We will prove that in all the cases, the distance of the sender and receiver of these two messages is either at most 1 or at least 3. So the interference happens in the same condition for $d_I = 1$ and $d_I = 2$.

case 1-1': Then, the distance between a sender and a receiver is at least $2h-1 \geq 3$ as $h' = h-2 \geq 0$.

case 1-2': Then, the distance between a sender and a receiver is at least $2(y'+\varepsilon')+1 \geq 3$ as $y' \geq 1$.

case 1-3': $h = h'+2 = x'+y'+3$. The distance between a sender and a receiver is at least $h-x'+y' = 2y'+3 \geq 5$, as $y' \geq 1$.

case 2-1': Then, the distance between a sender and a receiver is either $|x+\varepsilon| + |x+\varepsilon-3| \geq 3$ or $2(x+\varepsilon)-1 \geq 3$ as $x \geq 2$.

case 2-2': Then, the distance between a sender and a receiver is at least $2(x+\varepsilon+y'+\varepsilon'-h)+1$. If $y' \geq y-\alpha$, then $h \leq x+y+\varepsilon-1 \leq x+y'+\alpha+\varepsilon-1$ implies $x+\varepsilon+y'+\varepsilon'-h \geq \varepsilon'+1-\alpha$ and the distance is at least $2\varepsilon'+3-2\alpha$. If $\alpha \leq 0$ ($y' \geq y$) then the distance is ≥ 3. Furthermore if $\varepsilon' = 1$ and $\alpha = 1$, the distance is also ≥ 3.

Otherwise, $y' < y$ and by the hypothesis $x' < x$. Let $x' = x-1-\beta$ with $\beta \geq 0$; $h'+2 = h \leq x'+y'+\varepsilon'+1 = x-\beta+y'+\varepsilon'$ implies $x+\varepsilon+y'+\varepsilon'-h \geq \varepsilon+\beta$ and the distance is at least $2\varepsilon+1+2\beta$. If $\beta \geq 1$ or $\varepsilon = 1$, then the distance is ≥ 3. Otherwise when $\beta = 0$ (i.e. $x' = x-1$) and $\varepsilon = 0$, we have a distance 1, achieved for $h = x'+y'+\varepsilon'+1$. More precisely when $\varepsilon' = 0$, it is achieved with $x' = x-1$ and $y' \leq y-1$, which corresponds to case 00. When $\varepsilon' = 1$, we have already seen that the distance is 3, for $y' = y-1$ (case $\alpha = 1$); otherwise the distance is 1 with $x' = x-1$ and $y' \leq y-2$ (case 01).

case 2-3': In this case $\varepsilon' = 1$ and $h = x'+y'+3 \leq x+y+\varepsilon-1$. The distance between a sender and a receiver is $|x+\varepsilon-x'|$ + $|x+\varepsilon+y'-h|$. If $y' \geq y-1$, $h = x'+y'+3 \leq x+y+\varepsilon-1 \leq x+y'+\varepsilon$ implies $x' \leq x+\varepsilon-3$ and so $|x+\varepsilon-x'| \geq 3$. Otherwise, by hypothesis, $x' < x$; if $x' \leq x-3$, then $|x+\varepsilon-x'| \geq 3$. In the remaining case $x' = x-1$ or $x' = x-2$. If $x' = x-1$, then $|x+\varepsilon-x'| = 1+\varepsilon$ and $h = x'+y'+3 = x+y'+2$, which implies $|x+\varepsilon+y'-h| = 2-\varepsilon$. So the distance is 3; If $x' = x-2$, then $|x+\varepsilon-x'| = 2+\varepsilon$ and $h = x'+y'+3 = x+y'+1$, which implies $|x+\varepsilon+y'-h| = 1-\varepsilon$. So the distance is 3.

case 3-1': Then, the distance between a sender and a receiver is at least $2x-1 \geq 3$ as $x \geq 2$.

case 3-2': In that case $\varepsilon = 1$ and $h = x+y+1$ and $h \leq x'+y'+\varepsilon'+1$. The distance between a sender and a receiver is $|x+y'+\varepsilon'+2-h| + |y'+\varepsilon'-y|$. If $x' \leq x-1$, $h = x+y+1 \leq x'+y'+\varepsilon'+1 \leq x+y'+\varepsilon'$ implies $|x+y'+\varepsilon'+2-h|+y'+\varepsilon'-y \geq 2+1=3$. Otherwise $x' \geq x$, and, by hypothesis, $y' \geq y$; Let $y' = y+\gamma$ with $\gamma \geq 0$. So $x+y'+\varepsilon' = x+y+\gamma+\varepsilon' = h-1+\gamma+\varepsilon'$ implies $|x+y'+\varepsilon'+2-h|+y'+\varepsilon'-y \geq 2\varepsilon'+2\gamma+1$. If $\varepsilon'=1$ or $\gamma \geq 1$, then the distance is at least 3; otherwise the distance is 1 and so we have interference if $\varepsilon = 1, \varepsilon' = 0, x' \geq x$ and $y' = y$ (case 10).

case 3-3': Then , $\varepsilon = 1, \varepsilon' = 1$ and $h = x+y+1 = x'+y'+3$. The distance between a sender and a receiver is either $|x+1-x'|+|y'-y|$ or $|x-x'|+|y'+1-y|$. If $y' \geq y$, then $h = x+y+1 = x'+y'+3$ implies $x \geq x'+2$ and the distance is 3. If $y' \leq y-1$, then by hypothesis $x' \leq x-1$ and $x+y+1 = x'+y'+3$ implies $y' = y-1$ and $x' = x-1$. Then the distance is 1 we have interference. In summary, we have interference if $\varepsilon = 1, \varepsilon' = 1, x' = x-1$ and $y' = y-1$ (case 11).

□

By exchanging horizontally and vertically , x and y and x' and y' in Lemma 60 and Lemma 61 we get the following two lemmata:

Lemma 62. *Let G be open grid. Let dest$(m) = (x,y)$ and m be sent vertically (without detour) at step t. Let dest$(m') = (x',y')$ and m' be sent horizontally with an ε'-detour ($\varepsilon' = 0$ or 1) at step $t' = t+1$. Let furthermore $\{x' \geq x$ or $y' < y\}$ (i.e. $(m,m') \in VH$ in the basic scheme). Then, for $d_I = 1$, m and m' do not interfere.*

Lemma 63. *let G be an open grid. Let dest$(m) = (x,y)$ with $y \geq 2$ and m be sent vertically at step t with an ε-detour ($\varepsilon = 0$ or 1). Let dest$(m') = (x',y')$ and m' be sent horizontally with an ε'-detour ($\varepsilon' = 0$ or 1) at step $t' = t+2$. Let furthermore $\{x' \geq x$ or $y' < y\}$ (i.e. $(m,m') \in VH$ in the basic scheme). Then for $d_I = 1$ or 2, m and m' interfere if and only if*

case 00. $\varepsilon = 0, \varepsilon' = 0$*: $x' \leq x-1$ and $y' = y-1$*

case 01. $\varepsilon = 0, \varepsilon' = 1$*: $x' \leq x-2$ and $y' = y-1$*

case 10. $\varepsilon = 1, \varepsilon' = 0$*: $x' = x$ and $y' \geq y$*

case 11. $\varepsilon = 1, \varepsilon' = 1$*: $x' = x-1$ and $y' = y-1$*

6.5.3 General-scheme $d_I = 1$.

We will have to define general-scheme by indicating not only the ordered sequence of messages $\mathcal{S} = (s_1, \cdots, s_M)$ sent by the source, but also by specifying for each s_i the time t_i at which the message s_i is sent and the directed path followed by the message s_i, in fact the direction D_i and the ε_i-detour used for sending it. More precisely,

Definition 13. *A general-scheme is defined as a sequence of M quadruples $(s_i,t_i,D_i,\varepsilon_i)$, where the i-th message sent by the source is s_i. This message is sent at step t_i in direction D_i with an ε_i-detour.*

Note that we will send the messages alternatively horizontally and vertically in our algorithm. Therefore, we have only to specify the direction of the first (or last) message. We will see in the next theorem that the sequence $\mathcal{S}$ obtained by the algorithm $OneApprox[d_I = 0, first = D](\mathcal{M})$ in Section 6.3 almost works when $d_I = 1$. More precisely, we propose a scheme that sends the messages in the same order as in $\mathcal{S}$. However, BS waits one step every three steps; i.e., the source sends two messages of the sequence $\mathcal{S}$ during two consecutive steps and then stops sending for one step. Furthermore, a message must sometimes be sent with a detour to avoid interference. That is, the messages are sent without detours like in $\mathcal{S}$, except that, if the first message is sent in direction D, an even message s_{2k+2} is sent in direction $\bar{D}$ with a 1-detour if and only if without detour it would interfere with s_{2k+3}.

