# Beating the Worst-Case: Analysis of a Practical Algorithm for Treewidth

Besser als im Worst-Case: Analyse eines Praktischen Algorithmus für Baumweite

**Marcus Wilhelm**

Universitätsmasterarbeit
zur Erlangung des akademischen Grades

Master of Science
*(M. Sc.)*

im Studiengang
IT Systems Engineering

eingereicht am 24. August 2020 am
Fachgebiet Algorithm Engineering der
Digital-Engineering-Fakultät
der Universität Potsdam

| | |
|---|---|
| **Gutachter** | Prof. Dr. Tobias Friedrich |
| | Prof. Dr. Anja Lehmann |
| **Betreuer** | Dr. Thomas Bläsius |
| | Maximilian Katzmann |

# Abstract

Many NP-hard graph problems can be solved in polynomial time if the input instance is a tree. The same holds for input graphs that have small treewidth, a parameter describing how tree-like a graph is. While many graph problems are fixed-parameter tractable when parametrized by treewidth, in most cases this requires the computation of a tree decomposition of the input graph, which is NP-hard as well.

By contrast, an algorithm (PID-BT), submitted to the PACE Challenge 2017 achieved surprisingly fast running times on real-world networks despite the problem's hardness. This points at a big gap between the theoretical understanding of the problem and the practical performance of the algorithm.

In this thesis, we work towards reducing this gap, focussing on hyperbolic random graphs (HRGs) as a realistic model of real-world networks. On the theoretical side, we start by giving an intuitive presentation of the algorithm and show a super-polynomial lower bound on its expected running time on HRGs. This indicates that the observed performance in practice might be due to the implementation's preprocessing. We confirm this via empirical experiments, that show that the employed greedy heuristics discard a large linear part of the graph. We also explain this theoretically, by proving the existence of a linear expected number of simplicial vertices in HRGs.

# Zusammenfassung

Viele NP-schwere Graphenprobleme können in Polynomialzeit gelöst werden, solange die Eingabe ein Baum ist. Selbiges gilt für Graphen mit kleiner Baumweite, einem Parameter der beschreibt, wie baumähnlich ein Graph ist. So sind viele Graphenprobleme festparameterberechenbar bei Parametrisierung nach Baumweite, jedoch benötigen die meisten dieser Ansätze eine Baumzerlegung der Eingabe, deren Berechnung ebenfalls NP-schwer ist.

Ein Algorithmus, genannt PID-BT, welcher bei der PACE Challenge 2017 eingereicht wurde, erzielte trotz der Schwere des Problems außerordentlich gute Laufzeiten. Dies weißt auf eine große Lücke zwischen dem theoretischen Verständnis des Problems und der tatsächlichen Leistungsfähigkeit des Algorithmus in der Praxis hin.

In dieser Arbeit versuchen wir daher diese Lücke zu verkleinern, wobei wir uns dabei auf hyperbolische Zufallsgraphen (HZGs) als realistisches Modell für Echtweltgraphen konzentrieren. Auf der theoretischen Seite geben wir zu diesem Zwecke eine neue intuitivere Beschreibung von PID-BT und beweisen außerdem eine superpolynomielle untere Schranke für die erwartete Laufzeit des Algorithmus auf HZGs. Dies weißt darauf hin, dass die gute Laufzeit der Implementierung auf deren Vorverarbeitung zurückzuführen ist, was wir auch experimentell bestätigen. Im Detail beobachten wir, dass die Vorverarbeitung einen linear großen Teil der Eingabe entfernt und können dies ebenfalls theoretisch erklären, indem wir eine erwartete lineare Anzahl simplizialer Knoten nachweisen.

