

MASTER

Triangle partition on graphs of bounded cutwidth upper- and lower bounds on algorithmic and communication complexity

van Heck, I.

Award date:
2018

[Link to publication](#)

Disclaimer

This document contains a student thesis (bachelor's or master's), as authored by a student at Eindhoven University of Technology. Student theses are made available in the TU/e repository upon obtaining the required degree. The grade received is not published on the document as presented in the repository. The required complexity or quality of research of student theses may vary by program, and the required minimum study period may vary in duration.

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain

Take down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Triangle Partition on graphs of bounded Cutwidth

*Upper- and lower bounds on algorithmic
and communication complexity*

Ivo van Heck

Supervisors:
Dr. Bart M. P. Jansen
Dr. Jesper Nederlof

Committee:
Dr. Jesper Nederlof
Dr. Bart M. P. Jansen
Prof. dr. Hans L. Bodlaender

version 1.2

Eindhoven, June 2018

Abstract

In the field of parameterized complexity the efficiency of algorithms is measured in terms of an explicit parameter instead of (only) the general concept of input size. This is particularly useful for difficult computational problems. For graph problems, three classical parameters are the *cutwidth*, *pathwidth* and *treewidth*, which are strongly related. Lokshtanov et al.[17] have shown for many graph problems parameterized by both treewidth and pathwidth, the most efficient algorithms known are in fact optimal, assuming the *Strong Exponential Time Hypothesis* (SETH) holds. However, their lower bounds leave the cutwidth out of scope.

As such, we investigate the behaviour of one of the problems studied by Lokshtanov et al., TRIANGLE PARTITION, when parameterized by cutwidth. Based on the approach taken by Lokshtanov et al., we show that assuming SETH, there cannot exist an $\varepsilon > 0$ such that TRIANGLE PARTITION can be solved in $\mathcal{O}^*((2 - \varepsilon)^{\text{ctw}(G)/2})$ time. This contrasts the results known for pathwidth and treewidth, which state TRIANGLE PARTITION cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^{\text{pw}(G)})$ and $\mathcal{O}^*((2 - \varepsilon)^{\text{tw}(G)})$ respectively. Our second result is an algorithm which exploits the relation between the cutwidth and pathwidth of graphs where each edge is contained in a triangle. Using a simple preprocessing algorithm and a dynamic programming algorithm on a path decomposition of the resulting graph, we find an upper bound of $\mathcal{O}^*(2^{\frac{3}{4}\text{ctw}(G)})$ time.

In an attempt to improve on this bound, we investigate a communication variant of TRIANGLE PARTITION, where the graph is divided between two players. Their goal is to solve the problem using a minimum amount of communication. The parameter we use for TRIANGLE PARTITION COMMUNICATION is the number of edges connecting the parts of the two players. By constructing a large family of graphs, each requiring a unique message in any communication protocol, we show a lower bound of $\Omega(\frac{3^{k/3}}{k})$ bits. Note the mismatch between this lower bound and the one found for the classic version, which suggests this approach is unlikely to prove fruitful with respect to finding an optimal algorithm for TRIANGLE PARTITION. Finally, we describe a communication algorithm based on a branching algorithm executed by both players, which leads to an upper bound of $\mathcal{O}(1.54369^k)$.

Contents

Contents	v
1 Introduction	1
2 Preliminaries	3
3 Lower Bound on algorithmic complexity	7
4 Upper Bound on algorithmic complexity	13
5 Lower Bound on communication complexity	17
6 Upper Bound on communication complexity	23
7 Conclusions	27
Bibliography	29

Chapter 1

Introduction

Many interesting graph problems have been shown to be NP-hard [9, 14]. One approach to designing algorithms for such problems is through parameterization, generally with the goal of finding an algorithm whose complexity is only superpolynomial in the chosen parameter (known as *fixed-parameter tractable algorithms*). To be precise, a parameterized problem is fixed-parameter tractable if every instance (I, k) can be solved in $f(k)|I|^{\mathcal{O}(1)}$, for an arbitrary function f . Generally speaking, for many graph problems there are numerous interesting parameters which allow such algorithms. Three classical parameters are the *cutwidth*, *pathwidth* and *treewidth* of a graph. Each of these parameters measures the interconnectedness of the graph in their own way. The relation between the parameters is well known; for any graph G the relation $\mathbf{ctw}(G) \geq \mathbf{pw}(G) \geq \mathbf{tw}(G)$ holds [16].

Recently it has been shown that the most efficient algorithms known for many problems on graphs parameterized by treewidth are optimal under the *Strong Exponential Time Hypothesis* (see theorem 2.1) [17]. The proof is stated in terms of pathwidth, which implies a lower bound on both the complexity of parameterization by pathwidth and by treewidth, but leaves cutwidth out of scope. Since then, more results have been found for graph problems parameterized by cutwidth. For some problems the lower bound found for pathwidth has been extended to cutwidth, such as DOMINATION SET and INDEPENDENT SET[2]. For other problems more efficient algorithms in terms of cutwidth have been found, most notably q -COLORING [12]. The contrast between the results for different graph problems motivates research into problems for which matching upper and lower bounds in terms of cutwidth are not yet known. As such, we investigate whether the additional structure of cutwidth over treewidth is of algorithmic use for the problems TRIANGLE PACKING and TRIANGLE PARTITION.

An instance I of TRIANGLE PACKING consists of a graph G and an integer p . The goal is to decide whether G contains p vertex-disjoint triangles. Specifically, I is a yes-instance if and only if there exists a set $S \subset \mathcal{P}(V(G))$ such that $|S| \geq p$ and all distinct $s, s' \in S$ satisfy $s \cap s' = \emptyset$, $|s| = 3$ and for all distinct $u, v \in s : \{u, v\} \in E(G)$.

TRIANGLE PARTITION is a special case of TRIANGLE PACKING where $p = \lfloor \frac{|V(G)|}{3} \rfloor$. As such, an instance of TRIANGLE PARTITION consists solely of a graph G . Because TRIANGLE PARTITION is a special case of TRIANGLE PACKING, any upper bound on the complexity of TRIANGLE PACKING also holds for TRIANGLE PARTITION, and any lower bound on the complexity of TRIANGLE PARTITION also holds for TRIANGLE PACKING.

TRIANGLE PARTITION is NP-complete on general graphs [9]. Inclusion-exclusion can be used to solve it in $\mathcal{O}^*(2^n)$ time and polynomial space on general graphs [4]. This can be improved upon to $\mathcal{O}^*(1.769^n)$ if one is allowed exponential space [15] and to $\mathcal{O}^*(1.496^n)$ if one is allowed a failure probability exponentially small in n [3]. The problem remains NP-complete when restricted to planar graphs [9], but for triangle packing approximation algorithms do exist [1]. When restricted to graphs of degree at most three, the problem can be solved in polynomial time [23]. However, when restricted to graphs of degree at most four, it is NP-complete again and it has even been shown it cannot be solved in subexponential time unless ETH fails [23]. However, very fast

exponential time ($\mathcal{O}^*(1.02445^n)$) algorithms do exist on this class of graphs [23].

For both TRIANGLE PACKING and TRIANGLE PARTITION, we will also consider a one-way communication variant. This means there are two players, Alice and Bob, who need solve a problem together. Both players receive some information and Alice needs to send a message to Bob such that Bob can solve the decision problem. The complexity of a protocol is measured in the size of the message Alice sends to Bob. Studying such variants has proved fruitful in the past, as the result in communication complexity could be used as a basis for a dynamic programming algorithm for the original version of the problem [12].

Specifically, an instance I of TRIANGLE PACKING COMMUNICATION consists of a graph G , an integer n and a set $A \subset V(G)$. Alice is given a graph $G_A = G[N_G[A]]$, Bob is given a graph $G_B = G[N_G[B]]$ where $B = V(G) \setminus A$. We denote set of vertices both players have access to by $\Delta = V(G_A) \cap V(G_B)$, which we will refer to as the boundary. For this problem, Bob needs to decide whether G contains n vertex-disjoint triangles. As before, TRIANGLE PARTITION COMMUNICATION is a special case of TRIANGLE PACKING COMMUNICATION where $n = \lfloor \frac{|V(G)|}{3} \rfloor$. Both these communication problems are parameterized by the number of edges in G where one of the vertices is contained in A and the other is not. Note that this is strongly related the concept of cutwidth, since this is precisely the cutwidth of the cut after the last vertex in A for any linear layout whose ordering contains all vertices of A before the vertices not in A .

Our first result is a lower bound on the complexity of TRIANGLE PARTITION, in which we show it cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^{\text{ctw}(G)/2})$ time for any $\varepsilon > 0$. We obtain this by analyzing the cutwidth of the graph constructed by Lokshantov et al.[17] as a reduction from CNF-SAT to TRIANGLE PARTITION. Note the contrast with the bound when parameterized by pathwidth, which is $\mathcal{O}^*(2^{\text{pw}(G)})$. This suggests that TRIANGLE PARTITION is a problem where the more restricted parameter of cutwidth allows for more efficient algorithms. As our second result, we show this is indeed the case by providing a $\mathcal{O}^*(2^{\frac{3}{4}\text{ctw}(G)})$ time algorithm. This algorithm is based on the observation that while in general there are graphs whose cutwidth and treewidth are equal, these graphs can be reduced to equivalent instances where the cutwidth and treewidth do significantly differ. As such, we can use dynamic programming on a tree decomposition of these reduced instances.

In an attempt to further improve upon this algorithm, we study a communication variant of TRIANGLE PARTITION. We prove a lower bound on the communication complexity by constructing a large family of graphs and argue why a unique message must be sent for each of these graphs. We define this family of graphs by constructing a graph for each monotone ternary boolean function. The idea behind this approach came from the bound on the number of monotone binary boolean functions obtained in [13]. Surprisingly, we found a lower bound of $\Omega^*(3^{k/3})$ for the communication complexity, which is larger than the lower bound we found for the computational complexity. Finally, we describe a communication protocol based on a branching algorithm, which provides an upper bound on the communication complexity of $\mathcal{O}^*(1.54369^k)$.

Chapter 2

Preliminaries

We use the shorthand $[n]$ for the set $\{1, 2, \dots, n\}$ and $[m..n]$ for $\{m, m + 1, \dots, n\}$.

Complexity Assumptions

When analyzing the complexity of exponential-time algorithms we use \mathcal{O}^* notation, which suppresses polynomial factors in both the total input size and the variables inside, in addition to constant factors. In the q -SAT problem one has to decide whether a given boolean formula, in conjunctive normal form with clauses containing at most q literals, has a satisfying assignment. As the hardness assumption for our lower bound we use the *Strong Exponential Time Hypothesis* (SETH for short). Intuitively, SETH states that no significant improvements can be made over exhaustive search for q -SAT for arbitrary q . As the name suggests, it is a (significantly) stronger variant of *Exponential Time Hypothesis* (ETH) which states that for fixed $q \geq 3$, q -SAT cannot be solved in sub-exponential time.

Theorem 2.1 (Strong Exponential Time Hypothesis [10, 11]). *For every $\varepsilon > 0$, there is a constant q such that q -SAT on n variables cannot be solved in $\mathcal{O}^*((2 - \varepsilon)^n)$.*

Both ETH and SETH are fairly popular conjectures for proving lower bounds on exact exponential time algorithms [18]. For example, Cygan et al. [7] show an equivalence between SETH and the assumption that a number of other NP-hard problems cannot be solved significantly better than by exhaustive search. Though there may not be a strong consensus that SETH holds, improving on lower bounds based on SETH is at least as hard improving our best known algorithms for a large number of problems.