Theorem 26. *Let $d_I = 1$, and let BS be in a corner of a 1-open grid. Let $\mathcal{M} = (m_1,\ldots,m_M)$ be the set of messages ordered by non-increasing distances from BS and suppose that the destination $v = (x,y)$ of any message satisfies $\{x \geq 1, y \geq 2\}$ (condition (2) of 1-open grid). Let us define:*

- *$\mathcal{S} = (s_1,\ldots,s_M)$ is the ordered sequence obtained by the Algorithm $OneApprox[d_I = 0, first = H](\mathcal{M})$*
- *for any $i = 2k+1$, $0 \leq k \leq \lfloor (M-1)/2 \rfloor$, let $t_i = 3k+1$ $D_i = H$ and $\varepsilon_i = 0$,*
- *for any $i = 2k+2$, $0 \leq k < \lfloor M/2 \rfloor$, let $t_i = 3k+2$, $D_i = V$ and $\varepsilon_{2k+2} = 0$ if s_{2k+2} does not interferes with s_{2k+3} for $d_I = 1$, otherwise $\varepsilon_{2k+2} = 1$.*

Then the general-scheme defined by the sequence $(s_i,t_i,D_i,\varepsilon_i)_{i\leq M}$ broadcasts the messages without collisions for $d_I = 1$ and the first message is sent in direction H.

Proof. To prove the theorem, we need to prove that any two messages do not interfere at any step in the general scheme with parameters $(s_i,t_i,D_i,\varepsilon_i)$. A message s_i cannot interfere with a message s_{i+j} for $j \geq 2$ sent at least 3 steps after; indeed the senders of such two messages will be at distance at least 3 (at each step, including the last step when the messages do a 1-detour, the distance of a sender to the base station increases by one). So we have only to care about s_i and s_{i+1}.

First consider the message s_{2k+1}. Let $s_{2k+1} = m$, with $dest(m) = (x,y)$ and $s_{2k+2} = m'$, with $dest(m') = (x',y')$. Message m is sent horizontally at step $t = 3k+1$ without detour and m' is sent vertically at step $t' = t+1 = 3k+2$ with an ε'-detour for $\varepsilon' = \varepsilon_{2k+2}$. Furthermore, by Theorem 21, we have $(m,m') \in HV$. We conclude by Lemma 60 that s_{2k+1} and s_{2k+2} do not interfere.

Now let us prove that s_{2k+2} does not interfere with s_{2k+3}. Let $s_{2k+2} = m$ with $dest(m) = (x,y)$ and $s_{2k+3} = m'$ with $dest(m') = (x',y')$. Message m is sent vertically with an ε-detour, $\varepsilon = \varepsilon_{2k+2}$ at step $t = 3k+2$ and m' is sent horizontally at step $t' = t+2 = 3k+4$. Furthermore by Theorem 21 $(m,m') \in VH$ and so $\{x' \geq x \text{ or } y' < y\}$ by Fact 6. So we can apply Lemma 63. If $\{x' \leq x-1 \text{ and } y' = y-1\}$, we are in the case 00 of Lemma 63 and

so if m and m' were sent without detour they would interfere. Then by the algorithm we have to choose $\varepsilon_{2k+2} = 1$, but now we are in the case 10 of Lemma 63 which implies no interference. (Case 11 never happens in the Theorem.) Otherwise we have $\{x' > x-1$ or $y' \neq y-1\}$; also we have $\varepsilon = 0$ according to the Theorem. By case 00 of Lemma 63, they do not interfere. The proof works because interferences in case 00 and 10 of Lemma 63 cannot appear simultaneously. □

Remark 12. *Note that we cannot relax the hypothesis that the messages satisfy $y \geq 2$. Indeed if $y = 1$, we might have to do a 1-detour for $m = s_{2k+2}$ when $x' \geq x$ as at any step $t+h$ ($2 \leq h \leq x$) the sender of m is at distance 1 from the receiver of $m' = s_{2k+3}$ (case 2-1' in the proof). So we have to send m vertically with a 1-detour; but at step $t+x+2$ the sender of m' $(x',0)$ is at distance 1 from the receiver of m $(x',1)$ (case 3-1' in the proof). A simple example is given with 3 messages m_1, m_2, m_3 whose destinations are respectively $(5,1),(4,1),(3,1)$. Then OneApprox$[d_I = 0, first = H](\mathcal{M})$gives the sequence (m_1, m_3, m_2), where $m_3 = s_2$ is sent vertically at step 2 and $m_2 = s_3$ is sent horizontally at step 3. Now, for $d_I = 1$, m_2 is sent at step 4. If m_3 is sent without detour, it interferes with m_2 at step 4 and 5; otherwise if m_3 is sent with a 1-detour it interferes with m_2 at step 7.*

By exchanging x and y, H and V, we also get that when the destination $v = (x,y)$ of any message satisfies $\{x \geq 2, y \geq 1\}$ (condition (1) of 1-open grid) we can adapt our algorithm to compute a general-scheme that broadcasts the messages without collisions for $d_I = 1$ and where the first message is sent in direction V. Furthermore, if we are in a 2-open grid we can have a general-scheme where the direction of the first message is arbitrary.

Theorem 27. *In the 1-open grid with BS in the corner and $d_I = 1$, there exists a general-scheme achieving a makespan at most $LB_c(1)+3$.*

Proof. Applying the Algorithm *OneApprox*$[d_I = 0, first = D](\mathcal{M})$, we get an ordered sequence $\mathcal{S}$ which satisfies the Property (b) of Theorem 21: $m_i \in \{s_{i-1}, s_i, s_{i+1}\}$. Consider parameters as in Theorem 26 in case of condition (2) of 1-open grid (the proof is similar for condition(1)). Recall that a message m sent at step t with an ε-detour reaches its destination at step $d(m) + 2\varepsilon + t - 1$. Then s_{2k+1} reaches its destination (the worst case being $s_{2k+1} = m_{2k}$ sent without detour at step $3k+1$) at step at most $d(m_{2k}) + 3k + 1 - 1 = d(m_{2k}) + \lceil \frac{3(2k)}{2} \rceil - 2 + 2$. Similarly s_{2k+2} reaches its destination (the worst cases being $s_{2k+2} = m_{2k+1}$ sent with a 1-detour at step $3k+2$) at step at most $d(m_{2k+1}) + 2 + 3k + 2 - 1 = d(m_{2k+1}) + 3k + 3 = d(m_{2k+1}) + \lceil \frac{3(2k+1)}{2} \rceil - 2 + 3$. So the makespan is at most $max_{i \leq M} d(m_i) + \lceil 3i/2 \rceil + 1 = LB_c(1) + 3$. □

6.5.4 General-scheme $d_I = 2$.

In this section, we present a linear-time (in the number of messages) algorithm that computes a general-scheme (Definition 13) broadcasting the messages without collisions for $d_I = 2$ in a 2-open grid, and achieving a makespan up to 4 from the optimal.

As in the case $d_I = 1$, *BS* will send the messages in the same order as in $\mathcal{S}$. However, BS sends one message only every two steps (which is necessary when $d_I = 2$). The difficulty here is to decide the detour that must be followed by each message, in order to avoid

interference. Next algorithm, described in Figure 6.8, is dedicated to compute the sequence $(\varepsilon_i)_{i\leq M}$ of the detours.

Input: $\mathcal{M} = (m_1, \cdots, m_M)$ the set of messages ordered by non-increasing distances from *BS*, in a 2-open grid, and the direction D of the first message.
Output: $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_M)$ where $\varepsilon_i \in \{0,1\}$
begin
1 Let $(s_1, \cdots, s_M) = OneApprox[d_I = 0, first = D](\mathcal{M})$
2 Let $t_i = 2i-1$, and $D_i = D$ if i is odd and $D_i = \bar{D}$ otherwise, for any $1 \leq i \leq M$
3 Let start with $\varepsilon_i = 1$ for $1 \leq i \leq M$.
4 **for** $i = M-1$ to 1
5 **if** s_i interferes with s_{i+1} in the general-scheme defined by $(s_i, t_i, D_i, \varepsilon_i)_{i\leq M}$ when $d_I = 2$ **then**
(we emphasis that we consider interferences with the *current* values of the $(\varepsilon_i)_{i\leq M}$)
6 $\varepsilon_i \leftarrow 0$
7 **return** $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_M)$
end

Figure 6.8: Algorithm $Epsilon(\mathcal{M}, first = D)$

Theorem 28. *Let $d_I = 2$, and let BS be in a corner of a 2-open grid. Let $\mathcal{M} = (m_1, \ldots, m_M)$ be the set of messages ordered by non-increasing distances from BS. Let us define $(s_i, t_i, D_i, \varepsilon_i)_{i\leq M}$ such that*

- *$\mathcal{S} = (s_1, \ldots, s_M)$ is the ordered sequence obtained by the Algorithm $OneApprox[d_I = 0, first = D](\mathcal{M})$*
- *for any $i \leq M$, $t_i = 2i-1$ and $D_i = D$ if i is odd and $D_i = \bar{D}$ otherwise.*
- *$\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_M)$ is the sequence obtained by Algorithm $Epsilon(\mathcal{M}, first = D)$*

Then the general-scheme defined by the sequence $(s_i, t_i, D_i, \varepsilon_i)_{i\leq M}$ broadcasts the messages without collisions for $d_I = 2$ and the first message is sent in direction D.