# Contents

# 1 Introduction

Many problems that are NP-hard on general graphs, like for instance Vertex Cover, Independent Set, etc. are efficiently solvable on trees. Intuitively, these problems should not immediately become hard for inputs that are only slightly more complex than trees. Indeed, many hard problems are still efficiently solvable on, e.g. cactus graphs and series parallel graphs. The *treewidth* of a graph is a parameter that generalizes this idea, by describing how tree-like a graph is. The more a graph resembles a tree, the smaller its treewidth, with actual trees having treewidth 1 and cliques having treewidth $n - 1$. Treewidth is a central tool in the field of parametrized algorithms, and many hard graph problems are *fixed-parameter tractable* (FPT) when parametrized by treewidth. This means that they have an algorithm running in time $O(f(k) \cdot \text{poly}(n))$, where $k$ is the treewidth of the input graph and $f$ is a computable function. For example, a maximum independent set of a graph with treewidth $k$ can be found in $O\big(2^k \cdot \text{poly}(n)\big)$ time. Even more generally, for every problem that can be expressed with a formula in monadic second-order logic ($MSO_2$) there is an FPT algorithm parametrized by the treewidth of the input graph exists, as proven by Courcelle's theorem [Cou90].

Most of these algorithms, however, depend on a so-called *tree decomposition* as input. A tree decomposition is a tree-like structure that acts as a witness for the treewidth of a graph and that allows dynamic programming approaches similar to the ones used on actual trees. Unfortunately, deciding if the treewidth of a graph is at most $k$ is NP-complete [ACP87] and finding an optimal tree decomposition is NP-hard. Both problems are solvable with FPT running time parametrized by treewidth, but so far these approaches were of rather limited practical use.

A major motivation for the development of practically feasible treewidth algorithms was presented in the Parameterized Algorithms and Computational Experiments (PACE) Challenge. In the PACE challenge 2016 and 2017 participants were invited to implement algorithms that compute tree decompositions. The submissions were evaluated and ranked based on their performance on a number of graphs, motivating the participants to design algorithms that work well in practice.

The winner of PACE'16 was Hisao Tamaki, whose submission was based on an algorithm by Arnborg et al. [ACP87] that is also known as ACP algorithm. For the PACE Challenge 2017, Hisao Tamaki and Hiromu Ohtsuka extended an algorithm by Bouchitté and Todinca [BT02] (BT-algorithm). The resulting algorithm, which we call PID-BT, was very successful and won the second place in the challenge, falling behind a more optimized C++ implementation of Tamaki's algorithm from 2016. In particular, the relative performance of the algorithm was especially good on the harder evaluation instances. For example, the algorithm solved instances with thousands of vertices and tens of thousands of edges within minutes.

Considering that no polynomial algorithm for treewidth can exist unless $\mathbf{P} = \mathbf{NP}$, this performance is very surprising. Consequently, the question arises how it can be that the PID-BT algorithm works so well in practice, despite negative theoretical results like the NP-hardness of treewidth. In this thesis, we search for an answer to this question, by analysing and understanding the PID-BT algorithm. As the algorithm is particularly designed for practical application, this does not only include studying the algorithm itself but also how it relates to its input instances. In this respect, we focus on a specific class of input instances called real-world *scale-free* networks. These are networks from many different domains, such as social networks, communication networks, and protein interaction networks that are relevant to practical applications and that were observed to share certain characteristics. We need a model of these networks that is a realistic representation and also accessible to analysis. For this purpose, we use *hyperbolic random graphs* (HRGs), because they exhibit the most important traits of scale-free networks, such as a heterogeneous degree distribution, high clustering and small diameter. Additionally, hyperbolic random graphs are both mathematically tractable and can be efficiently generated, which allows us to analyse them theoretically as well as empirically.

Our experiments on sampled HRGs confirm the impressive running times of PID-BT on practical instances with thousands of vertices. Very importantly, we also observed that the preprocessing employed by the implementation plays a major role in the performance of the algorithm. Consequently, in this thesis, we do not only analyse the PID-BT algorithm itself, but also the heuristics used in the preprocessing of the PACE'17 implementation. One of our main contributions in this context is a proof that explains why the preprocessing is able to reduce the instance size by an asymptotically constant factor in expectation. Also, we contribute to the analysis of greedy heuristics for tree decompositions in a general way, by developing new sufficient conditions for the *safeness* of *separators* in graphs. To the best of our knowledge, our results constitute the first theoretical analysis of such greedy heuristics in general and on a practically relevant graph model such as hyperbolic random graphs.