Theorem 2.2 (Stirling's Approximation [20]). *For all $n > 0$: $\sqrt{2\pi n}^{n+\frac{1}{2}} e^{-n} \leq n! \leq e\sqrt{2\pi n}^{n+\frac{1}{2}} e^{-n}$.*

Graphs

Unless further specified, when we talk about a graph $G = (V, E)$ we mean a simple graph. Specifically, G is undirected, unweighted, and does not contain self loops or parallel edges. We denote the vertex set and edge set as $V(G)$ and $E(G)$ respectively. For $u, v \in V(G)$, we write the edge between u and v as the pair $\{u, v\}$. For $V' \subset V(G)$, the induced subgraph $G[V']$ is the graph $(V', E(G) \cup (V' \times V'))$. The open and closed neighborhood of a vertex $v \in V(G)$ are $N_G(v) = \{u \in V(G) \mid \{u, v\} \in E(G)\}$ and $N_G[v] = N_G(v) \cup \{v\}$ respectively. We extend this definition to sets of vertices $V' \subset V(G)$ as follows: $N_G(V') = \bigcup_{v \in V'} N_G(v) \setminus V'$ and $N_G[V'] = \bigcup_{v \in V'} N_G[v]$. For all these definitions we may drop the reference to G when there is no risk of confusion as to which graph is being considered.

A *linear layout* of a graph G is a bijection $\pi : V(G) \rightarrow \{1, \dots, |V(G)|\}$. The *cut* after vertex v is the set of all edges with one endpoint before or at v and the other endpoint after v . Formally, it is the set $\{\{u, w\} \in E(G) \mid \pi(u) \leq \pi(v) \wedge \pi(w) > \pi(v)\}$. An example is shown in figure 2.1. The number of edges in such a cut is referred to as the *width* of the cut. The *cutwidth* of a linear layout

π is defined as width of the largest cut of π and is denoted $\text{ctw}(\pi)$. The cutwidth of a graph G is the minimum cutwidth over all possible linear layouts of G and is denoted $\text{ctw}(G)$.

Testing if an arbitrary graph has cutwidth at most k is NP-hard [9], but FPT-algorithms for finding linear layouts of minimum cutwidth do exist [22]. Cutwidth has been extensively studied [22, 6, 5, 19] and is closely related to several other graph parameters [16, 5].

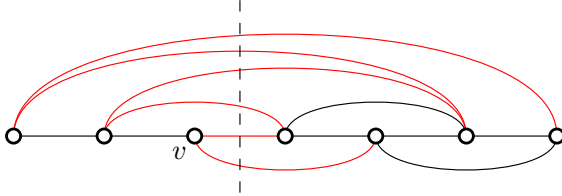


Figure 2.1: A visualization of the cut after v , containing all edges crossing the dotted line. The edges contained in the cut are highlighted in red. The linear layout is implicitly defined as left-to-right in the image.

A *path decomposition* of a graph G is pair (\mathcal{X}, P) where P is a path graph and $\mathcal{X} = \{X_i \mid i \in V(P)\}$ is a family of subsets of $V(G)$ (referred to as *bags*) satisfying the properties:

1. $\bigcup_{i \in V(P)} X_i = V(G)$, every vertex of G is contained in some bag
2. $\forall \{u, v\} \in E(G) : \exists i \in V(P) : \{u, v\} \subset X_i$, every edge of G is contained in some bag
3. $\forall v \in V(G)$ the set $\{i \mid v \in X_i\}$ induces a connected subgraph of P

The *pathwidth* of such a decomposition is $\max_{i \in V(P)} |X_i - 1|$. The pathwidth of G is the minimum width among all path decompositions of G . When analyzing the pathwidth of a graph, we will use an equivalent definition of pathwidth through *mixed search games*.

In a mixed search game, our goal is to decontaminate the graph. Initially, all edges of the graph are contaminated. We clear edges by either placing a cleaner on both endpoints of the edge, or by moving a cleaner over the contaminated edge. Edges become immediately recontaminated whenever there is a path from any contaminated edge to any cleared edge without any cleaners on the path. A search strategy is a sequence of moves of the following types:

1. placement of a new searcher on a vertex
2. removal of a searcher from a vertex
3. movement of a searcher from a vertex over an incident edge to the neighbouring vertex

A strategy is a winning strategy if after execution all edges are cleared. Takahashi, Ueno and Kajitani [21] showed the minimum number of searchers required for a winning strategy is equivalent to the pathwidth of a graph.

Branching Algorithms

One of the simplest and most commonly used techniques for designed parameterized algorithms is branching. A branching algorithm solves the problem by building a search tree. Whenever a decision has to be made, the branching algorithm splits the problem into multiple sub-problems, each representing a possible choice for the decision that had to be made. In order to argue correctness, one generally argues why some sequence of decisions made by the algorithm finds a solution, if it exists. In order to argue complexity, one generally bounds the size of the search tree as a function of the parameter.

In order to obtain the complexity of the branching algorithm, we can derive a *branching vector* for each of the rules that define the choices made by the algorithm. This branching vector has an entry for each subproblem defined by the rule, and the entry corresponds with the reduction in the parameter caused by this decision. When an algorithm is defined by multiple rules, which apply under different circumstances, it is common to simply analyze the algorithm in terms of the worst-case branching vector. By solving the recurrence implied by the branching vector, we can obtain a *branching number* associated with the rule [8, Chapter 3.2]. This branching number is the base of the exponent in the upper bound on the running time.

Ternary boolean functions

A *ternary boolean function* is any function of the form $f : \{0, 1, 2\}^n \rightarrow \{0, 1\}$. Such a function is *monotone* when $x \preceq x' \Rightarrow f(x) \leq f(x')$ where we define $x \preceq x'$ if and only if $\forall_{i \in [n]} x_i \leq x'_i$.

Proposition 1. *For any monotone ternary boolean function f there exists a formula ϕ of the form $\phi(x) = \bigwedge_{j \in [m]} \bigvee_{i \in [n]} (x_i > c_{i,j})$ with $c_{i,j} \in \{0, 1, 2\}$ such that $f(x) = 1$ if and only if $\phi(x)$ is satisfied.*

Proof. Let X be the set of all maximal vectors $x \in \{0, 1, 2\}^n$ such that $f(x) = 0$, that is all vectors $x \in X$ with $f(x) = 0$ such that there exists no x' with $f(x') = 0$ and $x \preceq x'$. If X is empty, f must be constant 1, and we can choose the empty formula ϕ (that is, a formula without any clauses). From now on, we assume X is non-empty. Since f is monotone, $f(x) = 0$ if and only if $x \leq x'$ for some $x' \in X$. We define $\phi(x) = \bigwedge_{x' \in X} \bigvee_{i \in [n]} (x_i > x'_i)$.

Suppose $f(x) = 1$, then there must not exist some $x' \in X$ with $x \leq x'$, which means for all $x' \in X$ there must be some $i \in [n]$ such that $x_i > x'_i$, which clearly implies all the clauses are satisfied and thus $\phi(x) = 1$.

Conversely, if $f(x) = 0$, there is some $x' \in X$ such that $x \leq x'$, which means for all $i \in [n]$ $x_i \leq x'_i$, which shows the clause corresponding to x' is not satisfied, thus $\phi(x) = 0$. \square

A vector $x \in \{0, 1, 2\}^n$ is *balanced* if and only if it is a permutation of the vector x^* with $x_i^* = i \pmod 3$. Let \mathcal{X} be the set of all balanced vectors $x \in \{0, 1, 2\}^n$. We can associate a monotone ternary boolean function f with each subset $X \subset \mathcal{X}$ by defining $f_X(x) = 1$ if and only if there exists an $x' \in X$ such that $x' \leq x$.

Proposition 2. *Let $X, X' \subset \mathcal{X}$ such that $X \neq X'$, then $f_X \neq f_{X'}$.*

Proof. Without loss of generality, let $x \in X$ and $x \notin X'$. Clearly, $f_X(x) = 1$ since trivially $x \leq x$. Since all $x' \in X'$ are permutations of each other, the only $x' \in X'$ with the property $x' \leq x$ is x itself. Since $x \notin X'$, there is no such $x' \in X'$, thus $f_{X'}(x) = 0$. Therefore $f_X \neq f_{X'}$. \square

Corollary 2.1. *Let \mathcal{F} be the set of all monotone n -dimensional ternary boolean functions. Then $\log_2 |\mathcal{F}| = \Omega\left(\frac{3^n}{n}\right)$.*

Proof. As a clear consequence of Proposition 2, we find $|\mathcal{F}| \geq 2^{|\mathcal{X}|}$. As such, we will need to show $|\mathcal{X}| = \Omega\left(\frac{3^n}{n}\right)$. Note that since the number of balanced vectors on length $n \not\equiv 0 \pmod 3$ is at most 9 times the number of balanced vectors of size n' for the largest $n' < n$ with $n' \equiv 0 \pmod 3$, we may assume $n \equiv 0 \pmod 3$ for our asymptotic bound. By definition of \mathcal{X} , we can clearly see $|\mathcal{X}| = \binom{n}{n/3} \cdot \binom{2n/3}{n/3}$. Writing out the binomial coefficients gives us

$$\begin{aligned} |\mathcal{X}| &= \binom{n}{n/3} \cdot \binom{2n/3}{n/3} \\ &= \frac{n!}{(n/3)!(2n/3)!} \cdot \frac{(2n/3)!}{(n/3)!(n/3)!} \\ &= \frac{n!}{(n/3)!(n/3)!(n/3)!} \\ &= \frac{n!}{((n/3)!)^3} \end{aligned}$$

Using Stirling's approximation for the factorials and absorbing the constant factors in the big- Ω

notation, we find

$$\begin{aligned} |\mathcal{X}| &= \Omega \left(\frac{\sqrt{n} \cdot n^n \cdot e^{-n}}{\left(\sqrt{n/3} \cdot (n/3)^{(n/3)} \cdot e^{-n/3} \right)^3} \right) \\ &= \Omega \left(\frac{n^n}{(n/3)^n} \cdot \frac{\sqrt{n}}{\sqrt{n/3}^3} \right) \\ &= \Omega \left(\frac{3^n}{n} \right) \end{aligned}$$

□

Chapter 3

Lower Bound on algorithmic complexity

In this chapter we prove a runtime lower bound for solving TRIANGLE PARTITION on graphs of bounded cutwidth. We modify the lower bound for parameterization by pathwidth previously shown by Lokshtanov et al.[17] to be applicable to parameterization by cutwidth.

We first describe a reduction from CNF-SAT to TRIANGLE PARTITION by constructing a graph G^* parameterized by a formula ϕ , such that G^* can be partitioned into triangles if and only if ϕ is satisfiable. We follow the approach taken by Lokshtanov et al., whom first construct a graph G parameterized by a formula ϕ which has a triangle packing of size $mn(n+1) + m(n+1)$ if and only if ϕ is satisfiable. Next, they extend the graph G to a graph G' with the claimed property that G' can be partitioned into triangles if and only if ϕ is satisfiable. Unfortunately, G' cannot be partitioned into triangles regardless of ϕ , since some vertices are never 'cleaned up' as we will show in further detail after describing the construction. After that, we will be further extending the graph G' into a graph G^* in order to resolve this issue and we prove that G^* does satisfy the required property. Finally, we will bound the cutwidth of G^* in order to derive the desired lower bound.

We first describe the graph G as constructed by Lokshtanov et al. and summarize their proof of the properties of G . A drawing of G is shown in figure 3.1.

Lemma 3.1. *There is a polynomial-time algorithm that, given a CNF formula ϕ on n variables and m clauses, outputs a graph G such that G has a packing of $mn(n+1) + m(n+1)$ vertex-disjoint triangles if and only if ϕ is satisfiable.*

Proof. Let v_1, \dots, v_n be the variables of ϕ and C_1, \dots, C_m the clauses of ϕ . For each $i \in [n]$ we add a path P_i containing $2m(n+1) + 1$ vertices and a vertex set T_i of size $2m(n+1)$. We denote the l -th vertex of P_i and T_i by p_i^l and t_i^l respectively. For each $i \in [n]$, $l \in [2m(n+1)]$ we add the edges $\{t_i^l, p_i^l\}$ and $\{t_i^l, p_i^{l+1}\}$. For each $j \in [m]$ we add $n+1$ gadgets. In particular, for all $r \in [0..n]$ we add the vertices $\widehat{c}_j^r, \widehat{d}_j^r$ and the edge $\{\widehat{c}_j^r, \widehat{d}_j^r\}$. We will refer to this pair as \widehat{C}_j^r . Furthermore, whenever a variable v_i occurs positively in C_j we add the edges $\{\widehat{c}_j^r, t_i^{2(mr+j)}\}$ and $\{\widehat{d}_j^r, t_i^{2(mr+j)}\}$ for each $r \in [0..n]$. When v_i occurs negatively in C_j we add the edges $\{\widehat{c}_j^r, t_i^{2(mr+j)-1}\}$ and $\{\widehat{d}_j^r, t_i^{2(mr+j)-1}\}$ for each $r \in [0..n]$.