Proof. We need to prove that any two messages do not interfere at any step. A message s_i cannot interfere with a message s_{i+j}, for $j \geq 2$, sent at least 4 steps after. Indeed, at any step, the senders of two such messages are at distance at least 4. This is because, at each step including the last step when the messages do a 1-detour the distance of a sender to the base station increases by one. So we have only to show that s_i does not interfere with s_{i+1} for any $1 \leq i < M$. For this purpose, we need the following claim that we will prove thanks to Lemma 61 and 63.

Claim 6. *For $d_I = 2$, if s_i sent with an $\varepsilon_i = 1$-detour interferes with s_{i+1} , then if we send s_i without detour, s_i does not interfere with s_{i+1}.*

Indeed suppose s_i is sent in direction D. As the sequence $\mathcal{S}$ is obtained by Algorithm $OneApprox[d_I = 0, first = D](\mathcal{M})$, $(s_i, s_{i+1}) \in D\bar{D}$. So we are in cases 10 if $\varepsilon_{i+1} = 0$ or in case 11 if $\varepsilon_{i+1} = 1$ of Lemma 61 ($D = H$) or Lemma 63 ($D = V$). First suppose that we are in case 10, then we are not in the case 00; therefore if we send s_i without detour, s_i does not interfere with s_{i+1}. Now assume that we are in case 11, then we are not in the case 01; therefore if we send s_i without detour, s_i does not interfere with s_{i+1}.

Now the algorithm $Epsilon(\mathcal{M}, first = D)$ was designed in such a way it gives either $\varepsilon_i = 1$ in which case s_i does not interfere with s_{i+1} or it gives $\varepsilon_i = 0$ because s_i sent with a 1 detour was interfering with s_{i+1}, but then, by the claim 6, s_i sent without detour does not interfere with s_{i+1}.

□

Theorem 29. *In the 2-open grid with BS in the corner and $d_I = 2$, the general-scheme defined in Theorem 28 achieves a makespan at most $LB(2) + 4$.*

Proof. By definition of the scheme, the messages are sent in the same order as computed by $OneApprox[d_I = 0, first = D](\mathcal{M})$. Therefore, by Property (b) of Theorem 21, $s_i \in \{m_{i-1}, m_i, m_{i+1}\}$. So the message s_i arrives at its destination at step $d(s_i) + 2\varepsilon_i + t_i - 1 \leq d(m_{i-1}) + 2 + 2i - 1 - 1 = d(m_{i-1}) + 2(i - 1 - 1) + 4$. Then the result follows from the definition of $LB(2)$. □

6.6 Personalized Broadcasting in Grid with Arbitrary Base Station

In this section, we show how to use the algorithms proposed above to broadcast (or equivalently to gather) a set of personalized messages $\mathcal{M}$, in a grid with a base station placed in an arbitrary node. More precisely, *BS* will still have coordinates $(0,0)$, but the coordinates of the other nodes are in $\mathbb{Z}$. A grid with arbitrary base station is said to be an *open-grid* if no destination nodes are on the axes. More generally, a grid with arbitrary base station is said to be an 2-*open-grid* if no destination nodes are at distance at most 1 from any axis.

We divide the grid into four *quadrants* $Q_q, 1 \leq q \leq 4$, where $Q_1 = \{(x,y)$ such that $x \geq 0, y \geq 0\}$, $Q_2 = \{(x,y)$ such that $x \leq 0, y \geq 0\}$, $Q_3 = \{(x,y)$ such that $x \leq 0, y \leq 0\}$, and $Q_4 = \{(x,y)$ such that $x \geq 0, y \leq 0\}$. Note that, *BS* belongs to all quadrants, and any other node on an axis belongs to two different quadrants.

Each quadrant can be considered itself as a grid with the *BS* in the corner. Therefore, we can extend all the definitions of the preceding sections, in particular the basic scheme and general-scheme by considering a move in Q_1 (resp., Q_2, Q_3, Q_4) as horizontal, if it is on the positive x-axis (reps. positive y-axis, negative x-axis, negative y-axis) and a vertical move as one on the other half-axis of the quadrant. Then, if we have a sequence of consecutive messages, still ordered by non-increasing distance to *BS*, and all in the same quadrant we can apply any of the preceding algorithms. Otherwise, we can extend the algorithms by splitting the sequence of messages into maximal subsequences, where all the messages are in the same quadrant and applying any of the algorithms to this subsequence. We have just to be careful that there is no interference between the last message of a subsequence and

the first one of the next subsequence; fortunately we will take advantage of the fact that we can choose the direction of the first message of any subsequence.

Theorem 30. *Given a grid with any arbitrary base station BS, and* $\mathcal{M} = (m_1, m_2, \cdots, m_M)$ *the set of messages ordered by non-increasing distances from BS, then there are linear-time algorithms which broadcast the messages without interferences, with makespan:*

- *at most* $LB + 2$ *if* $d_I = 0$*;*
- *at most* $LB + 1$ *if* $d_I = 0$ *in an open-grid;*
- *at most* $LB_c(1) + 3$ *if* $d_I = 1$ *in a* 2*-open-grid;*
- *at most* $LB(2) + 4$ *if* $d_I = 2$ *in a* 2*-open-grid;*

Proof. We partition the ordered set of messages into maximal subsequences, of messages in the same quadrant. That is $\mathcal{M} = \mathcal{M}_1 \odot \mathcal{M}_2 \ldots \mathcal{M}_j \ldots \odot \mathcal{M}_t$, where all the messages in $\mathcal{M}_j$ belong to the same quadrant and the messages of $\mathcal{M}_j$ and $\mathcal{M}_{j+1}$ belong to different quadrants. Then, depending on the cases of the theorem, we apply Algorithms $TwoApprox[d_I = 0, first = D](\mathcal{M})$, $OneApprox[d_I = 0, first = D](\mathcal{M})$, or the algorithms defined in Theorems 26 or 28 to each $\mathcal{M}_j$, in order to obtain a sequence $\mathcal{S}_j$. Now we define the value of D in the algorithms by induction. The direction of the first message of $\mathcal{S}_1$ is arbitrary. Then the direction of the first message of $\mathcal{S}_{j+1}$ has to be chosen on an half-axis different from that of the last message of $\mathcal{S}_j$, which is always possible as two quadrants have at most one half axis in common. For example, suppose the messages of $\mathcal{M}_j$ belong to Q_1 and the last message of $\mathcal{S}_j$ is sent vertically (i.e. on the positive y-axis) and that the messages of $\mathcal{M}_{j+1}$ belong to Q_2, then the first message of $\mathcal{S}_{j+1}$ cannot be sent on the the positive y-axis (that is horizontally in Q_2), but should be sent to avoid interferences on the negative x-axis (that is vertically in Q_2). Otherwise if the last message of $\mathcal{S}_j$ is sent horizontally (i.e. on the positive x-axis), we can sent the first message of $\mathcal{S}_{j+1}$ as we want (as the positive x-axis does not belong to Q_2); similarly if the messages of $\mathcal{M}_{j+1}$ belong to Q_3 we can send the first message of $\mathcal{S}_{j+1}$ as we want (as there are no half axes in common between Q_1 and Q_3). Finally, in the case $d_I = 2$, we have to wait one step between the sending of the last message of $\mathcal{S}_j$ and the first message of $\mathcal{S}_{j+1}$. With these restrictions, we have no interferences between two consecutive messages inside the same $\mathcal{S}_j$ by the correctness of the various algorithms; furthermore we choose the direction of the first message of $\mathcal{S}_{j+1}$ and we add in the case $d_I = 2$ a waiting step in order to avoid interferences between the last message of $\mathcal{S}_j$ and the first message of $\mathcal{S}_{j+1}$. Unconsecutive messages are sent far apart to avoid interferences; indeed the distance between two senders is $> d_I + 1$. Finally the values of the makespan follow from that of the respective algorithms. □

Note that the values of LB (resp., $LB(2)$) are lower bounds for the case of an arbitrary position of *BS*. Therefore, we get the following corollary

Corollary 13. *There are linear-time (in the number of messages) algorithms that solve the gathering and the personalized broadcasting problems in any grid, achieving an optimal makespan up to an additive constant c where:*

- $c = 2$ *when* $d_I = 0$*;*
- $c = 1$ *in open-grid when* $d_I = 0$*;*
- $c = 3$ *in* 1*-open-grid when* $d_I = 1$ *and BS is a corner;*
- $c = 4$ *in* 2*-open-grid when* $d_I = 2$*.*

However, for $d_I = 1$, LB_c is not a lower bound when *BS* is not in the corner; the best lower bound we know is *LB*. In fact this bound can be achieved in some cases. For example suppose that, in the ordered sequence $\mathcal{M}$, the message m_{4j+q} belong to the quadrant Q_q, then we send the messages m_{4j+q} horizontally in Q_q that is on the positive x-axis for $q = 1$, on the positive y-axis for $q = 2$, on the negative x-axis for $q = 3$, and on the negative y-axis for $q = 4$. There is no interferences and the makespan is exactly *LB*. On the opposite, we conjecture that, when all the messages are in the same quadrant, we can obtain a makespan differing of $LB_c(1)$ by a small constant; so in that case our algorithm will give a good approximation.