We also analyse the PID-BT algorithm without the preprocessing and derive various bounds for the running time on a number of graph classes. This includes a super-polynomial lower bound for the expected running time on hyperbolic random graphs. Further, we also provide a novel way of presenting the algorithm itself. This is accomplished by framing it as a natural extension of the algorithm Tamaki submitted in the 2016 PACE challenge and by highlighting how the basic idea behind it can be seen from a perspective of the construction of partial solutions. Although this is not in itself a novel result, it is nevertheless scientifically valuable. One of the basic virtues of science is reproducibility. In maths and theoretical computer science the lucid, graspable, and maybe even elegant presentation of concepts, proofs, and algorithms is a core aspect of reproducibility. For this reason, our intuitive description of the algorithm constitutes another important contribution of this thesis.

## 1.1 Related Work

Treewidth was first introduced in 1986 by Robertson and Seymour via the definition of *tree decompositions* [RS86]. Apart from this original definition, there exists a multitude of equivalent characterizations of treewidth, some of which were discovered independently. Before Robertson and Seymour, Bertelè and Brioschi [BB72], as well as Halin [Hal76], invented definitions that are equivalent to treewidth. In this thesis, we use the definition via tree decompositions, a characterisation via graph searching games by Seymour and Thomas [ST93] and a definition via chordal graphs [PS97]. See [Bod98] for a survey of equivalent definitions and characterizations of treewidth.

The usefulness of treewidth comes from the fact that many generally hard problems are efficiently solvable on graphs of small treewidth. One common algorithmic application is dynamic programming on tree decompositions, which can be seen as a direct extension of folklore algorithms for, e.g. vertex cover or independent set on trees. There are also approaches in which no explicit witness of the treewidth is required, such as for $k$-path, feedback vertex set, and vertex cover. Most of these are based on structural properties (e.g. a graph of treewidth $k$ contains a $k$-path) or based on a theorem that states that graphs of large treewidth contain large grids as minor [RST94][CC16]. A survey by Bodlaender covers more algorithmic applications of small treewidth [BK07].

In summary, many theoretical algorithmic results require graphs with small treewidth in order to be feasible in practice. There are specific graph classes like, for instance, graphs arising from the control flow of certain (commonly used) programming languages, for which constant treewidth could be shown [Tho98]. However, for a long, time it was rather unclear how commonly graphs with small treewidth appear in general settings within the real world. A study by Maniou et al. [MSJ19] surveyed the treewidth of graphs from 25 datasets from 8 different domains and found that the tested graphs generally have a rather large treewidth, except for infrastructure networks which were found to exhibit a treewidth of $O(n^{2/3})$. Still, even such a sub-linear treewidth is very likely too large to run heavy exponential algorithms on these graphs in practice.

There are many algorithms for the exact computation of treewidth. A classical result by Arnborg, Corneil, and Proskurowski [ACP87] shows an algorithm that decides whether a graph has treewidth at most $k$ in time $O(n^{k+2})$. We refer to this algorithm as ACP algorithm. There are also algorithms that decide treewidth in $f(k) \cdot n^c$ time for a computable function $f$ and a constant $c$, fitting into the framework of fixed-parameter tractability by Downey and Fellows [DF99]. The first algorithm for treewidth with such a running time was proposed by Robertson and Seymour [RS86][RS95]. A notable improvement of this algorithm due to Bodlaender [Bod93] even runs in linear time for constant $k$, but is completely infeasible in practice due to large constants and its running time dependency in $k$. A different approach has been proposed by Bouchitté and Todinca [BT02]. Their algorithm, which we refer to as BT algorithm, decides treewidth in time polynomial in the number of minimal separators by listing all minimal separators and some other combinatorial structures. This algorithm is useful for a number of graph classes that all have a polynomial number of minimal separators. A different approach by Gogate and Dechter is based on a *branch and bound* technique and has been implemented and evaluated practically.