Next we formulate the properties of G as separate claims. Note that these claims correspond to [17, Lemmata 18, 19] respectively.

Claim 1. ϕ is satisfiable $\Rightarrow G$ has a packing of $mn(n+1) + m(n+1)$ vertex-disjoint triangles.

Proof. Suppose ϕ is satisfiable and consider a satisfying assignment of ϕ . For every variable v_i that is set true in the assignment we pack the triangles $\{t_i^{2l-1}, p_i^{2l-1}, p_i^{2l}\}$ for $l \in [m(n+1)]$. For every variable v_i set to false in the assignment we pack the triangles $\{t_i^{2l}, p_i^{2l}, p_i^{2l+1}\}$ for $l \in [m(n+1)]$.

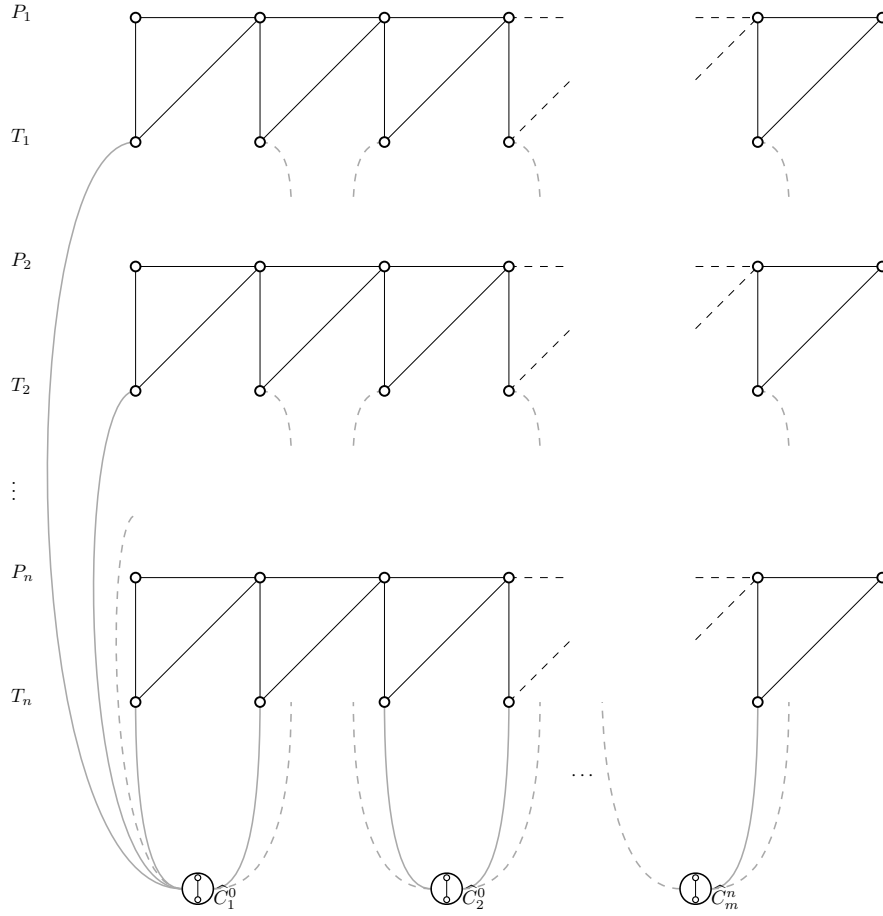


Figure 3.1: The graph G , constructed as a reduction from SAT to TRIANGLE PACKING. The grey edges represent edges that may or may not be present depending on the formula ϕ . The super vertices represent cliques with identical neighborhood.

Doing this for each variable gives us $mn(n+1)$ packed triangles. Since the assignment is satisfying, every clause has a literal which evaluates to true. That is, either a negative literal $\neg v_i$ with the v_i set to false, or a positive literal v_i with v_i set to true. As such, for all $0 \leq r \leq n$, if v_i occurs positively in C_j , then $t_i^{2(mr+j)}$ remains to be packed and we can add $\{\widehat{c}_j^r, \widehat{d}_j^r, t_i^{2(mr+j)}\}$. Conversely, if v_i occurs negatively in C_j , we know $t_i^{2(mr+j)-1}$ remains to be packed and we add $\{\widehat{c}_j^r, \widehat{d}_j^r, t_i^{2(mr+j)-1}\}$. Doing so for all $0 \leq r \leq n$ and $j \in [m]$ packs an additional $m(n+1)$ triangles, for a total of $mn(n+1) + m(n+1)$. ■

Claim 2. G has a packing of $mn(n+1) + m(n+1)$ vertex-disjoint triangles $\Rightarrow \phi$ is satisfiable.

Proof. We first observe that for any $i \in [n]$, any packing of $G[P_i \cup T_i]$ contains at most $m(n+1)$ triangles, since the only triangles in $G[P_i \cup T_i]$ are $\{t_i^\ell, p_i^\ell, p_i^{\ell+1}\}$ for $\ell \in [2m(n+1)]$, and no two subsequent triangles can be packed since they share a vertex. Secondly, we observe that, for any j and r , all triangles containing either \widehat{c}_j^r or \widehat{d}_j^r contain both these vertices.

Now suppose P is a packing of G containing $mn(n+1) + m(n+1)$ vertex-disjoint triangles. Let $P' \subset P$ be the packing containing those triangles of P which do not contain any vertex of the form \widehat{c}_j^r or \widehat{d}_j^r . Note that P' is a triangle packing of the graph $G[\bigcup_{i \in [n]} P_i \cup T_i]$. Furthermore, by our second observation, there can be no more than $n(m+1)$ triangles in P , which aren't contained in P' . Therefore, P' must contain at least $mn(n+1)$ triangles. Since $G[\bigcup_{i \in [n]} P_i \cup T_i]$ contains

n connected components, namely $G[P_i \cup T_i]$ for $i \in [n]$, which each contain at most $m(n+1)$ triangles, we also that find P' must contain at most $mn(n+1)$. Therefore, P' contains precisely $mn(n+1)$ triangles, by packing $m(n+1)$ triangles in each $G[P_i \cup T_i]$. Thus P must also pack $m(n+1)$ triangles in each $G[P_i \cup T_i]$. Note that since P contains an additional $m(n+1)$ triangles, it must also pack each pair $\{\widehat{c}_j, \widehat{d}_j\}$ in some triangle.

The only triangles present in $G[P_i \cup T_i]$ are of the form $\{t_i^l, p_i^l, p_i^{l+1}\}$, which we will refer to as Δ_i^l . Note that we cannot pack two subsequent triangles $\Delta_i^l, \Delta_i^{l+1}$ since they share the vertex p_i^{l+1} . The only way to pack $m(n+1)$ triangle in $G[P_i \cup T_i]$ is to pack the triangles $\Delta_i^1, \Delta_i^3, \dots, \Delta_i^{2k-1}, \Delta_i^{2k+2}, \Delta_i^{2k+4}, \dots, \Delta_i^{2m(n+1)}$ for some $k \in [0..m(n+1)]$. Note that for unless $k=0$ or $k=m(n+1)$, we skip one triangle when going from Δ_i^{2k-1} to Δ_i^{2k+2} . By the pigeon hole principle, there must be some $r \in [0..n]$ such that for no $i \in [n]$ the skip takes place between Δ_i^{2mr+1} and Δ_i^{2mr+2m} . For this fixed r , we know for each $i \in [n]$ either Δ_i^{2mr+1} or Δ_i^{2mr+2} must be used in the packing. For all variables v_i we set them to true if Δ_i^{2mr+1} is packed and false if Δ_i^{2mr+2} is packed.

For every j , the pair $\{\widehat{c}_j, \widehat{d}_j\}$ is packed in some triangle. The third vertex of this triangle must be either $t_i^{2(mr+j)}$ or $t_i^{2(mr+j)-1}$ for some $i \in [n]$. If it contains $t_i^{2(mr+j)}$, then C_j must have a positive literal of v_i . Since $t_i^{2(mr+j)}$ was not packed in $G[T_i \cup P_i]$, and the skip was not between Δ_i^{2mr+1} and Δ_i^{2mr+2m} , we know Δ_i^{2mr+1} must have been packed and therefore v_i is set to positive, which satisfies C_j . The argument to show that C_j contains a negative literal and v_i is set to false when $t_i^{2(mr+j)-1}$ is contained in the triangle is analogous. ■

Since G can be constructed in polynomial time, we combine the previous claims to complete the proof. □

Next, Lokshtanov et al. modify the construction to work for TRIANGLE PARTITION. They construct G' by adding a clique Q_i^l on four vertices for $i \in [n]$ and $l \in [m(n+1)]$ to G . All vertices from Q_i^l are connected to t_i^{2l} and t_i^{2l+1} . Furthermore, for all $i \in [n-1]$ and $l \in [m(n+1)]$ the vertices from Q_i^l are connected to Q_{i+1}^l . Let $p \in \{0, 1, 2\}$ such that $2n+2 = p \pmod 3$. We remove p vertices from Q_n^l for $l \in [m(n+1)]$. At this point, Lokshtanov et al. claim G' can be partitioned into triangles if and only if ϕ is satisfiable [17, Lemma 20]. However, for any $i \in [n]$, the set P_i contains an odd number of vertices, and any triangle containing a vertex from P_i contains exactly two vertices from P_i . Therefore, any triangle packing will leave at least one vertex from each P_i unpacked and thus G' cannot be partitioned into triangles, regardless of ϕ .

In order to modify the construction to work for TRIANGLE PARTITION, we will further adapt G' to a graph G^* , as shown in figure 3.2.

Lemma 3.2. *There is a polynomial-time algorithm that, given a CNF formula ϕ on n variables and m clauses with each clause containing at most q literals, outputs a graph G^* with linear layout π^* such that G^* can be partitioned into vertex-disjoint triangles if and only if ϕ is satisfiable and $\text{ctw}(\pi^*) \leq 2n + 2q + 56$.*

Proof. Let G be as constructed in Lemma 3.1. As Lokshtanov et al. did, we add a clique Q_i^l on four vertices for $i \in [n]$ and $l \in [m(n+1)]$ to G . We connect the vertices in Q_i^l to t_i^{2l} and t_i^{2l+1} , for $i \leq n-1$ we connect them to the vertices from Q_i^l to Q_{i+1}^l . We also remove p vertices from Q_n^l for $l \in [m(n+1)]$ for $p \in \{0, 1, 2\}$ such that $2n+2 = p \pmod 3$. Finally, we add cliques Q_i of size four for $i \in [2n]$. Similar to before, we connect the vertices in Q_i with the vertices in Q_{i+1} for all $i \in [2n-1]$. For $i \in [n]$ we connect the vertices in Q_i with p_i^1 . For $i \in [n+1..2n]$ we connect the vertices in Q_i with p_{2n+1-i}^1 . We formulate the properties of G^* as separate claims. The proof of the first two claims mostly follows the steps taken by Lokshtanov et al. but also takes into account the leftover vertices in the P_i .

Claim 3. ϕ is satisfiable $\Rightarrow G^*$ can be partitioned into triangles.