Remark 13. *Note that when buffering is allowed at the intermediate nodes, LB is still a lower bound for the makespan of any personalized broadcasting or gathering scheme. All our algorithms get makespans at most* $\frac{3}{2}LB+3$ *for* $d_I = 1$*, since* $LB_c(1) \leq \frac{3}{2}LB$ *and* $2LB+4$ *for* $d_I = 2$*, since* $LB(2) \leq 2LB$*. So we have almost* $\frac{3}{2}$ *and* 2*-approximation algorithms for* $d_I = 1$ *and* $d_I = 2$ *in 2-open grid respectively when buffering is allowed. For the special grid networks, this improves the result in [BGK*$^+$*06], which gives a 4-approximation algorithm.*

6.7 Perspective

We give several algorithms for the personalized broadcasting and so the gathering problem in grids with arbitrary base station for $d_I \leq 2$. It will be nice to have additive approximations for $d_I \geq 3$; we try to generalize the ideas developed before by using ε detours with $\varepsilon \geq 2$; doing so, we can avoid interferences between consecutive messages, but not with messages s_i and s_{i+2}. Another challenging problem consists in determining the complexity of finding an optimal schedule and routing of messages for achieving the gathering in the minimum completion time or characterizing when the lower bound is achieved. Example 4 shows it might not be an easy problem. Determining if there is a polynomial algorithm to compute the makespan in the restricted case where messages should be sent via shortest directed paths seems also to be a challenging problem (See Example 4). Last but not least, a natural extension will be to consider the gathering problem for other network topologies.

CHAPTER 7

Conclusion and Perspective

In this chapter, we conclude the thesis and present some perspectives.

Conclusion

In this thesis, we have mainly studied two topics: tree decompositions and some routing problems.

In the first part, we have focused on tree decompositions. In Chapter 2, the minimum size tree decomposition is to find a tree decomposition of minimum number of bags under the constraint that the width of the tree decomposition is at most a given integer $k \geq 1$. We have proved that for $k \geq 4$, it is NP-hard to find a minimum size tree decomposition of width at most k in general graphs; it is still NP-hard for $k \geq 5$ in the class of connected graphs. Moreover, polynomial algorithms have been presented to find a minimum size tree decomposition of width at most 2 (resp. 3) in class of graphs of treewidth at most 2 (resp. trees and 2-connected outerplanar graphs). In Chapter 3, inspired by the study of cops and robber game in k-chordal graphs for $k \geq 3$, we have investigated the k-good tree decompositions for $k \geq 2$, in which each bag contains a dominating set of size at most $k-1$ and inducing a path in the graph. Given a graph G and an integer $k \geq 3$, we have proved that there is a quadratic time algorithm either outputs an induced cycle of size at least $k+1$ of G or computes a k-good tree decomposition of G. We have also studied the problem of computing the chordality of planar graphs admitting a k-good tree decomposition. We were able to propose a linear time algorithm for the subclass of planar graphs containing a dominating set of size at most $k-1$ inducing a path and all other vertices inducing a cycle. We conjecture that it is NP-hard to compute the chordality of the class of planar graphs, which contains a dominating set of size at most $k-1$ inducing a path.

In the second part, we have investigated several routing problems, some of which are solved by applying tree decompositions. In Chapter 4, we have designed a compact routing scheme for the class of graphs admitting a k-good tree decomposition. The compact routing scheme has message header and routing table of size $O(k \log \Delta + \log n)$ and additive stretch $O(k \log \Delta)$, where Δ is the maximum degree of the graph. In Chapter 5, the Prize Collecting Steiner Tree (PCST) problem has been studied in the class of graphs of treewidth at most 2. We have presented a linear algorithm for PCST problem in the class of graphs of treewidth at most 2. The algorithm has been generalized to the class of bounded treewidth graphs in [CMZ11]. Moreover, for the PCST problem with interval costs and prizes, we have proposed the min-max risk model and the min-sum risk model. Polynomial algorithms have been presented for solving these two risk models of the PCST problem . In Chapter 6, we have studied the Data Gathering problem in grid radio networks under small interference

distance. Linear constant approximation algorithms have be presented for several cases. The results have been presented in Table 6.1 on Page 114.

Perspectives

For a short term future work, several open problems have been mentioned in the last section of each chapter. I summarize them in the following.

Given a graph G of treewidth at most 3, the complexity of finding a minimum size tree decomposition of width at most 3 of G is still open. In addition, given a tree T and an integer $k > 3$, the complexity of finding a minimum size tree decomposition of width at most k of T is also open. More generally, given a graph G of treewidth $tw \geq 2$ and an integer $k > tw$, the complexity of finding a minimum size tree decomposition of width at most k of G is open.

Given a graph G and a fixed integer $k \geq 2$, the complexity of deciding whether there is a k-good tree decomposition of G is open. In particular, it is still open even for $k = 2$. It is interesting to find a k-good tree decomposition of G with the minimum k, e.g. in its applications in designing compact routing scheme in Chapter 4. We have conjectured that it is NP-hard to compute the chordality of the class of planar k-super-caterpillar graphs, and we have reduced this conjecture to the determination of the complexity of the problem of deciding whether a planar graph with a Hamiltonian path (without precision on that path) also has a Hamiltonian cycle. Although Hamiltonian path and Hamiltonian cycle are both well-studied subjects, the complexity of this problem is still open.

Our compact routing scheme for any k-good tree decomposition admissible graph, presented in Chapter 4, has small size of routing tables and addresses for each vertex, but it has an additive stretch $O(k \log \Delta)$, where Δ is the maximum degree of the graph. It will be nice to decrease the additive stretch to be $O(k)$ without increasing the routing table and address sizes of each vertex. Moreover, it is more practical to design name-independent compact routing scheme, i.e. without designing special address for each vertex.

The min-max risk model and min-sum risk model presented in Chapter 5 for Prize Collecting Steiner Tree problem with interval data in class of graphs of treewidth at most 2 have been already used for the shortest path problem [Hu10] and minimum spanning tree problem with interval data in general graphs[CHH09]. It is interesting to classify the class of graph problem $\mathcal{P}$ with interval data on edges and vertices, which satisfies that, if $\mathcal{P}$ with deterministic data can be solved in polynomial time, then $\mathcal{P}$ with interval data under the two risk models can be solved in polynomial time. Moreover, if $\mathcal{P}$ with deterministic data is NP-hard, how can we approximate the $\mathcal{P}$ with interval data under the two risk models?

For the gathering problem in grid network with $d_I = 0$, we have presented a linear algorithm with at most one step more than the optimal solution. But it is still open to determining the complexity of find the optimal solution in this case. Another generalization of our work on this problem is to design additive approximation algorithm for $d_I \geq 3$.

In the rest of this section I share with you some long term perspectives.

Due to the applications of tree decompositions of bounded width, the computation of treewidth is of great significance. As we mentioned, the best constant approximation

algorithm presented in [BDD+13] is linear in n and single exponential in the treewidth. But this algorithm is mainly of theoretical importance, because the base c of the single exponential function on the treewidth is huge. It is an interesting challenge to design more efficient and practical approximation algorithm for treewidth.

A *multiplicative (resp. additive) t-spanner* of a graph $G = (V,E)$ is a spanning subgraph H of G such that the distance between any pair of vertices in H is at most t times (resp. plus) their distance in G. This notion is introduced by Peleg and Ullman in [PU87]. For any connected graph G, a sparsest t-spanner, i.e. with minimum number of edges, can be a spanning tree of G. Among plenty of open questions about the t-spanner problems, in [DAA13] Dragan and Abu-Ata proposed an interesting open problem to find a necessary and sufficient conditions under which a graph admits a multiplicative or additive t-spanner of treewidth at most k.