A problem with all algorithms mentioned above is that they are of very limited applicability in most practical circumstances. This is confirmed in an experimental study by Röhrig [Röh98] and indirectly also in other practical studies that either were not able to compute the treewidth of larger graphs [Bod+06] or that needed to rely on heuristics as the only way to obtain upper bounds for the treewidth [MSJ19].

The development of practically feasible treewidth algorithms was sparked by the Parameterized Algorithms and Computational Experiments (PACE) Challenge 2016 [Del+17]. The winning approach was developed by Hisao Tamaki based on the ACP algorithm. Bannach and Berndt [BB19] describe and generalize the algorithm, which we refer to as PID-ACP. It is based on an equivalent definition of treewidth that is explained in more detail in Section 3.1. In the PACE Challenge 2017 [Del+18], Hisao Tamaki and Hiromu Ohtsuka submitted an improved algorithm. The algorithm is based on the BT algorithm and Tamaki's formal description of the algorithm, *Positive-instance driven dynamic programming for treewidth* [Tam19], won a best paper award at ESA 2017.

Heuristics for upper and lower bounds seem to work a lot better than exact approaches in practice. A survey by Bodlaender gives a broad overview of different ways to compute treewidth or upper/lower bounds for treewidth both exactly and heuristically [Bod05]. This includes heuristic approaches that find upper bounds on the treewidth by greedily constructing an elimination scheme of the graph. In practice, such heuristics (e.g. the *Min-Fill* heuristic) seem to perform extremely well on many instances, often producing close to optimal tree decompositions. Experimental analyses of this phenomenon have been conducted by van Dijk et al. [DHS06] and Maniu et al. [MSJ19]. One of the preprocessing rules discussed by Bodlaender is based on so-called *safe-separators* [BK06], which we also explore.

Hyperbolic random graphs were first introduced by Krioukov et al. [Kri+10] and as an acronym of the authors' names are also referred to as KPKVB random graphs. In the original paper, the authors establish that the graphs described by this model share important characteristics of real-world networks, namely a *power-law degree distribution* and a high *clustering coefficient*. This means that the number of vertices with degree $k$ is proportional to $k^{-\beta}$ for a constant $\beta$ (usually between 2 and 3) called the *power-law exponent* and that vertices are more likely to be adjacent if they have a common neighbour. There are also experimental evaluations that show how closely real-world networks resemble hyperbolic random graphs. For instance, Boguñá et al. [BPK10] studied an embedding of an Internet graph into the HRG model, showing remarkable properties such as the high quality and resilience of a greedy routing strategy based on the geometry of the embedding. Apart from being a good representation of real-world networks, the model is also accessible to theoretical analysis. Gugelmann et al. [GPP12] gave rigorous proofs for the power-law degree distribution and high clustering coefficient as well as the average and maximum degree within the network. In further analyses, many more properties have been studied like the component structure [BFM13], the diameter [FK15][MS19], the clique size [BFK18], and the treewidth [BFK16]. Hyperbolic random graphs can be efficiently generated [Blä+19]. There is also a generalisation of hyperbolic random graphs called *geometric inhomogeneous random graphs* (GIRGs) that was introduced by Bringmann et al. [BKL17]. HRGs can be seen as closely related special cases of GIRGs in the sense that there is a coupling between GIRGs and HRGs that produces a GIRG that is very close to being a sub-graph and super-graph of the generated HRG.

There are also analyses of algorithms on HRGs. The motivation behind these is that many algorithms have been observed to perform better on typical real-world networks than on worst case graphs. Thus the analyses can be seen as an attempt to explain these observations in order to gain a better understanding of the underlying phenomena. For example, it has been shown that on HRGs the maximum clique can be found in polynomial time [BFK18], bidirectional breath-first search runs in sub-linear expected time [Blä+18], and that vertex cover can be solved in polynomial time using a reduction rule that also works well in practice [Blä+20]. Other studies consider and explain the effectiveness of greedy routing on HRGs [Bri+17].