Proof. Suppose ϕ is satisfiable. Then G has a triangle packing of the form described above, by Claim 1. Since G is a subgraph of G^* , we can embed this packing in G^* . We will extend this

any leftover vertices are packed into a triangle with some number of vertices from Q_{i+1} . Since the total number of vertices packed this way are $2 \cdot 4n + n = 9n$, this can be partitioned into triangles exactly. Since we have packed all vertices of G^* into triangles, G^* can be partitioned into triangles. \blacksquare

Claim 4. G^* can be partitioned into triangles $\Rightarrow \phi$ is satisfiable.

Proof. Suppose G^* can be partitioned into triangles. We show ϕ is satisfiable by first showing G has a triangle packing of size $mn(n+1) + m(n+1)$. Let us consider which triangles from the partition are contained entirely in G . First we consider the vertices in the paths P_i for $i \in [n]$. We observe that any triangle containing a vertex from P_i contains either only the first vertex, only the final vertex, or exactly two subsequent vertices from P_i . Since every vertex must be packed in some triangle and the total number of vertices in P_i is $2m(n+1) + 1$, we can conclude the packing must contain one triangle containing either only the first or only final vertex from P_i , and $m(n+1)$ triangles containing two subsequent vertices from P_i . Since the third vertex of the $m(n+1)$ triangles is always a vertex from T_i , we can conclude that each $P_i \cup T_i$ contains at least $m(n+1)$ triangles from the packing. Since the graph G contains $P_i \cup T_i$ for all $i \in [n]$, G must contain all $mn(n+1)$ triangles.

Next, we consider the triangles containing the clause vertices \widehat{c}_j^r and \widehat{d}_j^r for $j \in [m]$, $r \in [0..n]$. First note that any triangle containing either of these vertices must also contain the other. Since the only vertices adjacent to these clause vertices are contained in G , we can safely assume the triangle containing these vertices is entirely contained in G .

Therefore, the triangle packing of G containing all triangles in the triangle partition of G^* which are entirely contained in G contains at least $mn(n+1) + m(n+1)$ triangles. Therefore we can conclude ϕ is satisfiable by Claim 2. \blacksquare

We now proceed to bound the cutwidth of G^* . In order to do so, we partition the graph G^* into $m(n+1)$ columns. For $k \in [m(n+1)]$, the k -th column contains the following vertices $p_i^{2^k}$, $p_i^{2^k-1}$, $t_i^{2^k}$, $t_i^{2^k-1}$ for $i \in [n]$ together with the unique pair of clause vertices adjacent to these vertices and the cliques Q_i^k for $i \in [n]$. Additionally, the first column also contains the cliques Q_i for $i \in [n]$ and the final column also contains the vertices $p_i^{2^{m(n+1)+1}}$ and the cliques Q_{n+i} for $i \in [n]$.

Let π^* be the linear layout of G' which contains the columns in order, where each column is ordered as follows. Column k first contains the two clause vertices contained in that column. Next, it contains the vertices $p_i^{2^k-1}$, $t_i^{2^k-1}$, $p_i^{2^k}$, $t_i^{2^k}$ and the clique Q_i^k in that order, for fixed $i \in [n]$ in ascending order. In the first column the clique Q_i is ordered directly before the vertex p_i^1 . In the final column the vertex $p_i^{2^{m(n+1)+1}}$ and clique Q_{2n-i} are ordered directly before $Q_i^{m(n+1)}$.

Claim 5. $\text{ctw}(\pi^*) \leq 2n + 2q + 56$.

Proof. Consider an arbitrary vertex $v \in V(G')$ and the cut consisting of all edges crossing the gap after v in π^* . Suppose v lies in column k .

If v is either of the clause vertices, the cut will contain the $2n$ edges from the variable vertices of the previous column to the variable vertices of this column and the (at most) $2q$ edges from the clause vertices to the variable vertices of this column or the edge between the clause vertices and at most q edges from the clause vertices to variable vertices. Since $q \geq 1$, the width of such a cut is at most $2q + 2n$.

If v is any of the variable vertices for the i -th variable, the cut contains $2n$ edges connecting the variable vertices of this column with the variable vertices of the next column (for the first i variables) and previous column (for the other variables), up to eight edges from the variable vertices for the i -th variable to the clique Q_i^k , the 16 edges from the clique Q_{i-1}^k to the clique Q_i^k , and the (at most) $2q$ edges from the clause vertices to the variable vertices (that come after i). Unless $k = 1$, there are an additional 16 edges connecting Q_n and Q_{n+1} . If $k = 1$ or $k = m(n+1)$ an additional 16 edges from Q_i to Q_{i+1} or from Q_{2n-i} to Q_{2n-i-1} are also contained. This means that in total the width of such a cut is at most $2n + 2q + 56$.

If v is vertex from the clique Q_i for some $i \in [n]$, the cut contains up to four edges connecting the clique to p_i^1 , up to four edges connecting the vertices of the clique with each other and up to 16 edges Q_i to Q_{i+1} . It also contains $2(i-1)$ edges connecting variable vertices for the first $i-1$ variables to the variable vertices of the next column. Additionally, up to $2q$ edges connecting the clause vertices to the variable vertices may be contained in the cut. Furthermore, 16 edges connecting the clique Q_{i-1}^1 with Q_i^1 and 16 edges connecting Q_n with Q_{n+1} are contained in the cut. As such, the width of such a cut is at most $2q + 2n + 38$.

Similarly, if v is either $p_i^{2^{m(n+1)+1}}$ or a vertex from Q_{2n-i} , the cut contains up to four edges connecting $p_i^{2^{m(n+1)+1}}$ to Q_{2n-i} , up to four edges connecting vertices within Q_{2n-i} with each other, $2(n-i)$ edges connecting the variable vertices of this column with the variable vertices of the previous column, up to $2q$ edges connecting variable vertices to the clause vertices, 16 edges connecting Q_{2n-i} to Q_{2n-i-1} and Q_{2n-i+1} , and another 16 edges connecting Q_n with Q_{n+1} . Therefore, the width of such a cut is at most $2q + 2n + 40$.

If v is a vertex from the clique Q_i^k , the cut contains $2n$ edges connecting the variable vertices of this column with the variable vertices of the next column (for the first i variables) and previous column (for the other variables), at most $2q$ edges connecting the variable vertices with the clause vertices, up to six edges from the variable vertices for the i -th variable to the clique Q_i^k , the 16 edges connecting this clique to the next and previous cliques, another 16 edges connecting Q_n with Q_{n+1} and up to four edges between the vertices of Q_i^k . Thus the width of such a cut is at most $2n + 2q + 42$.

Thus regardless of the cut considered, its size is always bounded by $2n + 2q + 56$, which by definition implies $\text{ctw}(\pi^*) \leq 2n + 2q + 56$. \blacksquare

Since both G^* and π^* can be constructed in polynomial time, we combine the previous claims to complete the proof. \square

Theorem 3.3. *Assuming SETH, there is no $\varepsilon > 0$ such that TRIANGLE PARTITION on a graph G given along with a linear layout π^* can be solved in time $\mathcal{O}^*((2-\varepsilon)^{\text{ctw}(\pi^*)/2})$.*

Proof. Suppose TRIANGLE PARTITION on a graph G with linear layout π can be solved in $\mathcal{O}^*((2-\varepsilon)^{\text{ctw}(\pi)/2})$. We show this contradicts SETH, by showing that this implies the existence of a $\delta > 0$ such that n -variable q -SAT can be solved in $\mathcal{O}^*((2-\delta)^n)$ for any fixed q .

Let ϕ be a n -variable q -SAT formula for any fixed q . We solve the satisfiability problem on ϕ by constructing G^* and π^* by Lemma 3.2 and solve TRIANGLE PARTITION on it with the assumed algorithm. This procedure would take $\mathcal{O}^*((2-\varepsilon)^{\text{ctw}(\pi^*)/2}) = \mathcal{O}^*((2-\varepsilon)^{n+q+28}) = \mathcal{O}^*((2-\varepsilon)^n)$. Note that we can drop the contribution of q in the \mathcal{O}^* -notation since we assumed q to be a fixed constant. Since this holds for any q , it directly contradicts SETH. \square

Chapter 4

Upper Bound on algorithmic complexity

In this chapter we prove an upper bound on the complexity of TRIANGLE PARTITION on graphs of bounded cutwidth. We do so by describing an algorithm which reduces a graph G to a graph G' , such that G allows a triangle partition if and only if G' allows a triangle partition, but additionally satisfies the property $\mathbf{pw}(G') \leq \frac{3}{4}\mathbf{ctw}(G)$. This allows us to use the 'naive' path decomposition algorithm on G' and obtain an upper bound of $\mathcal{O}^*(2^{\frac{3}{4}\mathbf{ctw}(G)})$.

The modification algorithm is rather straightforward. For every edge in G , we simply check if it's part of at least one triangle. If it isn't, we simply remove it from the graph, since it clearly cannot be used in a triangle partition. Let G' be the graph obtained by doing so, then clearly $\mathbf{ctw}(G') \leq \mathbf{ctw}(G)$ since G' is entirely contained in G . Note that this reduction algorithm has polynomial complexity.

Lemma 4.1. *There is a polynomial-time algorithm that, given a graph G where each edge is contained in a triangle with linear layout π , outputs a path decomposition P satisfying $\mathbf{pw}(P) \leq \frac{3}{4}\mathbf{ctw}(\pi) + 2$.*

Proof. We will denote the i -th vertex in π by v_i . We will define a path decomposition of G via mixed search games. Our search strategy S consists of $|V(G)|$ rounds. The i -th round corresponds to the cut after v_i . Whenever we refer to the degree of a vertex, we mean the degree of a vertex in the graph induced on the edges over the cut. At the start of round i , we add a cleaner to v_i unless it already contains a cleaner. We then remove all cleaners from vertices with degree 0. Finally, any cleaner on a vertex with degree 1 left of the cut we move along the edge unless there is a cleaner on the other endpoint, in which case we remove the cleaner on the left endpoint instead.

In order to prove the lemma, we need to prove two properties of S , which formulate as two separate claims.

Claim 6. *The mixed search strategy S cleans all edges of G without allowing recontamination.*

Proof. We prove this claim by induction. Our induction hypothesis states at the end of round i all edges between vertices left of the cut after v_i are clean, and no recontamination occurs. Clearly, during the first round this is true, since there aren't any edges entirely left of the cut.

Suppose at the end of round $i - 1$ all edges left of the cut are clean. During round i , we first add a cleaner to the vertex v_i . As such, any edge between a vertex left of the cut with a cleaner on it and v_i is cleaned. We now argue that at this point all edges left of the cut are clean. The edges one might suspect are still contaminated are edges connecting a vertex w left of the cut which did not have a cleaner on it with v_i . Clearly, we didn't remove the cleaner from w at any point, because every round since we put a cleaner on w had the edge $\{w, v_i\}$ still in the cut. Thus, if there is no cleaner on vertex w , it must have moved along an edge. Since this only happens when w has only a single incident edge over the cut, it must have moved along $\{w, v_i\}$. Therefore, this

edge was cleaned in some earlier round. By our induction hypothesis, no recontamination occurs thus it must still be clean during this round. Therefore we can conclude that at this point all edges entirely left of the cut are clean.

However, we still have to argue that during the rest of this round we don't recontaminate any edges. We remove all cleaners from vertices with degree 0. This can only happen to vertices left of the cut. This means the only path from a contaminated edge to this one must use an edge over the cut. Since every edge over the cut has a cleaner on one of its endpoints, no recontamination can occur when we remove this cleaner. Finally, we may move some cleaners along edges. The only way this could cause recontamination is if there is some path to a contaminated edge which does not use the edge along which the cleaner moves. Since we only move the cleaner if there is only a single incident edge over the cut, this cannot happen, thus at the end of this round all edges left of the cut are still clean.

Thus, by induction we can conclude that after the final round all edges entirely left of the cut are cleaned and no recontamination occurred. Since all edges are entirely left of the cut, this is sufficient to prove our claim. ■

Claim 7. *The mixed search strategy S can be executed using $\frac{3}{4}\text{ctw}(\pi) + 1$ cleaners.*

Proof. Let $L_i, R_i \subset V(G)$ be the vertices containing a cleaner at the end of round i left and right of the cut respectively. To prove the claim, it suffices to show $|L_i \cup R_i| \leq \frac{3}{4}\text{ctw}(G)$ for all $i \in [n]$, since at most one additional cleaner is placed between the end of two consecutive rounds.