Tree decompositions have played an important role in proving the Graph Minor Theorem, which says that finite graphs are *well-quasi-ordered*[1] by the minor relation. Another famous conjecture about well-quasi-ordering, Nash-Williams' strong immersion conjecture [NW65], says that finite graphs are well-quasi-ordered by the *strong immersion*[2] relation. I would like to learn more about well-quasi-ordering in the future, since it will be interesting to work on the Nash-Williams' strong immersion conjecture; and explore some new tools, even more powerful than tree decompositions, for solving many graph problems.

To conclude, I am only at the early stage of my research life and I will continue to study interesting problems in order to design beautiful algorithms.

[1] A *quasi-ordering* $\preceq$ on a set S is a reflexive and transitive binary relation. It is a *well-quasi-ordering* if for every infinite sequences $s_1, s_2, \ldots$ in S, there exist $i < j$ such that $s_i \preceq s_j$.

[2] Given two graphs G, H, a *strong immersion* of H in G is a function α with domain $V(H) \cup E(H)$, such that

- $\alpha(v) \in V(G)$ for all $v \in V(H)$, and $\alpha(u) \neq \alpha(v)$ for all distinct $u, v \in V(H)$;
- for each edge $e \in E(H)$, if e has distinct ends u, v, then $\alpha(e)$ is a path of G with ends $\alpha(u), \alpha(v)$, and if e is a loop incident with a vertex v then $\alpha(e)$ is a circuit of G with $\alpha(v) \in V(\alpha(e))$;
- for all distinct $e, f \in E(H)$, $E(\alpha(e) \cap \alpha(f) = \emptyset$;
- for all $v \in V(H)$ and $e \in E(H)$, if e is not incident with v in H then $\alpha(v) \notin V(\alpha(e))$.

Without the last condition, it is called *weak immersion*. In [RS10], Robertson and Seymour already proved that finite graphs are well-quasi-ordered by the weak immersion relation.

Bibliography

My bibliography

— International journals —

[j-AMCC+14] E. Alvarez-Miranda, A. Candia, X. Chen, X. Hu, and B. Li. Risk models for the prize collecting steiner tree problems with interval data. *Acta Mathematicae Applicatae Sinica*, 30(1):1–26, 2014.

[j-KLNS14] A. Kosowski, B. Li, N. Nisse, and K. Suchan. k-chordal graphs: From cops and robber to compact routing via treewidth. *Algorithmica*, pages 1–20, jan 2014.

— International conferences —

[c-AMCC+10] E. Alvarez-Miranda, A. Candia, X. Chen, X. Hu, and B. Li. Efficient algorithms for the prize collecting steiner tree problems with interval data. In B. Chen, editor, *Algorithmic Aspects in Information and Management*, volume 6124 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin Heidelberg, 2010.

[c-KLNS12] A. Kosowski, B. Li, N. Nisse, and K. Suchan. k-chordal graphs: From cops and robber to compact routing via treewidth. In *Automata, Languages, and Programming*, volume 7392 of *Lecture Notes in Computer Science*, pages 610–622. Springer Berlin Heidelberg, 2012.

[c-LMN14] B. Li, F. Z. Moataz, and N. Nisse. Minimum size tree-decompositions. In *9th International colloquium on graph theory and combinatorics (ICGT)*, Grenoble, France, 2014.

— Submitted papers —

[s-BLN+13] J.-C. Bermond, B. Li, N. Nisse, H. Rivano, and M.-L. Yu. Data Gathering and Personalized Broadcasting in Radio Grids with Interferences. Technical Report RR-8218, INRIA, 2013. Submitted to Theor. Comput. Sci.

References

[AAD14] M. Abu-Ata and F. F. Dragan. Metric tree-like structures in real-life networks: an empirical study. *CoRR*, abs/1402.3364, 2014.

[ABV08] H. Aissi, C. Bazgan, and D. Vanderpooten. Complexity of the min-max (regret) versions of min cut problems. *Discrete Optimization*, 5(1):66–73, February 2008.

[ACP87] S. Arnborg, D. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

[ACPS93] S. Arnborg, B. Courcelle, A. Proskurowski, and D. Seese. An algebraic theory of graph reduction. *J. ACM*, 40(5):1134–1164, November 1993.

[AF84] M. Aigner and M. Fromme. A game of cops and robbers. *Discrete Applied Mathematics*, 8(1):1–12, 1984.

[AG06] I. Abraham and C. Gavoille. Object location using path separators. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 188–197. ACM, 2006.

[AGGM06] I. Abraham, C. Gavoille, A. Goldberg, and D. Malkhi. Routing in networks with low doubling dimension. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 75–75, 2006.

[AGM04] I. Abraham, C. Gavoille, and D. Malkhi. Routing with improved communication-space trade-off. In R. Guerraoui, editor, *Distributed Computing*, volume 3274 of *Lecture Notes in Computer Science*, pages 305–319. Springer Berlin Heidelberg, 2004.

[AGM05] I. Abraham, C. Gavoille, and D. Malkhi. Compact routing for graphs excluding a fixed minor. In *Distributed Computing*, volume 3724 of *Lecture Notes in Computer Science*, pages 442–456. Springer Berlin Heidelberg, 2005.

[AGM06] I. Abraham, C. Gavoille, and D. Malkhi. On space-stretch trade-offs: Lower bounds. In *Proceedings of the Eighteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '06, pages 207–216. ACM, 2006.

[AGM+08] I. Abraham, C. Gavoille, D. Malkhi, N. Nisan, and M. Thorup. Compact name-independent routing with minimum stretch. *ACM Trans. Algorithms*, 4(3):37:1–37:12, July 2008.

[AJ13] A. G. Aksoy and S. Jin. The apple doesn't fall far from the (metric) tree: The equivalence of definitions. 2013.

[AJB99] R. Albert, H. Jeong, and A.-L. Barabási. Internet: Diameter of the world-wide web. *Nature*, 401:130–131, 1999.

[AL04] I. Averbakh and V. Lebedev. Interval data minmax regret network optimization problems. *Discrete Appl. Math.*, 138(3):289–301, April 2004.

[AM05] I. Abraham and D. Malkhi. Name independent routing for growth bounded networks. In *Proceedings of the Seventeenth Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA '05, pages 49–55. ACM, 2005.

[Ami10] E. Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

[And86] T. Andreae. On a pursuit game played on graphs for which a minor is excluded. *Journal of Combinatorial Theory, Series B*, 41(1):37–47, 1986.

[AP86] S. Arnborg and A. Proskurowski. Characterization and recognition of partial 3-trees. *SIAM J. Algebraic Discrete Methods*, 7(2):305–314, April 1986.

[AP89] S. Arnborg and A. Proskurowski. Linear time algorithms for np-hard problems restricted to partial k-trees. *Discrete Applied Mathematics*, 23(1):11 – 24, 1989.

[APW12] P. Austrin, T. Pitassi, and Y. Wu. Inapproximability of treewidth, one-shot pebbling, and related layout problems. In A. Gupta, K. Jansen, J. Rolim, and R. Servedio, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 7408 of *Lecture Notes in Computer Science*, pages 13–24. Springer Berlin Heidelberg, 2012.

[AVH04] I. D. Aron and P. Van Hentenryck. On the complexity of the robust spanning tree problem with interval data. *Oper. Res. Lett.*, 32(1):36–40, January 2004.

[BA99] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[Bal89] E. Balas. The prize collecting traveling salesman problem. *Networks*, 19(6):621–636, 1989.

[BB05] E. H. Bachoore and H. L. Bodlaender. New upper bound heuristics for treewidth. In *4th InternationalWorkshop on Experimental and Efficient Algorithms (WEA)*, volume 3503 of *Lecture Notes in Computer Science*, pages 216–227. Springer, 2005.

[BBBB+11] M. Bailly-Bechet, C. Borgs, A. Braunstein, J. Chayes, A. Dagkessamanskaia, J.-M. Francois, and R. Zecchina. Finding undetected protein associations in cell signaling by belief propagation. *Proceedings of the National Academy of Sciences*, 108(2):882–887, 2011.

[BBBZ09] M. Bailly-Bechet, A. Braunstein, and R. Zecchina. A prize-collecting steiner tree approach for transduction network inference. In P. Degano and R. Gorrieri, editors, *Computational Methods in Systems Biology*, volume 5688 of *Lecture Notes in Computer Science*, pages 83–95. Springer Berlin Heidelberg, 2009.

[BBS05] P. Bertin, J.-F. Bresse, and B. L. Sage. Accès haut débit en zone rurale: une solution "ad hoc". *France Telecom R&D*, 22:16–18, 2005.

[BC03] H. Bandelt and V. Chepoi. 1-hyperbolic graphs. *SIAM Journal on Discrete Mathematics*, 16(2):323–334, 2003.

[BCKN13] H. L. Bodlaender, M. Cygan, S. Kratsch, and J. Nederlof. Deterministic single exponential time algorithms for connectivity problems parameterized by treewidth. In *Automata, Languages, and Programming*, volume 7965 of *Lecture Notes in Computer Science*, pages 196–207. Springer Berlin Heidelberg, 2013.