## 1.2 Outline

In the remainder of this thesis we first introduce basic concepts and definitions and report on our experimental set-up and the empirical running time of the PID-BT algorithm on HRGs. Next, in Chapter 3, we present PID-BT as an intuitive extension of PID-ACP and give a theoretical analysis of the algorithm including upper and lower running time bounds on various graph classes. The preprocessing used for PID-BT is discussed in Chapter 4, along with both an empirical and theoretical analysis of its effectiveness. Finally, we conclude with a review of our results and some remarks about open questions and promising directions for future work in Chapter 5.

# 2 Preliminary Considerations

In this chapter, we introduce the notation and important definitions used throughout the thesis and provide an overview of the experimental set-up and the main findings of our empirical observations.

## 2.1 Basic Definitions and Concepts

For a set $X$ we write $\binom{X}{k}$ for the set of $k$-element subsets of $X$. Let $V$ be a finite set. Then for $E \subseteq \binom{V}{2}$, we call $G = (V, E)$ an *undirected graph*, or simply *graph* with *vertices* $V$ and *edges* $E$. If not clear from context, we write $V(G)$ for the set of vertices of $G$ and $E(G)$ for the set of edges. We generally assume a graph $G$ to be *simple*, that is we forbid self-loops or, equivalently, assume the relation on $V$ defined by $E$ to be irreflexive. Also, we often use $n$ for the number of vertices of a graph, if the graph we are referring to is clear from the context.

We say that two vertices $v, w \in V$ are *adjacent* if they have an edge, that is if $\{v, w\} \in E$. The *(open) neighbourhood* of a vertex $v$ is the set of all vertices $v$ is adjacent to, written as $N_G(v) = \{w \in V \mid \{v, w\} \in E\}$. The *degree* of a vertex written as $\deg_G(v) = |N(v)|$ is the size of its open neighbourhood. We define the *closed neighbourhood* of a vertex as the union of the open neighbourhood and the vertex itself, written as $N_G[v] = N(v) \cup \{v\}$. If the graph we are referring to is clear from context, we omit the subscript from our notation. This also applies to the definitions that follow.

The concept of open and closed neighbourhood is canonically extended to sets of vertices as follows. For a set $U \subseteq V$ of vertices the *open neighbourhood* is $N(U) = (\bigcup_{u \in U} N(u)) \setminus U$ and the *closed neighbourhood* is $N[U] = N(U) \cup U$.

A vertex set $U \subseteq V$ is called a *clique* if any two vertices $v, w \in U$ are adjacent. A graph whose vertices form a clique is called a complete graph, and we write $K_n$ for the complete graph with $n$ vertices. We say that the vertices $v_1, \ldots, v_k$ form a *cycle* of length $k$ if for $1 \leq i \leq k$ the vertices $v_i$ and $v_{i+1 \bmod k}$ are adjacent. If for all $v_1, \ldots, v_k$ contains no vertex twice, we call the cycle *simple*. A set of vertices $U \subseteq V$ forms an *induced cycle* if there is a simple cycle that covers all edges between vertices of $U$.