We first observe any vertex in R_i must have degree at least 2. If a vertex right of the cut contains a cleaner, it must be because the cleaner was moved over an edge. This only happens if the other endpoint of that vertex was incident to only one edge over the cut. Since this edge is part of some triangle, there must be another vertex on the left side of the cut that also has an edge to the same vertex on the right, which means that vertex must be incident to at least two edges. As a consequence, for any i we find $\text{ctw}(\pi) \geq 2|R_i|$.

Next, we observe the same must be true for any vertex in L_i . After all, if a vertex left of the cut had either degree 0 or 1, the cleaner would have been removed or moved respectively.

Finally, we note that any vertex in R_i must have a neighbour left of the cut without a cleaner on it, since the cleaner must have moved from somewhere. This allows us to derive a second inequality on $\text{ctw}(\pi)$, by counting the number of edges over a cut in terms of $|R_i|$ and $|L_i|$:

$$\begin{aligned} \text{ctw}(\pi) &\geq \sum_{v \in L_i} d(v) + \sum_{u \in R_i} d(u) - |E(G) \cap (L_i \times R_i)| \\ &\geq 2|L_i| + \sum_{u \in R_i} d(u) - \sum_{u \in R_i} d(u) - 1 \\ &\geq 2|L_i| + |R_i| \end{aligned}$$

Together with the first inequality this gives us $\text{ctw}(\pi) \geq \max\{2|R_i|, 2|L_i| + |R_i|\}$. If $|L_i| = 0$, we immediately find $|L_i| + |R_i| = |R_i| \leq \frac{\text{ctw}(\pi)}{2}$. Otherwise, there is an α such that $|R_i| = \alpha \cdot |L_i|$. This reduces our inequality to $\text{ctw}(\pi) \geq \max\{2\alpha|L_i|, (2 + \alpha)|L_i|\}$. We now have to show $|L_i| + |R_i| = (1 + \alpha)|L_i| \leq \frac{3}{4}\text{ctw}(\pi)$. We will do so by a case distinction on the value of α .

Suppose $\alpha > 2$, then our inequality reduces to $\text{ctw}(\pi) \geq 2\alpha|L_i|$. This allows us to derive

$$\begin{aligned} (1 + \alpha)|L_i| &\leq \left(\frac{1}{2}\alpha + \alpha\right)|L_i| \\ &\leq \frac{3}{2}\alpha|L_i| \\ &\leq \frac{3}{2} \cdot \frac{\text{ctw}(\pi)}{2} \\ &= \frac{3}{4}\text{ctw}(\pi) \end{aligned}$$

Now suppose $\alpha \leq 2$, then our inequality reduces to $\mathbf{ctw}(\pi) \geq (2 + \alpha)|L_i|$. This allows us to derive

$$(1 + \alpha)|L_i| \leq (1 + \alpha) \frac{\mathbf{ctw}(\pi)}{2 + \alpha}$$

Since the right hand side is an increasing function in α , and we know $0 \leq \alpha \leq 2$, we know it attains its maximum at $\alpha = 2$, where it attains the value $\frac{3}{4}\mathbf{ctw}(\pi)$.

Thus, at the end of any round there are at most $\frac{3}{4}\mathbf{ctw}(\pi)$ cleaners in the graph. Since during each round we introduce (at most) a single cleaner, this implies S can be executed with at most $\frac{3}{4}\mathbf{ctw}(\pi) + 1$ cleaners. ■

Thus, by the equivalence of mixed search games and pathwidth as discussed in chapter 2, the path decomposition P implicitly defined by S satisfies $\mathbf{pw}(P) \leq \frac{3}{4}\mathbf{ctw}(\pi) + 2$. □

This lemma combined with the reduction algorithm described earlier proves the main result of this chapter.

Theorem 4.2. *Given a graph G and a linear layout π , there exists an algorithm which solves TRIANGLE PARTITION on G in $\mathcal{O}^*(2^{\frac{3}{4}\mathbf{ctw}(\pi)})$.*

Proof. We first remove all edges from G not contained in any triangle to obtain the graph G' . Clearly, G' allows a triangle partition if and only if G allows a triangle partition, so we may solve triangle partition on G' instead of G . Since G and G' share their vertex set, π is linear layout of G' as well with a width no more than in G . Lemma 4.1 can be used to obtain a path decomposition P of G' from π , which satisfies $\mathbf{pw}(P) \leq \frac{3}{4}\mathbf{ctw}(\pi)$. Using the straightforward dynamic programming approach on P gives us a complexity of $\mathcal{O}^*(2^{\frac{3}{4}\mathbf{ctw}(\pi)})$. □

Chapter 5

Lower Bound on communication complexity

In this chapter we will derive a lower bound on the complexity of TRIANGLE PARTITION COMMUNICATION. Recall that in this variant of TRIANGLE PARTITION, the graph G is divided among two players, Alice and Bob. Together they have to decide whether G can be partitioned using minimal communication. Roughly speaking, we derive the lower bound by constructing a family of graphs \mathcal{G} each representing a possible part of G that Alice may receive. \mathcal{G} is constructed such that Alice must send a unique message for each graph in the family. This allows to argue that any protocol which uses a large number of unique messages must also use large messages.

We construct G^f parameterized by a monotone ternary boolean function $f : \{0, 1, 2\}^n \rightarrow \{0, 1\}$. We define $\mathcal{G} = \{G^f \mid f \in \mathcal{F}\}$ where \mathcal{F} is the family of all monotone n -dimensional ternary boolean functions. Let $f : \{0, 1, 2\}^n \rightarrow \{0, 1\}$ be a monotone function. By Proposition 1 there exist coefficients $c_{i,j} \in \{0, 1, 2\}$ such that $f(x) = \bigwedge_{j=1}^m C_j$ where $C_j = \bigvee_{i=1}^n (x_i > c_{i,j})$. We use these coefficients to construct a graph G^f , which is visualized in figure 5.1.

For each $i \in [2n]$, we add a path P_i containing $2m + 1$ vertices, with the l -th vertex denoted by p_i^l . We also add a vertex set T_i of size $2m$, denoting the l -th vertex by t_i^l . We connect these by adding the edges $\{t_i^l, p_i^l\}$ and $\{t_i^l, p_i^{l+1}\}$ for $l \in [2m]$. We then pair up these structures. Specifically, for all $i \in [n]$, $l \in [m]$ we add the edges $\{t_{2i-1}^{2l-1}, t_{2i}^{2l-1}\}$ and $\{t_{2i-1}^{2l}, t_{2i}^{2l}\}$. Finally, for each $i \in [n]$ we add the edge $\{p_{2i-1}^{2m+1}, p_{2i}^{2m+1}\}$.

For each $j \in [m]$ we add a vertex \hat{c}_j . For each $i \in [n]$ we add edges dependent on the coefficients $c_{i,j}$. If $c_{i,j} = 0$ we connect \hat{c}_j to t_{2i-1}^{2j-1} , t_{2i}^{2j-1} and t_{2i}^{2j} . If $c_{i,j} = 1$ we connect \hat{c}_j to t_{2i-1}^{2j-1} and t_{2i}^{2j-1} . If $c_{i,j} = 2$ we don't add any edges. These three cases are visualized in figures 5.2, 5.3, and 5.4 respectively.

Next we add a path of cliques for each $j \in [m]$. For each $j \in [m]$ and $i \in [2n]$ we add a clique Q_i^j of size four. We connect the clique vertices in Q_i^j with t_i^{2j-1} and t_i^{2j} and the clique Q_{i+1}^j (unless $i = 2n$). For each $j \in [m]$ we remove $k \in \{0, 1, 2\}$ vertices from Q_{2n}^j for k such that $4n + 4 - k \equiv 0 \pmod{3}$. We remove these vertices so that the vertices we wish to partition in triangles using these cliques is a multiple of 3, thus preventing any left-over vertices.

Next we add one final path of cliques. That is, for each $i \in [2n]$ we add a clique Q_i of size four. We connect each vertex of clique Q_i to each vertex in cliques Q_{i-1} and Q_{i+1} (when they exist) and the vertex p_i^1 . We do not need to remove any vertices from the final clique of this path, since the number of vertices we wish to partition with these cliques is a multiple of 3, regardless of n . This concludes the construction of G^f .

Next we describe a family of graphs $\mathcal{G}_B = \{G_c^x \mid x \in \{0, 1, 2\}^n, c \in \{0, 1, 2\}\}$ which represent different options Bob might get as his part of the graph. We use these graphs to argue why a unique message must be sent for each G^f . Let $x \in \{0, 1, 2\}^n$ and $c \in \{0, 1, 2\}$. G_c^x contains the gadgets B_i for $i \in [n]$. All B_i have the vertices b_i^1 and b_i^2 and the edge $\{b_i^1, b_i^2\}$ which will be part of the boundary. The rest of B_i depends on whether x_i is zero, one, or two. These cases are

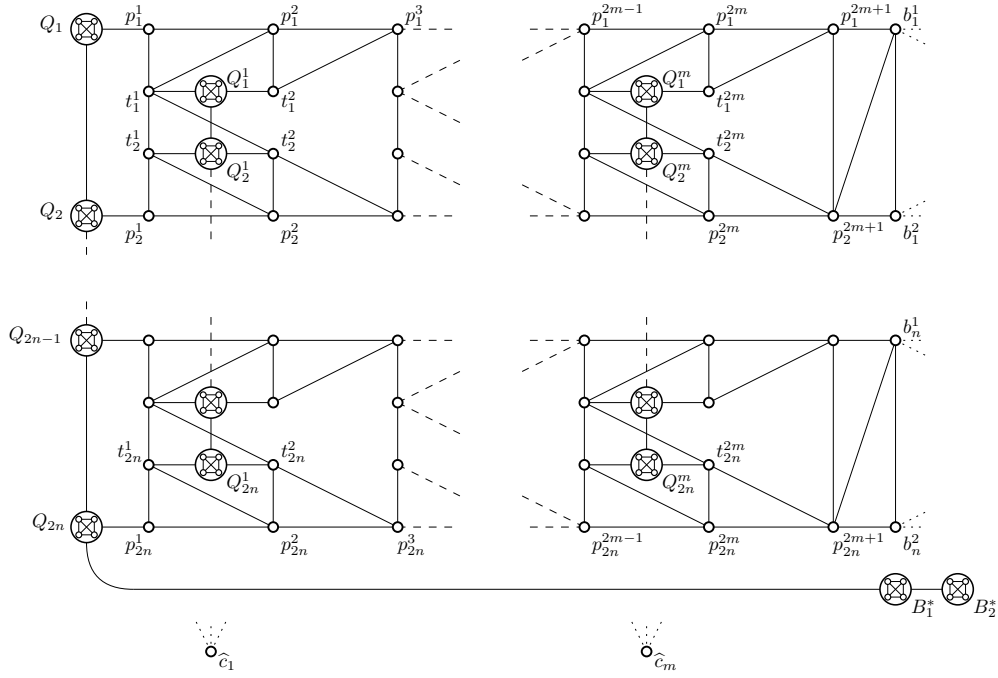


Figure 5.1: The graph G^f and the boundary Δ . Edges incident to clause vertices are not drawn fully. The super vertices represent cliques with identical neighborhood.

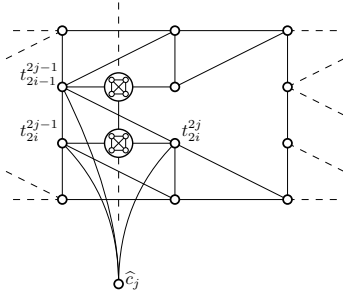


Figure 5.2: The edges connecting \hat{c}_j when $c_{i,j} = 0$.