[BCY09] J.-C. Bermond, R. Correa, and M.-L. Yu. Optimal gathering protocols on paths under interference constraints. *Discrete Mathematics*, 309(18):5574–5587, September 2009.

[BDD+13] H. L. Bodlaender, P. Drange, M. Dregi, F. Fomin, D. Lokshtanov, and M. Pilipczuk. An $o(c^k n)$ 5-approximation algorithm for treewidth. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 499–508, Oct 2013.

[BF96] H. L. Bodlaender and B. Fluiter. Reduction algorithms for constructing solutions in graphs with small treewidth. In J.-Y. Cai and C. Wong, editors, *Computing and Combinatorics*, volume 1090 of *Lecture Notes in Computer Science*, pages 199–208. Springer Berlin Heidelberg, 1996.

[BGHK91] H. L. Bodlaender, J. R. Gilbert, H. Hafsteinsson, and T. Kloks. Approximating treewidth, pathwidth, and minimum elimination tree height. *J. Algorithms*, 18:238–255, 1991.

[BGK+06] J.-C. Bermond, J. Galtier, R. Klasing, N. Morales, and S. Pérennes. Hardness and approximation of gathering in static radio networks. *Parallel Processing Letters*, 16(2):165–183, 2006.

[BGP+11] J.-C. Bermond, L. Gargano, S. Pérennes, A. Rescigno, and U. Vaccaro. Optimal time data gathering in wireless networks with omni-directional

antennas. In *SIROCCO 2011*, volume 6796 of *Lecture Notes in Computer Science*, pages 306–317, Gdansk, Poland, June 2011. Springer-Verlag.

[BGR10] J.-C. Bermond, L. Gargano, and A. A. Rescigno. Gathering with minimum completion time in sensor tree networks. *Journal of interconnection networks*, 11(1-2):1–33, 2010.

[BH98] H. L. Bodlaender and T. Hagerup. Tree decompositions of small diameter. In *Mathematical Foundations of Computer Science 1998*, volume 1450 of *Lecture Notes in Computer Science*, pages 702–712. Springer Berlin Heidelberg, 1998.

[BK91] H. L. Bodlaender and T. Kloks. Better algorithms for the pathwidth and treewidth of graphs. In J. Albert, B. Monien, and M. Artalejo, editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 544–555. Springer Berlin Heidelberg, 1991.

[BK07] H. L. Bodlaender and A. M. C. A. Koster. On the maximum cardinality search lower bound for treewidth. *Discrete Applied Mathematics*, 155(11):1348–1372, 2007.

[BK10] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations i. upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.

[BK11] H. L. Bodlaender and A. M. C. A. Koster. Treewidth computations ii. lower bounds. *Inf. Comput.*, 209(7):1103–1119, 2011.

[BKK+10] V. Bonifaci, R. Klasing, P. Korteweg, A. Marchetti-Spaccamela, and L. Stougie. Data gathering in wireless networks. In A.Koster and X. Munoz, editors, *Graphs and Algorithms in Communication Networks*, pages 357–377. Springer Monograph, 2010.

[BKSS08] V. Bonifaci, P. Korteweg, A. M. Spaccamela, and L. Stougie. An approximation algorithm for the wireless gathering problem. *Operations Research Letters*, 36(5):605–608, 2008.

[BM93] H. L. Bodlaender and R. Mhring. The pathwidth and treewidth of cographs. *SIAM Journal on Discrete Mathematics*, 6(2):181–188, 1993.

[BM08] A. Bondy and U. Murty. *Graph Theory*. Graduate Texts in Mathematics. Springer, 2008.

[BN11] A. Bonato and R. Nowakovski. *The game of Cops and Robber on Graphs*. American Math. Soc., 2011.

[BNRR09] J.-C. Bermond, N. Nisse, P. Reyes, and H. Rivano. Minimum delay data gathering in radio networks. In *Proceedings of the 8th international conference on Ad Hoc Networks and Wireless (AdHoc-Now)*, volume 5793 of *Lecture Notes in Computer Science*, pages 69–82. Springer, 2009.

[Bod93] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–22, 1993.

[Bod96] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, December 1996.

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[Bod07] H. L. Bodlaender. Treewidth: Structure and algorithms. In G. Prencipe and S. Zaks, editors, *Structural Information and Communication Complexity*, volume 4474 of *Lecture Notes in Computer Science*, pages 11–25. Springer Berlin Heidelberg, 2007.

[BP12] J.-C. Bermond and J. Peters. Optimal gathering in radio grids with interference. *Theoretical Computer Science*, 457:10–26, October 2012.

[BS03] D. Bertsimas and M. Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1-3):49–71, 2003.

[BSK94] H. F. Beltran and D. Skorin-Kapov. On minimum cost isolated failure immune networks. *Telecommunication Systems*, 3(2):183–200, 1994.

[BT97] H. L. Bodlaender and D. M. Thilikos. Treewidth for graphs with small chordality. *Discrete Applied Mathematics*, 79(1-3):45–61, 1997.

[BT01] V. Bouchitté and I. Todinca. Treewidth and minimum fill-in: Grouping the minimal separators. *SIAM J. Comput.*, 31(1):212–232, January 2001.

[BT02] V. Bouchitté and I. Todinca. Listing all potential maximal cliques of a graph. *Theoretical Computer Science*, 276(1-2):17 – 32, 2002.

[Bun74] P. Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9(3):205 – 212, 1974.

[BY10] J.-C. Bermond and M.-L. Yu. Optimal gathering algorithms in multi-hop radio tree networks with interferences. *Ad Hoc and Sensor Wireless Networks*, 9(1-2):109–128, 2010.

[CC13] C. Chekuri and J. Chuzhoy. Polynomial bounds for the grid-minor theorem. *CoRR*, abs/1305.6577, 2013.

[CCC06] L. Cai, S. M. Chan, and S. O. Chan. Random separation: a new method for solving fixed-cardinality optimization problems. In *Proceedings 2nd International Workshop on Parameterized and Exact Computation, IWPEC 2006*, pages 239–250. Springer-Verlag, 2006.

[CCNV11] J. Chalopin, V. Chepoi, N. Nisse, and Y. Vaxs. Cop and robber games when the robber can hide and ride. *SIAM Journal on Discrete Mathematics*, 25(1):333–359, 2011.

[CD14] D. Coudert and G. Ducoffe. On the recognition of C_4-free and $1/2$-hyperbolic graphs. Research Report RR-8458, INRIA, January 2014.

[CDE+08] V. Chepoi, F. Dragan, B. Estellon, M. Habib, and Y. Vaxès. Diameters, centers, and approximating trees of delta-hyperbolicgeodesic spaces and graphs. In *Proceedings of the Twenty-fourth Annual Symposium on Computational Geometry*, SCG '08, pages 59–68. ACM, 2008.

[CDE+12] V. Chepoi, F. Dragan, B. Estellon, M. Habib, Y. Vaxs, and Y. Xiang. Additive spanners and distance and routing labeling schemes for hyperbolic graphs. *Algorithmica*, 62(3-4):713–732, 2012.

[CF07] Y. Chen and J. Flum. On parameterized path and chordless path problems. In *Computational Complexity, 2007. CCC '07. Twenty-Second Annual IEEE Conference on*, pages 250–263, 2007.

[CGH75] E. Cockayne, S. Goodman, and S. Hedetniemi. A linear algorithm for the domination number of a tree. *Information Processing Letters*, 4(2):41 – 44, 1975.

[CHH09] X. Chen, J. Hu, and X. Hu. A polynomial solvable minimum risk spanning tree problem with interval data. *European Journal of Operational Research*, 198(1):43–46, October 2009.

[Cis14] Cisco visual networking index: Global mobile data traffic forecast update, 2013-2018, 2014.

[CMZ11] M. Chimani, P. Mutzel, and B. Zey. Improved steiner tree algorithms for bounded treewidth. In C. Iliopoulos and W. Smyth, editors, *Combinatorial Algorithms*, volume 7056 of *Lecture Notes in Computer Science*, pages 374–386. Springer Berlin Heidelberg, 2011.

[CN05] N. E. Clarke and R. J. Nowakowski. Tandem-win graphs. *Discrete Mathematics*, 299(1-3):56–64, 2005.

[Cou90] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Information and Computation*, 85(1):12 – 75, 1990.

[CSTW09] W. Chen, C. Sommer, S.-H. Teng, and Y. Wang. Compact routing in power-law graphs. In *Distributed Computing*, volume 5805 of *Lecture Notes in Computer Science*, pages 379–391. Springer Berlin Heidelberg, 2009.

[CSTW12] W. Chen, C. Sommer, S.-H. Teng, and Y. Wang. A compact routing scheme and approximate distance oracle for power-law graphs. *ACM Transactions on Algorithms*, 9(1):4, 2012.