For a graph $G = (V, E)$ and a subset of vertices $U \subseteq V$ we denote the *(induced) sub-graph* as $G[U] = \left(U, E \cap \binom{U}{2}\right)$. Notation wise, if $H$ is a sub-graph of $G$ then $G$ is a *super-graph* of $H$. We write $G \setminus U$ as shorthand notation for $G[V \setminus U]$. Similar the *complement* graph $\overline{G}$ of $G$ is defined as $\overline{G} = \left(V, \left\{\{v,w\} \in \binom{V}{2} \mid \{v,w\} \notin E\right\}\right)$. In a slight abuse of notation we denote the graph derived by completing a vertex set $U \subseteq V$ to a clique with $G + \text{clique}(U) = \left(V, E \cup \binom{U}{2}\right)$. A *contraction* of an edge $\{v,w\} \in E$ is an operation that transforms $G$ to a graph $G' = (V', E')$ by replacing the vertices $v$ and $w$ with a new vertex $vw$ and modifying the edges so that $vw$ is adjacent with $N_G(v) \setminus \{w\} \cup N_G(w) \setminus \{v\}$. If a graph $G'$ can be obtained from $G$ through a series of contractions and deletions of edges or vertices, we call $G'$ a *minor* of $G$.

Further, a vertex $v \in V$ is called *simplicial* if its neighbourhood is a clique.

**Components, blocks, separators.** A *path* of length $l$ from $v_0$ to $v_l$ is a sequence of vertices $v_0, \ldots, v_l$ such that for $i \in 0 \ldots l-1$, $v_i$ is adjacent to $v_{i+1}$. A vertex set $C \subseteq V$ is called *connected* if for every $v, w \in C$ there is a path from $v$ to $w$. If $C$ is connected and no superset $C' \subseteq V$ of $C$ is connected, then we call $C$ a connected component of $G$. A connected graph that contains no cycles is called a *tree*.

Let $S \subseteq V$ be a set of vertices. We call the vertex set $C \subseteq V$ a *component associated* with $S$, if $C$ is a connected component in $G \setminus S$. Further, if removing $S$ from $G$ increases the number of connected components in the remaining graph, we call $S$ a *separator*. For a separator $S$ and a component $C$ associated with $S$, the pair $(S, C)$ is called a *block*. Next, we call a component $C$ associated with $S$ *full* if $N(C) = S$. Accordingly, a block $(S, C)$ is called *full* if $C$ is a full component associated with $S$. In this case we have $(S, C) = (N(C), C)$. See Figure 3.1 (a) for an example of full and non-full components. A separator $S$ is called *minimal* if there are at least two full components associated with it.

**Treewidth.** The treewidth of a graph can be defined via *tree decompositions*. A tree decomposition of a graph $G$ is a tuple $(T, \iota)$, where $T$ is a tree and $\iota : V(T) \mapsto 2^{V(G)}$ is a function mapping nodes of $T$ to sets of vertices of $G$ (called *bags*), such that

1. each vertex appears in some bag of the tree, that is for all $v \in V(G)$ there is a $t \in V(T)$ with $v \in \iota(t)$;

2. the endpoints of each edge are contained together in some bag, that is for all edges $\{u, v\} \in E(G)$ there is a node $t \in V(T)$ with $\{u, v\} \subseteq \iota(t)$;

3. the bags of each vertex form a subtree of $T$, that is for each vertex $v \in V(G)$ the set $\{t \in V(T) \mid v \in \iota(t)\}$ is connected in $T$.

For simplicity we call two bags $\iota(t_1), \iota(t_2)$ adjacent, if $t_1$ and $t_2$ are adjacent in $T$. The width of a tree decomposition $(T, \iota)$ is defined as the size of the biggest bag decremented by one, i.e. $\max\{|\iota(t)| \mid t \in V(T)\} - 1$. The treewidth $\mathrm{tw}(G)$ of $G$ is the minimum width of any tree decomposition for $G$.

The following equivalent characterisation of treewidth is important for Section 4.1. A graph $G$ is called *chordal* if every induced cycle in $G$ has length exactly three. We now define the *clique number* $\omega(G)$ as the size of the largest clique in a graph $G$ and the *chordal-width* of $G$ as the minimum clique number of any super-graph $G' = (V, E')$ with $E \subseteq E'$ of $G$ that is chordal. Then, the treewidth of $G$ is the chordal-width of $G$ decremented by one [PS97]