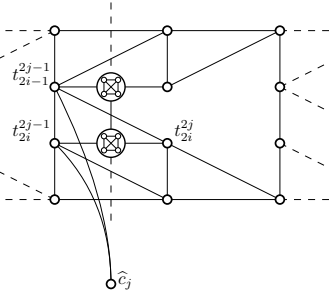


Figure 5.3: The edges connecting \hat{c}_j when $c_{i,j} = 1$.

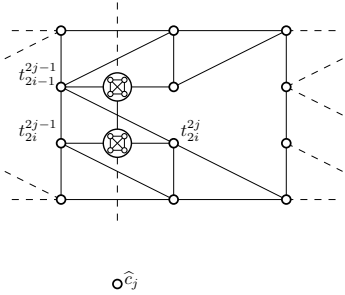


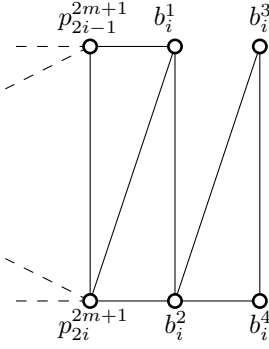
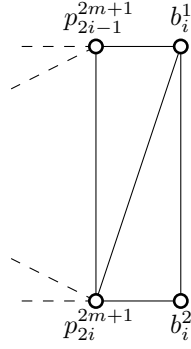
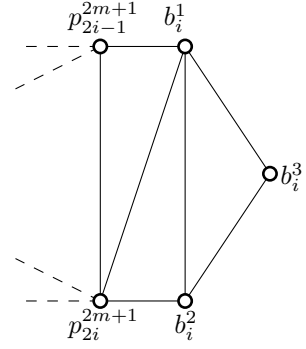
Figure 5.4: The edges connecting \hat{c}_j when $c_{i,j} = 2$.

visualized in figures 5.5, 5.6 and 5.7 respectively. If $x_i = 0$ we add the vertices b_i^3 and b_i^4 and the edges $\{b_i^2, b_i^3\}$, $\{b_i^2, b_i^4\}$, and $\{b_i^3, b_i^4\}$. If $x_i = 1$ nothing further is added to B_i . If $x_i = 2$ we add an additional vertex b_i^3 and the edges $\{b_i^1, b_i^3\}$ and $\{b_i^2, b_i^3\}$ to B_i . Additionally, G_B^x contains two additional gadgets B_1^* and B_2^* . The gadget B_1^* is a clique of size four, and is part of the boundary. The gadget B_2^* will be a clique of a size c and will be fully connected to B_1^* .

In order to construct the complete graph G from the parts G_A and G_B we connect them in the following manner. For each $i \in [n]$ we add the edges $\{p_{2i-1}^{2m+1}, b_i^1\}$, $\{p_{2i}^{2m+1}, b_i^1\}$, and $\{p_{2i}^{2m+1}, b_i^2\}$. Finally we completely connect the cliques Q_{2n} and B_1^* . We will denote the graph constructed in this manner as $G_A \oplus G_B$.

Next we will argue that for every pair of functions $f, f' \in \mathcal{F}$, the graphs associated with them behave differently with respect to our communication problem. This allows us to conclude Alice must send a different message for each graph in \mathcal{G} , which in turn implies the protocol must use messages of at least size $\Omega(\frac{3^n}{n})$.

Lemma 5.1. *For every pair $G^f, G^{f'}$ where $f \neq f'$ there exists a graph G_B such that exactly one of $G^f \oplus G_B, G^{f'} \oplus G_B$ can be partitioned into triangles.*


 Figure 5.5: The gadget B_i when $x_i = 0$.

 Figure 5.6: The gadget B_i when $x_i = 1$.

 Figure 5.7: The gadget B_i when $x_i = 2$.

Proof. Since $f \neq f'$ there must be some $x \in \{0, 1, 2\}^n$ such that $f(x) \neq f'(x)$. Let us assume without loss of generality that $f(x) = 1$ and $f'(x) = 0$ and let c be such that $|V(G^f)| + |V(G_c^x)| = 0 \pmod 3$. Then $G^f \oplus G_c^x$ can be partitioned into triangles and $G^{f'} \oplus G_c^x$ cannot.

Claim 8. $G^f \oplus G_c^x$ can be partitioned into triangles.

Proof. For each pair P_i, T_i for $i \in [2n]$ we pack triangles in either the even or the odd configuration. The even configuration contains the triangles $\{p_i^{2l}, p_i^{2l+1}, t_i^{2l}\}$ for $l \in [m]$ and the odd configuration contains the triangles $\{p_i^{2l-1}, p_i^{2l}, t_i^{2l-1}\}$ for $l \in [m]$. The vertices from $P_i \cup T_i$ not yet packed into triangles are either $\{t_i^{2l-1} \mid l \in [m]\} \cup \{p_i^1\}$ or $\{t_i^{2l} \mid l \in [m]\} \cup \{p_i^{2m+1}\}$ in case of even or odd configuration respectively.

The values of x_i determine which configuration we use for the pairs P_{2i-1}, T_{2i-1} and P_{2i}, T_{2i} . If $x_i = 0$, we use the odd configuration for both pairs. If $x_i = 1$, we pack P_{2i-1}, T_{2i-1} in the even configuration and P_{2i}, T_{2i} in the odd configuration. If $x_i = 2$, we use the even configuration for both pairs.

The gadget B_i is constructed precisely such that whichever of the vertices $p_{2i-1}^{2m+1}, p_{2i}^{2m+1}$ remain unpacked by the configuration can be partitioned into triangles with the vertices in B_i . If $x_i = 0$ both p_{2i-1}^{2m+1} and p_{2i}^{2m+1} are yet unpacked into a triangle. We add the triangles $\{p_{2i-1}^{2m+1}, p_{2i}^{2m+1}, b_i^1\}$ and $\{b_i^2, b_i^3, b_i^4\}$ to the packing. If $x_i = 1$, only p_{2i}^{2m+1} remains to be packed, which we do by adding the triangle $\{p_{2i}^{2m+1}, b_i^1, b_i^2\}$ to the packing. If $x_i = 2$, both vertices are already packed into triangles, however in this case we can pack B_i itself into the single triangle $\{b_i^1, b_i^2, b_i^3\}$.

Now let us consider how to pack the clause vertices \hat{c}_j for $j \in [m]$. Since $f(x) = 1$ by assumption, we know for each $j \in [m]$ there must be some $i \in [n]$ such that $x_i > c_{i,j}$. If $x_i = 1$ and $c_{i,j} = 0$, we add the triangle $\{t_{2i-1}^{2j-1}, t_{2i}^{2j}, \hat{c}_j\}$ to the packing. If $x_i = 2$ and $c_{i,j}$ is either zero or one, we add the triangle $\{t_{2i-1}^{2j-1}, t_{2i}^{2j-1}, \hat{c}_j\}$ to the packing. Now for each $j \in [m]$, there is exactly one $i \in [n]$ such that all four vertices $t_{2i-1}^{2j-1}, t_{2i-1}^{2j}, t_{2i}^{2j-1}, t_{2i}^{2j}$ are already packed into triangles, and for all other $i \in [n]$ exactly one of $t_{2i-1}^{2j-1}, t_{2i-1}^{2j}$ and of $t_{2i}^{2j-1}, t_{2i}^{2j}$ has been packed. These leftover vertices will be packed into triangles with the vertices from the cliques Q_i^j . For each $i \in [2n]$ and $j \in [m]$ we add the triangle containing the leftover vertex (whenever it exists) with two vertices from the adjacent clique.

For each $j \in [m]$, we have a path of cliques $Q_1^j, Q_2^j, \dots, Q_{2n}^j$ with some number of vertices of the cliques already packed. To be precise, from two of the cliques no vertices have been packed and from all other cliques exactly two vertices have been packed. This means that the total number of unpacked vertices is $2(2n - 2) + 8 - k = 4n + 4 - k$ for $k \in \{0, 1, 2\}$ such that $4n + 4 - k = 0 \pmod 3$. We pack these vertices in the following manner. We go through the cliques in ascending order of i , each time packing the vertices in a triangle possibly also containing vertices from the next clique. If the clique has zero unpacked vertices, we go to the next clique. If the clique has one unpacked vertex, we pack it with two vertices from the next clique. If the clique has two unpacked

vertices, we pack it with one vertex from the next clique. If the clique has three unpacked vertices we simply pack them into a single triangle. If the clique has four unpacked vertices we pack it into two triangles, one containing three vertices from this clique and the other containing one vertex from the current clique and two vertices from the next clique.

Note that in all these cases we use at most two vertices from the next clique. Since every clique except the last each have at least two unpacked vertices at the start of this procedure, it cannot fail before the packing of the penultimate clique. Note that at this point we may have packed some vertices from the penultimate clique into a triangle with vertices from the clique before it. At the point where we start packing the penultimate clique, we know all vertices from cliques before it must have been packed. Since we started with $4n + 4 - k$ unpacked vertices, which is a multiple of three, we know the number of unpacked vertices at this point is also a multiple of three. Furthermore, all unpacked vertices are in either the penultimate or final clique, which are two subsequent cliques. Therefore, all remaining vertices are connected and thus form a clique. Thus, they can clearly be partitioned into triangles.

The only vertices yet to be packed into triangles are the p_i^1 for P_i packed in the even configuration, the vertices from the path of cliques Q_i and the vertices from the cliques B_1^* and B_2^* . We first pack the vertices p_i^1 into a triangle with two vertices from the adjacent clique Q_i . Then the path of cliques contains some cliques with four unpacked vertices and some cliques with two unpacked vertices. We pack this path of cliques using the same procedure as we used to pack the previous paths of cliques. The final clique in the path may contain some number of unpacked vertices at the end of the procedure. If there is one unpacked vertex in the final clique, we pack it together with two vertices from B_1^* . If there are two unpacked vertices in the final clique we pack them together with one vertex from B_1^* . Afterwards all unpacked vertices form a clique, and since by choice of c the total number of vertices is divisible by three, we are able to partition the remaining vertices in triangles. Thus we have partitioned the entire graph into triangles. ■

Claim 9. $G^{f'} \oplus G_c^x$ cannot be partitioned into triangles.

Proof. We will prove this by contradiction. Suppose there exists a packing which partitions $G^{f'} \oplus G_c^x$ into triangles. We start by considering the paths P_i for $i \in [2n]$. Other than the first and final vertices of each path, these vertices can only be packed into triangles containing two subsequent vertices from P_i and the unique vertex from T_i adjacent to both of them. This property leads to the fact that any triangle partition must put the paths into either the odd or even configuration as described before.

Suppose that the final vertex p_i^{2m+1} is packed into the triangle $\{p_i^{2m}, p_i^{2m+1}, t_i^{2m}\}$, then the previous vertex in the path p_i^{2m-1} can only be packed into the triangle $\{p_i^{2m-2}, p_i^{2m-1}, t_i^{2m-2}\}$. This in turn implies there is only one triangle in which p_i^{2m-3} can be packed, and this property propagates backward, until the only vertex of the path P_i that is unpacked is the first. This describes precisely the even configuration which we defined previously. Conversely, suppose that the final vertex p_i^{2m+1} is not packed into the triangle $\{p_i^{2m}, p_i^{2m+1}, t_i^{2m}\}$. Then $\{p_i^{2m-1}, p_i^{2m}, t_i^{2m-1}\}$ must be in the packing since it's the only triangle containing p_i^{2m} but not p_i^{2m+1} . Similar to before, this implies in which triangle p_i^{2m-2} must be packed. Since this property again propagates through the path, we find that this describes the odd configuration. Thus we can conclude that the path P_i must be packed into the even configuration when p_i^{2m+1} is used in the triangle $\{p_i^{2m}, p_i^{2m+1}, t_i^{2m}\}$ and in the even configuration otherwise.