[DAA13] F. F. Dragan and M. Abu-Ata. Collective additive tree spanners of bounded tree-breadth graphs with generalizations and consequences. In P. Emde Boas, F. Groen, G. Italiano, J. Nawrocki, and H. Sack, editors, *SOFSEM 2013: Theory and Practice of Computer Science*, volume 7741 of *Lecture Notes in Computer Science*, pages 194–206. Springer Berlin Heidelberg, 2013.

[DFHT05] E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and h-minor-free graphs. *J. ACM*, 52(6):866–893, November 2005.

[DFT12] F. Dorn, F. V. Fomin, and D. M. Thilikos. Catalan structures and dynamic programming in h-minor-free graphs. *J. Comput. Syst. Sci.*, 78(5):1606–1622, September 2012.

[DG04] Y. Dourisboure and C. Gavoille. Sparse additive spanners for bounded tree-length graphs. In *Structural Information and Communication Complexity*, volume 3104 of *Lecture Notes in Computer Science*, pages 123–137. Springer Berlin Heidelberg, 2004.

[DG07] Y. Dourisboure and C. Gavoille. Tree-decompositions with bags of small diameter. *Discrete Mathematics*, 307(16):2008–2029, 2007.

[DH08a] E. D. Demaine and M. Hajiaghayi. The bidimensionality theory and its algorithmic applications. *Comput. J.*, 51(3):292–302, 2008.

[DH08b] E. D. Demaine and M. Hajiaghayi. Linearity of grid minors in treewidth with applications through bidimensionality. *Combinatorica*, 28(1):19–36, 2008.

[DK11] F. F. Dragan and E. Kohler. An approximation algorithm for the tree t-spanner problem on unweighted graphs via generalized chordal graphs. In L. Goldberg, K. Jansen, R. Ravi, and J. D. Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 6845 of *Lecture Notes in Computer Science*, pages 171–183. Springer Berlin Heidelberg, 2011.

[DKR+08] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Muller. Identifying functional modules in protein-protein interaction networks: an integrated exact approach. In *ISMB*, pages 223–231, 2008.

[DKZ13] D. Dereniowski, W. Kubiak, and Y. Zwols. Minimum length path decompositions. *CoRR*, abs/1302.2788, 2013.

[dMSV11] F. de Montgolfier, M. Soto, and L. Viennot. Treewidth and hyperbolicity of the internet. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 25–32, 2011.

[DN66] D. E. Daykin and C. P. Ng. Algorithms for generalized stability numbers of tree graphs. *Journal of the Australian Mathematical Society*, 6:89–100, 2 1966.

[Dou05] Y. Dourisboure. Compact routing schemes for generalized chordal graphs. *J. of Graph Alg. and App*, 9(2):277–297, 2005.

[DPBF10] F. Dorn, E. Penninkx, H. L. Bodlaender, and F. V. Fomin. Efficient exact algorithms on planar graphs: Exploiting sphere cut decompositions. *Algorithmica*, 58(3):790–810, 2010.

[FFF99] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *annual conference of the Special Interest Group on Data Communication (SIGCOMM)*, pages 251–262. ACM, 1999.

[FFFdP07] P. Feofiloff, C. G. Fernandes, C. E. Ferreira, and J. C. de Pina. Primal-dual approximation algorithms for the prize-collecting steiner tree problem. *Information Processing Letters*, 103(5):195 – 202, 2007.

[FFM04] C. Florens, M. Franceschetti, and R. McEliece. Lower bounds on data collection time in sensory networks. *IEEE Journal on Selected Areas in Communications*, 22(6):1110–1120, 2004.

[FG01] P. Fraigniaud and C. Gavoille. Routing in trees. In *Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 757–772. Springer Berlin Heidelberg, 2001.

[FHL08] U. Feige, M. T. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, May 2008.

[fIDAC] T. C. A. for Internet Data Analysis (CAIDA). The CAIDA AS relationships dataset.

[FIV12] H. Fournier, A. Ismail, and A. Vigneron. Computing the gromov hyperbolicity of a discrete metric space. *CoRR*, abs/1210.3323, 2012.

[FKTV08] F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38(3):1058–1079, July 2008.

[Fra87] P. Frankl. Cops and robbers in graphs with large girth and cayley graphs. *Discrete Applied Mathematics*, 17(3):301–305, 1987.

[FV12] F. V. Fomin and Y. Villanger. Treewidth computation and extremal combinatorics. *Combinatorica*, 32(3):289–308, 2012.

[Gav74] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47 – 56, 1974.

[GGJK78] M. R. Garey, R. L. Graham, D. S. Johnson, and D. E. Knuth. Complexity results for bandwidth minimization. *SIAM Journal on Applied Mathematics*, 34(3):477–495, 1978.

[GJ77] M. R. Garey and D. S. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4):pp. 826–834, 1977.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[GJT76] M. Garey, D. Johnson, and R. Tarjan. The planar hamiltonian circuit problem is np-complete. *SIAM Journal on Computing*, 5(4):704–714, 1976.

[GKL03] A. Gupta, R. Krauthgamer, and J. R. Lee. Bounded geometries, fractals, and low-distortion embeddings. In *44th Symposium on Foundations of Computer Science (FOCS)*, pages 534–543. IEEE Computer Society, 2003.

[GP96] C. Gavoille and S. Pérennès. Memory requirement for routing in distributed networks. In *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '96, pages 125–133, New York, NY, USA, 1996. ACM.

[GPRR08] C. Gomes, S. Perennes, P. Reyes, and H. Rivano. Bandwidth allocation in radio grid networks. In *Algotel'08*, Saint Malo, May 2008.

[GR09] L. Gargano and A. Rescigno. Optimally fast data gathering in sensor networks. *Discrete Applied Mathematics*, 157:1858–1872, 2009.

[Gro87] M. Gromov. Hyperbolic groups. *Essays in Group Theory*, 8:75–263, 1987.

[Gus93] J. Gusted. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233 – 248, 1993.

[GW95] M. Goemans and D. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, 24(2):296–317, 1995.

[Heg06] P. Heggernes. Minimal triangulations of graphs: A survey. *Discrete Mathematics*, 306(3):297 – 317, 2006. Minimal Separation and Minimal Triangulation.

[HR92] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[HT73] J. Hopcroft and R. Tarjan. Algorithm 447: Efficient algorithms for graph manipulation. *Commun. ACM*, 16(6):372–378, June 1973.

[Hu10] J. Hu. Minimizing maximum risk for fair network connection with interval data. *Acta Mathematicae Applicatae Sinica, English Series*, 26(1):33–40, 2010.

[iKK12] K. ichi Kawarabayashi and Y. Kobayashi. Linear min-max relation between the treewidth of h-minor-free graphs and its largest grid. In C. Durr and T. Wilke, editors, *STACS*, volume 14 of *LIPIcs*, pages 278–289. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2012.

[iKKR12] K. ichi Kawarabayashi, Y. Kobayashi, and B. Reed. The disjoint paths problem in quadratic time. *Journal of Combinatorial Theory, Series B*, 102(2):424 – 435, 2012.

[IPS82] A. Itai, C. H. Papadimitriou, and J. L. Szwarcfiter. Hamilton Paths in Grid Graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[JMP00] D. S. Johnson, M. Minkoff, and S. Phillips. The prize collecting steiner tree problem: Theory and practice. In *Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '00, pages 760–769, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[Kar72] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, and J. D. Bohlinger, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Springer US, 1972.

[KFY04] D. V. Krioukov, K. R. Fall, and X. Yang. Compact routing on internet-like graphs. In *23th IEEE International Conference on Computer Communications (INFOCOM)*, 2004.

[KK95] T. Kloks and D. Kratsch. Treewidth of chordal bipartite graphs. *Journal of Algorithms*, 19(2):266–281, 1995.

[KK09] Y. Kobayashi and K.-i. Kawarabayashi. Algorithms for finding an induced cycle in planar graphs and bounded genus graphs. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '09, pages 1146–1155, 2009.

[KLNP09] R. Klasing, Z. Lotker, A. Navarra, and S. Pérennes. From balls and bins to points and vertices. *Algorithmic Operations Research (AlgOR)*, 4(2):133–143, 2009.

[KPBnV09] D. Krioukov, F. Papadopoulos, M. Boguñá, and A. Vahdat. Greedy forwarding in scale-free networks embedded in hyperbolic metric spaces.

SIGMETRICS Performance Evaluation Review, 37(2):15–17, October 2009.

[KRX06] G. Konjevod, A. W. Richa, and D. Xia. Optimal-stretch name-independent compact routing in doubling metrics. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC '06, pages 198–207. ACM, 2006.

[KV90] D. J. Kleitman and R. Vohra. Computing the bandwidth of interval graphs. *SIAM J. Discrete Math.*, 3(3):373–375, 1990.