$$\mathrm{tw}(G) = \mathrm{chordal} - \mathrm{width}(G) - 1.$$

**Hyperbolic Random Graphs.** In order to define hyperbolic random graphs, we first need to introduce the *hyperbolic plane*. After choosing a point $O$ as the origin and some ray originating in $O$ as a reference for angles, any point $p$ of the hyperbolic plane is identified by its polar coordinates. We write $r(p)$ for the *radius* of $p$, which is the hyperbolic distance between $O$ and $p$ and $\theta(p)$ for the *angle* of $p$ which is the angle between between $p$, $O$, and the reference ray. Then the *hyperbolic distance* between two points $p$ and $q$ is

$$\mathrm{dist}(p, q) = \mathrm{acosh}(\cosh(r(p))\cosh(r(q)) - \sinh(r(p))\sinh(r(q))\cos(\Delta_\theta(p, q))),$$

where $\Delta_\theta(p, q)$ denotes the *angular distance* between $p$ and $q$ and $\cosh(x) = (e^x + e^{-x})/2$, $\sinh(x) = (e^x - e^{-x})/2$. In general all angles and arithmetic operations with angles are performed modulo $2\pi$ and positive angles between 0 and $\pi$ stand for anticlockwise rotations. The functions $\cosh(x)$ and $\sinh(x)$ are asymptotically approximated as $\frac{1}{2}e^x \pm o(1)$.

We write $B_p(r)$ for the *disk* of radius $r$ around point $p$, that is the set of points which have hyperbolic distance at most $r$. In cases in which the angle of the point is 0 (or not relevant), we simply write $B_{r_1}(r_2)$ for the disk of radius $r_2$ around point $(r_1, 0)$. The biggest intuitive difference between the Euclidean plane and the hyperbolic plane is that the hyperbolic plane somehow contains more space. For example, a Euclidean disk with radius $r$ has circumference $2\pi r$ and area $\pi r^2$, which are both polynomial functions in $r$. In contrast to that, a hyperbolic disk with radius $r$ has circumference $2\pi(\sinh(r))$ and area $2\pi(\cosh(r-1)$, which is exponential in $r$. There are several models that can visualize the hyperbolic plane in Euclidean space. For drawings of hyperbolic random graphs, we use the *native representation*, which works by simply treating hyperbolic coordinates as Euclidean polar coordinates. This way, shapes far away from the origin are distorted, leading to for example tear-drop shaped circles (see Figure 4.4 (a)).

We can now define *hyperbolic random graphs*, which are sampled by distributing $n$ points in the disk $B_O(R)$ according to a quasi-uniform distribution and connecting any two points if their hyperbolic distance is at most $R$. The radius $R$ of the disk is set to $R = 2\log n + C$ for a constant $C \in \mathbb{R}$. The coordinates of the $n$ points are sampled as follows. The angular coordinate is drawn uniformly at random from $[0, 2\pi]$ and the radius $r \in [0, R]$ of each point is drawn according to the probability density function

$$f(r,\theta) = f(r) = \frac{1}{2\pi}\frac{\alpha\sinh(\alpha r)}{\cosh\alpha R - 1} = \frac{\alpha}{2\pi}e^{-\alpha(R-r)}\left(1 + \Theta\left(e^{-\alpha R} - e^{-2\alpha r}\right)\right). \quad (2.1)$$

The constant $\alpha \in \left(\frac{1}{2}, 1\right)$ affects how strongly the probability density rises for values of $r$ closer to $R$. This affects the power-law exponent $2\alpha + 1$ of the degree distribution of the generated graph. The expected average degree of a hyperbolic random graph is constant and determined by Gugelmann et al. [GPP12, Lemma 2.3] as

$$\frac{2\alpha^2 e^{-C/2}}{\pi(\alpha - 1/2)^2}(1 + o(1)).$$

The $o(1)$ in the above formula comes from the fact that in general we are describing and analysing the asymptotic properties of hyperbolic random graphs for large $n$ under the assumption that $C$ and $\alpha$ are constant.