Next, let us consider the gadgets B_i for $i \in [n]$. We will consider the three forms B_i can have depending upon x_i separately. If $x_i = 0$, clearly the triangle $\{b_i^2, b_i^3, b_i^4\}$ must be in the packing. Then there is only one triangle which can be in the packing containing b_i^1 , thus $\{p_{2i-1}^{2m+1}, p_{2i}^{2m+1}, b_i^1\}$ must also be in the packing. Note that this also implies that both P_{2i-1} and P_{2i} must be put into the odd configuration. If $x_i = 1$, there is only a single triangle containing b_i^2 , thus the triangle $\{b_i^1, b_i^2, p_{2i}^{2m+1}\}$ must be in the triangle. This means the only triangle in which p_{2i-1}^{2m+1} can be packed is $\{p_{2i-1}^{2m}, p_{2i-1}^{2m+1}, t_{2i-1}^{2m}\}$. Thus in this case we can also derive that P_{2i} is put into the odd configuration and P_{2i-1} in the even configuration. Finally, if $x_i = 2$, the triangle $\{b_i^1, b_i^2, b_i^3\}$ must clearly be contained in the packing. This implies p_{2i-1}^{2m+1} must be packed in $\{p_{2i-1}^{2m}, p_{2i-1}^{2m+1}, t_{2i-1}^{2m}\}$

and p_{2i}^{2m+1} in $\{p_{2i}^{2m}, p_{2i}^{2m+1}, t_{2i}^{2m}\}$, thus implying that both paths are in even configuration. As such, we can derive the configurations of P_{2i-1} and P_{2i} from the value of x_i .

To derive our contradiction, we will now investigate the relation between the configuration of the paths and the packing of clause vertices. We first observe that the only way to pack a clause vertex \hat{c}_j is to find an $i \in [n]$ such that $c_{i,j} = 0$ and P_{2i-1} is in the even configuration or $c_{i,j} = 1$ and P_{2i-1} and P_{2i} are both in the even configuration (note that these $c_{i,j}$ are the coefficients for the function f' , not those of f). By the direct relation between the value of x_i and the configuration of P_{2i-1} and P_{2i} , this is equivalent to finding an $i \in [n]$ such that $c_{i,j} = 0$ and $x_i \geq 1$ or $c_{i,j} = 1$ and $x_i = 2$. In either of these cases we find $x_i > c_{i,j}$. Since such i must exist for all j , this implies $f'(x) = 1$, which contradicts our assumption that $f'(x) = 0$. ■

Combining these two claims completes the proof of the lemma. □

Theorem 5.2. *Any one-way communication protocol which solves TRIANGLE PARTITION COMMUNICATION has communication complexity $\Omega(\frac{3^{k/3}}{k})$, where the parameter k is the number of edges connecting the two parts.*

Proof. Since for every $f \neq f'$ there exists a G_B such that exactly one of $G^f \oplus G_B, G^{f'} \oplus G_B$ can be partitioned into triangles by Lemma 5.1, any communication protocol must send a different message for G^f and $G^{f'}$. Since these were arbitrary members of \mathcal{G} , the number of unique messages used in the protocol must be at least $|\mathcal{G}|$. Since the messages are binary strings, the size of largest message in the protocol must be at least the logarithm of the number of unique messages. Therefore, the size of this message is bounded from below by $\log_2 |\mathcal{G}|$. By construction of \mathcal{G} , we clearly find $|\mathcal{G}| = |\mathcal{F}|$, which combined with Corollary 2.1 proves a lower bound on the size of the largest message of $\Omega(\frac{3^n}{n})$. Since the number of edges between the parts of the graph is $3n + 16$, this gives the required bound. □

Chapter 6

Upper Bound on communication complexity

In order to derive an upper bound on the communication complexity, we will describe a communication protocol which solves the TRIANGLE PARTITION COMMUNICATION. Recall that for communication problems, we measure the efficiency of a protocol in terms of the amount of communication, not runtime complexity. This means that any local computation by Alice and Bob is essentially ‘free’, even if they require exponential time. Roughly, the protocol works as follows. First, both Alice and Bob run an algorithm to determine which subsets of the boundary Δ are relevant. Alice determines for each relevant subset $S \subset \Delta$ if $G_A \cup S$ can be partitioned into triangles. Alice then communicates which of these S allow Alice to find a partition. Next, for each of these S , Bob computes if $G_B \cup (\Delta \setminus S)$ can also be partitioned into triangles. Finally, Bob determines that G can be partitioned into triangles if and only if there was such a S such that both Alice and Bob could partition their respective parts into triangles.

Before going into further detail, we will give a quick reminder on the problem setting. Alice and Bob have to decide whether a graph G can be partitioned into triangles, however either player only has a part of the graph. Specifically, for some set $A \subset V(G)$, Alice only has access to the graph $G_A = G[N_G[A]]$ and Bob only has access to $G_B = G[N_G[B]]$ where $B = V(G) \setminus A$. Observe that any triangle containing a vertex in A is contained in G_A , and any triangle containing a vertex from B is contained in G_B . Note that there is a set of vertices, which we will denote by Δ , to which both players have access. We call this set the boundary.

We call $S \subset \Delta$ a *witness* of a partition P if every triangle in P is contained either entirely in $A \cup S$ or entirely in $B \cup (\Delta \setminus S)$. For any witness S , both $G[A \cup S]$ and $G[B \cup (\Delta \setminus S)]$ can be partitioned into triangles.

The algorithm used to determine which subsets $S \subset \Delta$ are relevant is crucial for both the complexity and the correctness of the protocol. In order to guarantee correctness, it is essential that for any graph which can be partitioned into triangles, at least one witness of a partition is chosen. On the other hand, the communication complexity of the protocol scales linearly with the number of relevant subsets, so we want an algorithm which chooses a small number of those.

We will use a branching algorithm to decide which subsets of the boundary are relevant. This algorithm will branch by labeling the vertices in the boundary with labels in $\{0, 1\}$. Intuitively, label 0 corresponds with a vertex being assigned to Alice, and label 1 corresponds to a vertex being assigned to Bob. When we say we give a vertex the *trivial label* we mean the label 0 if the vertex is in A and the label 1 if the vertex is in B . Each vertex in the branching tree will have a partial labeling of the vertices from the boundary. Each leaf of the branching tree will have a complete labeling of the vertices in the boundary, from which we define S to be the set of vertices which are labeled by zero.

Let $\ell : L \rightarrow \{0, 1\}^{|L|}$ be a labeling of some $L \subset \Delta$ and let T be a triangle in G . We say a triangle is *local* if it is contained entirely in A or entirely in B . We say ℓ has labeled T *partially*

if there exist vertices $u, v \in T \cap \Delta$ such that $u \in L$ and $v \notin L$. Furthermore, we say ℓ has labeled T *inconsistently* if there exists $u, v \in T \cap L$ such that $\ell(u) \neq \ell(v)$. Finally, we say ℓ has labeled T *incorrectly* if $T \subset A$ and there exists a vertex $v \in T$ with $\ell(v) = 1$ or, symmetrically, $T \subset B$ and there exists a vertex $v \in T$ with $\ell(v) = 0$. Note that only local triangles can be labeled incorrectly. We say a labeling ℓ and a partition P are *consistent* if no triangle in P is labeled inconsistently or incorrectly, and no triangle in P is both non-local and partially labeled. Note that if ℓ is a complete labeling of Δ and consistent with a partition P , then the set $S = \{v \in \Delta \mid \ell(v) = 0\}$ is a witness of P .

Let G_Δ^* be the bipartite graph induced by the vertex sets $A \cap \Delta$ and $B \cap \Delta$. To be precise, let $E' = \{e \in (A \cap \Delta) \times (B \cap \Delta) \mid e \in E(G)\}$ and $V' = (A \cap \Delta) \cup (B \cap \Delta)$, then $G_\Delta^* = (V', E')$. For a partial labeling ℓ of Δ , we define the graph G_ℓ as the subgraph of G_Δ^* induced on the set of vertices not yet labeled by ℓ . The parameter of a vertex in the branching tree is the number of edges in G_ℓ , where ℓ is the labeling associated with that vertex of the branching tree. Whenever we refer to the degree of vertices in the branching algorithm, we mean the degree in G_ℓ .

The branching algorithm is defined by five rules, which should always be applied in order (eg. rule three should only be applied if neither rule one or two can be applied). For each rule, we will also prove that if there is a partition consistent with the partial labeling at a node on which that rule was matched, one of the children of that node in the branching tree also has a labeling consistent with a partition. In order to bound the number of relevant subsets generated we will analyze the branching vector for each of the rules which branch into multiple scenarios.

Rule 1. *Suppose a vertex v has degree zero. Then we label v as trivially.*

Proof. Suppose there exists a partition P consistent with the current labeling ℓ . Let ℓ' be the labeling extending ℓ by labeling v trivially. Let T be the triangle in P containing v . Since v has degree zero in G_ℓ , T must be local. Therefore, ℓ' cannot have labeled T incorrectly. Furthermore, since ℓ did not label T incorrectly, ℓ' must also label T consistently. Finally, even though we may have partially labeled T , ℓ' is still consistent with P since T is local. \square

Since this is a reduction rule rather than a branching rule, we do not need to discuss its impact on the number of leaves in the branching tree.

Rule 2. *Suppose a vertex v has degree one and its only neighbour w also has degree one. Then we label both v and w trivially.*

Proof. Suppose there exists a partition P consistent with the current labeling ℓ . Let ℓ' be the labeling extending ℓ by labeling v and w trivially. Let T be the triangle in P containing v and T' the triangle in P containing w .

If $T = T'$, then clearly it is non-local. Furthermore, it must contain a third vertex adjacent to both v and w . Since both vertices have degree one in G_ℓ , the third vertex of this triangle must already be labeled. This would imply ℓ partially labeled the non-local triangle T , which conflicts with our assumption that ℓ is consistent with P . Thus we can conclude $T \neq T'$. Furthermore, both triangles T and T' must be local. By similar reasoning as in the proof of Rule 1, we can conclude P and ℓ' are consistent. \square

Similar to the previous rule, this is again a reduction rule rather than a branching rule and thus cannot increase the number of leaves in the branching tree.

Rule 3. *Suppose a vertex v has $k \geq 2$ neighbours of degree one. Then we branch into $2 + d(v) - k$ different scenarios. In the first, we label v and all neighbours of degree one with the trivial label of the degree one vertices. In the second, we label all degree-one neighbours trivially and leave v unlabeled. In the other $d(v) - k$ branches we each label a different neighbour u with $d(u) \geq 2$, the vertex v and all its degree-one neighbours with the trivial label of the degree-one neighbours.*

Proof. Let w_1, \dots, w_k be the degree-one neighbours of v and let u_1, \dots, u_m the other neighbours of v (with $m = d(v) - k$). Suppose there exists a partition P consistent with the current labeling ℓ . Let ℓ'_1 be the labeling extending ℓ by labeling v and w_1, \dots, w_k with the trivial label of w_1 . Let

ℓ'_2 be the labeling extending ℓ by labeling w_1, \dots, w_k trivially. For $i \in [m]$, let ℓ'_{i+2} be the labeling extending ℓ by labeling v, u_i and w_1, \dots, w_k with the trivial label of w_1 . Let T be the triangle in P containing v .

Suppose T doesn't contain any of the vertices w_1, \dots, w_k . In that case, each of w_1, \dots, w_k must be contained in local triangles. Thus by arguments as in 1, ℓ'_2 is consistent with P .

Suppose T contains exactly one of the vertices w_1, \dots, w_k . Without loss of generality, let that vertex be w_1 . This means w_2, \dots, w_k must be contained in local triangles, and thus labeling them trivially is still consistent with P . Furthermore, the third vertex of T must be unlabeled by ℓ , since T is non-local. Thus it must be a neighbour of either v or w_1 in G_ℓ , which means it must be one of u_1, \dots, u_m . If $u_i \in T$, the labeling ℓ'_{i-2} must be consistent with P since it labels T entirely and consistently (and T cannot be labeled incorrectly since T is non-local).