[KY10] P. Kouvelis and G. Yu. *Robust Discrete Optimization and Its Applications*. Nonconvex Optimization and Its Applications. Springer, 2010.

[KZ06] A. Kasperski and P. Zielinski. The robust shortest path problem in series-parallel multidigraphs with interval data. *Operations Research Letters*, 34(1):69 – 76, 2006.

[KZ10] A. Kasperski and P. Zielinski. Minmax regret approach and optimality evaluation in combinatorial optimization problems with interval and fuzzy weights. *European Journal of Operational Research*, 200:680–687, 2010.

[LA91] J. Lagergren and S. Arnborg. Finding minimal forbidden minors using a finite congruence. In J. Albert, B. Monien, and M. Artalejo, editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 532–543. Springer Berlin Heidelberg, 1991.

[Lag96] J. Lagergren. Efficient parallel algorithms for graphs of bounded tree-width. *Journal of Algorithms*, 20(1):20 – 44, 1996.

[Lok10] D. Lokshtanov. On the complexity of computing treelength. *Discrete Appl. Math.*, 158(7):820–827, April 2010.

[LP12] L. Lu and X. Peng. On meyniel's conjecture of the cop number. *Journal of Graph Theory*, 71(2):192–205, 2012.

[LWP+06] I. Ljubic, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Mathematical Programming*, 105(2-3):427–449, 2006.

[Meg79] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4(4):414–424, 1979.

[Mez11] M. Mezzini. Fast minimal triangulation algorithm using minimum degree criterion. *Theoretical Computer Science*, 412(29):3775 – 3787, 2011.

[Mon86] B. Monien. The bandwidth minimization problem for caterpillars with hair length 3 is np-complete. *SIAM J. Algebraic Discrete Methods*, 7(4):505–512, October 1986.

[MS89] C. L. Monma and D. F. Shallcross. Methods for designing communications networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531–431, 1989.

[MT91] J. Matousek and R. Thomas. Algorithms finding tree-decompositions of graphs. *Journal of Algorithms*, 12(1):1 – 22, 1991.

[NRS12] N. Nisse, I. Rapaport, and K. Suchan. Distributed computing of efficient routing schemes in generalized chordal graphs. *Theoretical Computer Science*, 444(0):17–27, 2012.

[NW65] C. S. J. A. Nash-Williams. On well-quasi-ordering infinite trees. *Mathematical Proceedings of the Cambridge Philosophical Society*, 61:697–720, 7 1965.

[NW83] R. Nowakowski and P. Winkler. Vertex-to-vertex pursuit in a graph. *Discrete Mathematics*, 43(2-3):235–239, 1983.

[OP09] T. Opsahl and P. Panzarasa. Clustering in weighted networks. *Social Networks*, 31(2):155–163, 2009.

[PDL10] A. Prodon, S. DeNegre, and T. M. Liebling. Locating leak detecting sensors in a water distribution network by solving prize-collecting steiner arborescence problems. *Mathematical Programmming*, 124(1-2):119–141, 2010.

[PU87] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, pages 77–85, New York, NY, USA, 1987. ACM.

[Qui85] A. Quilliot. A short note about pursuit games played on a graph with a given genus. *Journal of Combinatorial Theory, Series B*, 38(1):89–92, 1985.

[Rag04] S. Raghavan. Low-connectivity network design on series-parallel graphs. *Networks*, 43(3):163–176, 2004.

[Ree92] B. A. Reed. Finding approximate separators and computing tree width quickly. In *Proceedings of the Twenty-fourth Annual ACM Symposium on Theory of Computing*, STOC '92, pages 221–228, New York, NY, USA, 1992. ACM.

[Ros74] D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7(3-4):317 – 322, 1974.

[RS83] N. Robertson and P. Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39 – 61, 1983.

[RS84] N. Robertson and P. D. Seymour. Graph minors. iii. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[RS86a] N. Robertson and P. D. Seymour. Graph minors. v. excluding a planar graph. *J. Comb. Theory, Ser. B*, 41(1):92–114, 1986.

[RS86b] N. Robertson and P. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322, 1986.

[RS90] N. Robertson and P. D. Seymour. Graph minors. iv. tree-width and well-quasi-ordering. *J. Comb. Theory, Ser. B*, 48(2):227–254, 1990.

[RS91] N. Robertson and P. Seymour. Graph minors. x. obstructions to tree-decomposition. *Journal of Combinatorial Theory, Series B*, 52(2):153 – 190, 1991.

[RS95] N. Robertson and P. Seymour. Graph minors .xiii. the disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65 – 110, 1995.

[RS04] N. Robertson and P. D. Seymour. Graph minors. xx. wagner's conjecture. *J. Comb. Theory, Ser. B*, 92(2):325–357, 2004.

[RS07] Y. Revah and M. Segal. Improved algorithms for data-gathering time in sensor networks ii: Ring, tree and grid topologies. In *Networking and Services, 2007. ICNS. Third International Conference on*, page 46, 2007.

[RS08] Y. Revah and M. Segal. Improved bounds for data-gathering time in sensor networks. *Computer Communications*, 31(17):4026–4034, 2008.

[RS10] N. Robertson and P. Seymour. Graph minors xxiii. nash-williams' immersion conjecture. *Journal of Combinatorial Theory, Series B*, 100(2):181 – 205, 2010.

[RSS11] S. Ruzika, H. Sperber, and M. Steiner. Earliest arrival flows on series-parallel graphs. *Networks*, 57(2):169–173, 2011.

[RST94] N. Robertson, P. D. Seymour, and R. Thomas. Quickly excluding a planar graph. *J. Comb. Theory, Ser. B*, 62(2):323–348, 1994.

[RSY10] Y. Revah, M. Segal, and L. Yedidsion. Real-time data gathering in sensor networks. *Discrete Applied Mathematics*, 158(5):543–550, 2010.

[San96] D. P. Sanders. On linear recognition of tree-width at most four. *SIAM J. Discret. Math.*, 9(1):101–117, 1996.

[Sch01] B. S. W. Schröder. The copnumber of a graph is bounded by $\lfloor \frac{3}{2} genus(G) \rfloor + 3$. In *Categorical Perspectives*, Trends in Mathematics, pages 243–263. Birkhuser Boston, 2001.

[SFFF03] G. Siganos, M. Faloutsos, P. Faloutsos, and C. Faloutsos. Power laws and the as-level internet topology. *IEEE/ACM Trans. Netw.*, 11(4):514–524, 2003.

[SS11] A. Scott and B. Sudakov. A bound for the cops and robbers problem. *SIAM Journal on Discrete Mathematics*, 25(3):1438–1442, 2011.

[SSR94] R. Sundaram, K. Singh, and C. Rangan. Treewidth of circular-arc graphs. *SIAM Journal on Discrete Mathematics*, 7(4):647–655, 1994.

[ST10] I. Sau and D. M. Thilikos. Subexponential parameterized algorithms for degree-constrained subgraph problems on planar graphs. *J. of Discrete Algorithms*, 8(3):330–338, September 2010.

[Sys79] M. M. Systo. Characterizations of outerplanar graphs. *Discrete Mathematics*, 26(1):47 – 53, 1979.

[TaP97] J. A. Telle and andrzej Proskurowski. Algorithms for vertex partitioning problems on partial k-trees. *SIAM J. Disc. Math.*, 10:529–550, 1997.

[TZ05] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, January 2005.

[Ueh99] R. Uehara. Tractable and intractable problems on generalized chordal graphs. Technical Report COMP98-83, IEICE, 1999.

[WC83] J. A. Wald and C. J. Colbourn. Steiner trees, partial 2-trees, and minimum ifi networks. *Networks*, 13(2):159–167, 1983.

[WNC08] J.-B. Wang, C. Ng, and T. Cheng. Single-machine scheduling with deteriorating jobs under a series-parallel graph constraint. *Computers and Operations Research*, 35(8):2684 – 2693, 2008. Queues in Practice.

[WS98] D. J. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393(6684):440–442, 1998.

[WZ11] Y. Wu and C. Zhang. Hyperbolicity and chordality of a graph. *Electronic Journal of Combinatorics*, 18(1), 2011.

[YT94] C.-C. YEN and L. R. C. T. algorithm to solve the weighted perfect domination problem in series-parallel graphs. *European journal of operational research*, 73(1):192–198, 1994. eng.

[Zie04] P. Zielinski. The computational complexity of the relative robust shortest path problem with interval data. *European Journal of Operational Research*, 158(3):570 – 576, 2004.

[ZTG05] X. Zhu, B. Tang, and H. Gupta. Delay efficient data gathering in sensor network. In X. Jia, J. Wu, and Y. He, editors, *Proc. of MSN'05*, volume 3794 of *Lecture Notes in Computer Science*, pages 380–389. Springer-Verlag, 2005.