Suppose T contains two of the vertices w_1, \dots, w_k . Without loss of generality, let these vertices be w_1, w_2 . This means w_3, \dots, w_k must be contained in local triangles, and thus labeling them trivially is still consistent with P . Furthermore, since T is both consistently and entirely labeled, we can conclude ℓ'_2 is consistent with P . \square

In the branch where we label both v and its degree one neighbours, the number of edges between unlabeled vertices is reduced by $d(v)$. In the branch where we only label the degree one vertices, the number of edges between unlabeled vertices is only reduced by k . In the branch where u_i is labeled, the parameter is reduced by $d(v) + d(u_i) - 1$. Since $d(u_i) \geq 2$, this is at least $d(v) + 1$. As such, the branching vectors associated with this rule are of length $d(v) - k + 2$, containing an entry with value k , an entry with value $d(v)$ and $d(v) - k$ entries with value $d(v) + 1$. Since $k \geq 2$, we can use numerical analysis to determine the worst branching vector of this form is $\{2, 3, 4\}$ which has an associated branching number of (approximately) 1.46557.

Rule 4. *Suppose a vertex v has degree one and its only neighbour w has degree d . Then we branch into d different scenarios. In the first, we only label v trivially. In the others, we label each vertex of the triple $\{u, v, w\}$ with the trivial label of v for the different neighbours u of w (except for v).*

Proof. Let the neighbours of w be $v, u_1, \dots, u_{d(w)-1}$. Suppose there exists a partition P consistent with the current labeling ℓ . Let ℓ'_1 be the labeling extending ℓ by labeling v trivially. Let ℓ_i for $i \in [2..d(w)]$ be the labeling extending ℓ by labeling the triple $\{u_{i-1}, v, w\}$ with the trivial label of v . Let T be the triangle in P containing v .

If T does not contain w , it must be a local triangle. As seen in the proof of 1, this implies ℓ'_1 and P must be consistent.

If T contains w , it must be a non-local triangle. Therefore, the third vertex of T must be unlabeled by ℓ . Since it must be adjacent to both v and w , it must be a neighbour of one of them in G_ℓ . This means it must be one of $u_1, \dots, u_{d(w)-1}$. If $u_i \in T$, the labeling ℓ'_{i-1} must be consistent with P since it labels T entirely and consistently (and T cannot be labeled incorrectly since T is non-local). \square

In the first branch, we reduce the parameter by only one. In the branch labeling $\{u, v, w\}$, the parameter is reduced by $d(u) + d(w) - 1$. Because rule 3 did not apply, we know the degree of u must be at least two. Thus the branching vector of this rule is of length $d(w)$, containing a single 1 and $d(w) - 1$ terms of $d(w) + 1$. Numerical analysis shows that the branching vector of this form with the largest associated branching number is the vector $(1, 4, 4)$ with a branching number of (approximately) 1.54369.

Rule 5. *Suppose either all vertices on one side have degree at least two (ie. either all vertices of A or all vertices of B). Let the number of vertices on that side be m . Then we branch into 2^m different scenarios. Each of these scenarios corresponds to a unique labeling of the m vertices on the side where all vertices have degree at least two and the trivial labeling of the vertices on the other side.*

Proof. Suppose there exists a partition P consistent with the current labeling ℓ . Suppose the side containing the degree-two vertices is A . Let ℓ' be the labeling extending ℓ with $\ell'(v) = 0$ if $v \in A$

and the triangle containing v is local and $\ell'(v) = 1$ otherwise. Clearly, all triangles are consistent and correct. Since all vertices are labeled, no triangle is partially labeled. Thus ℓ' is consistent with P . The case where the side containing the degree-two vertices is B is symmetric. \square

If our parameter is k , we know $m \leq k/2$ since all vertices have at least degree 2. Thus we branch into at most $2^{k/2}$ scenarios, each of which is a leaf in our branching tree. Thus the branching number of this rule is $\sqrt{2}$.

Lemma 6.1. *Given an instance I of the TRIANGLE PARTITION communication problem with parameter k , there exists an algorithm which finds a set $\mathcal{S} \subset \mathcal{P}(\Delta)$ such that $|\mathcal{S}| = \mathcal{O}(1.54369^k)$ and \mathcal{S} contains a witness if and only if I is a yes instance.*

Proof. Let I be an instance of TRIANGLE PARTITION with parameter k . The branching algorithm described above will produce a tree, where each leaf corresponds to a labeling of the vertices in the boundary. For each labeling, we obtain a subset of the boundary by selecting precisely the vertices labeled 0. Let \mathcal{S} be the set of all sets obtained by doing so for each leaf. Since each of the rules have a branching number of at most 1.54369, we can derive $|\mathcal{S}| = \mathcal{O}(1.54369^k)$. If I is a yes instance, there must exist some witness for it. Since each rule conserves the existence of a witness, this witness must be contained in \mathcal{S} . If I is a no instance, there cannot exist any witness, thus \mathcal{S} also cannot contain any. \square

We use this algorithm to complete the communication algorithm described at the start of this chapter. Both Alice and Bob will run the branching algorithm on the boundary separately, not requiring any communication. This gives them both the set of relevant subsets \mathcal{S} . For each $S \in \mathcal{S}$ Alice determines whether $G[A \cup S]$ can be partitioned into triangles and Bob determines whether $G[B \cup (\Delta \setminus S)]$ can be partitioned into triangles. Alice sends Bob a message containing for each $S \in \mathcal{S}$ whether Alice's part could be partitioned. Bob determines if there was any $S \in \mathcal{S}$ for which both players could partition their respective parts and outputs his findings.

Theorem 6.2. *Given an instance I of the TRIANGLE PARTITION COMMUNICATION with parameter k , there is a communication protocol which decides whether I is a yes instance with communication complexity $\mathcal{O}(1.54369^k)$.*

Proof. Let I be an instance of TRIANGLE PARTITION with parameter k . By lemma 6.1, \mathcal{S} contains a witness if and only if I is a yes instance. Suppose I is a yes instance, then both Alice and Bob will conclude their part of the graph can be partitioned for this witness. After receiving the message from Alice, Bob can conclude that this is a yes instance. If I is a no instance, then Bob will not find any witness, and conclude that I is a no instance.

The message requires one bit of information for each $S \in \mathcal{S}$, thus it has a length of $|\mathcal{S}|$. By lemma 6.1, this is at most $\mathcal{O}(1.54369^k)$. \square

Chapter 7

Conclusions

In this chapter we will give a short recap of the primary results and discuss the relations between them.

As a lower bound on the complexity of TRIANGLE PARTITION, we've shown that there cannot exist an $\varepsilon > 0$ such that TRIANGLE PARTITION can be solved in $\mathcal{O}^*((2 - \varepsilon)^{\text{ctw}(G)/2})$ time. We did this by analyzing the cutwidth of a graph constructed as a reduction from CNF-SAT to TRIANGLE PARTITION. The $\text{ctw}(G)/2$ in the exponent appears to be rather inherent to the problem, or at the very least to the approach taken in constructing the lower bound. Therefore we are skeptical to the existence of a better lower bound, even though we were unable to find a matching upper bound. Instead, the upper bound we managed to find is $\mathcal{O}^*(2^{\frac{3}{4}\text{ctw}(G)})$ by using the property that any graph that contains only edges contained in a triangle must have relatively large cutwidth compared to its pathwidth. Because we did not exploit the low cutwidth directly, but rather use the fact that it implies low pathwidth, we expect a more direct use may result in a better algorithm.

For TRIANGLE PARTITION COMMUNICATION we have shown a lower bound of $\Omega(\frac{3^{k/3}}{k})$ by constructing a graph for each monotone ternary boolean function and proving that a unique message is required for each these graphs. Note that this suggests that any dynamic programming approach where the columns represent different cuts will not surpass this bound. We also describe a communication protocol based on a branching algorithm which proves an upper bound on TRIANGLE PARTITION COMMUNICATION of $\mathcal{O}(1.54369^k)$. It seems plausible that this algorithm could be improved upon further, possibly even to match the lower bound we found.

One could try to formulate a dynamic programming approach using the messages from the communication protocol to define the rows, which would result in an improvement over the $\mathcal{O}^*(2^{\frac{3}{4}\text{ctw}(G)})$ algorithm we described. Unfortunately, due to the $\Omega(3^{k/3})$ lower bound on the communication complexity, even with improvements to the communication protocol, it will not be possible to find a $\mathcal{O}^*(2^{\text{ctw}(G)/2})$ algorithm using this method. Therefore if the goal is to find an optimal algorithm (and prove its optimality), we recommend to either approach the problem directly (ie. not using dynamic programming based on a communication strategy) or first attempt to find a lower bound matching the communication lower bound for the original problem.

Bibliography

- [1] Brenda S Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM (JACM)*, 41(1):153–180, 1994. 1
- [2] Noud A. W. M. de Kroon Bas A.M. van Geffen, Bart M. P. Jansen and Rolf Morel. Lower bounds for dynamic programming on planar graphs of bounded cutwidth. In *Proceedings of the 13th International Symposium on Parameterized and Exact Computation*, To appear. 2018. 1
- [3] Andreas Björklund. Exact covers via determinants. *arXiv preprint arXiv:0910.0460*, 2009. 1
- [4] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set partitioning via inclusion-exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. 1
- [5] Fan RK Chung. On the cutwidth and the topological bandwidth of a tree. *SIAM Journal on Algebraic Discrete Methods*, 6(2):268–277, 1985. 4
- [6] Moon-Jung Chung, Fillia Makedon, Ivan Hal Sudborough, and Jonathan Turner. Polynomial time algorithms for the min cut problem on degree restricted trees. *SIAM Journal on Computing*, 14(1):158–177, 1985. 4
- [7] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On problems as hard as CNF-SAT. *ACM Transactions on Algorithms (TALG)*, 12(3):41, 2016. 3
- [8] Marek Cygan, Fedor V Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*, volume 4. Springer, 2015. 4
- [9] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002. 1, 4
- [10] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k -SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. 3
- [11] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. 3
- [12] Bart M. P. Jansen and Jesper Nederlof. Computing the chromatic number using graph decompositions via matrix rank. In *Proceedings of the 26th European Symposium on Algorithms*, 2018. 1, 2
- [13] Bart M. P. Jansen and Jules J. H. M. Wulms. Lower bounds for protrusion replacement by counting equivalence classes. *arXiv preprint arXiv:1609.09304*, 2016. 2
- [14] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972. 1

- [15] Mikko Koivisto. Partitioning into sets of bounded cardinality. In *International Workshop on Parameterized and Exact Computation*, pages 258–263. Springer, 2009. 1
- [16] Ephraim Korach and Nir Solel. Tree-width, path-width, and cutwidth. *Discrete Applied Mathematics*, 43(1):97–101, 1993. 1, 4
- [17] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. In *Proceedings of the twenty-second annual ACM-SIAM symposium on Discrete Algorithms*, pages 777–789. Society for Industrial and Applied Mathematics, 2011. iii, 1, 2, 7, 9
- [18] Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Lower bounds based on the exponential time hypothesis. *Bulletin of EATCS*, 3(105), 2013. 3
- [19] Burkhard Monien and Ivan Hal Sudborough. Min cut is np-complete for edge weighted trees. *Theoretical Computer Science*, 58(1-3):209–229, 1988. 4
- [20] Herbert Robbins. A remark on Stirling’s formula. *The American mathematical monthly*, 62(1):26–29, 1955. 3
- [21] Atsushi Takahashi, Shuichi Ueno, and Yoji Kajitani. Mixed searching and proper-path-width. *Theoretical Computer Science*, 137(2):253–268, 1995. 4
- [22] Dimitrios M Thilikos, Maria Serna, and Hans L Bodlaender. Cutwidth i : A linear time fixed parameter algorithm. *Journal of Algorithms*, 56(1):1–24, 2005. 4
- [23] Johan MM van Rooij, Marcel E van Kooten Niekerk, and Hans L Bodlaender. Partition into triangles on bounded degree graphs. *Theory of Computing Systems*, 52(4):687–718, 2013. 1, 2