We will now state a number of useful lemmas that we use in our proofs and analyses. Many of these lemmas are helping us deal with the probabilities of events in which vertices fall or do not fall within certain regions of the disk.

In general, the probability with which a vertex lies in a region $A \subseteq D_0^R$ is given by the region's probability measure $\mu(A) = \int_A f(\theta, r) \mathrm{d}\theta \mathrm{d}r$. To calculate or simplify such an integral, the asymptotic approximation of the probability density function given in Equation (2.1) is useful.

Gugelmann et al. [GPP12] and Kromer [Kro17] further give equations for the measure of commonly needed regions.

▶ **Lemma 2.1 (Lemma 3.3 in [Kro17]).** For any $0 \le r, m \le R$ we have

$$\mu(B_0(r)) = e^{-\alpha(R-r)}(1 - \Theta(e^{-\alpha r})), \tag{2.2}$$

as well as

$$\mu(B_r(R) \cap B_0(R-m)) = \begin{cases} \mu(B_0(R-m)), \text{ if } r \le m, \\ \frac{4\alpha}{\pi(2\alpha-1)} e^{\frac{m-r}{2} - \alpha m} \cdot \varepsilon, \text{ if } r > m, \end{cases} \tag{2.3}$$

where $\varepsilon = 1 \pm O\left(e^{(m-r)(\alpha - \frac{1}{2})}\right)$. ◀

The first measure in the above lemma describes the region of a disk with radius $r$ around the centre. The second measure is the intersection of a disk with radius $R$ around a point with radius $r$ and a disk with radius $R - m$ around the origin. For $m = 0$ this region is exactly the region in which vertices are adjacent to a vertex $v$ at $(r, 0)$. We can thus call this the *region of the neighbourhood* of $v$. For larger values of $m$, the region is restricted to neighbours of radius at most $R-m$. By restricting the neighbourhood of $v$ to vertices with radius at most $r$, we can define the region of the *inner neighbourhood* of $v$ as $B_v(R) \cap B_0(r)$. Similarly, the region of the *outer neighbourhood* consists of all points of the region of the neighbourhood that have radius at least $r$. Using these definitions, we can also talk about the vertices in the inner/outer neighbourhood of $v$, which are subsets of the neighbourhood of $v$. If the context makes it clear what we are referring to, we often shorten this notation and simply write inner/outer neighbourhood for either the regions or the sets of vertices. This is also done with other regions of the disk, or respectively the sets of vertices lying in those regions, that we introduce in Section 4.4.

When calculating the measures of interesting regions, we often need to integrate over points in the region of the neighbourhood of a vertex. For this sake, it is necessary to know the outermost angles of the neighbourhood of a vertex at certain radii. Gugelmann et al. [GPP12, Lemma 3.1] gave a convenient asymptotic expression for this.

▶ **Lemma 2.2 (Lemma 3.1 in [GPP12]).** Let $v, w$ be vertices with radii $r_1$ and $r_2$ such that $r_1 + r_2 \geq R$. The maximum angular distance between $v$ and $w$ such that they are still adjacent is

$$\theta(r_1, r_2) = \arccos\left(\frac{\cosh(r_1)\cosh(r_2) - \cosh(R)}{\sinh(r_1)\sinh(r_2)}\right) = 2e^{\frac{R-r_1-r_2}{2}}\left(1 + \Theta\left(e^{R-r-y}\right)\right).$$

◀

We say that a vertex $v$ is *between* two vertices $u, w$, if the angle difference between $u$ and $v$ plus the angle difference between $v$ and $w$ equals the angle difference between $u$ and $w$. The following lemma gives us a simple rule for the adjacency of vertices that lie between adjacent vertices. This can be used to deduce that the left and right halves of the inner neighbourhood of a vertex form cliques (see Lemma 4.20).

▶ **Lemma 2.3 (Lemma 5.4 in [Kro17]).** Let $u, v, w \in V$ be vertices such that $v$ lies between $u$ and $w$, and let $\{u, w\} \in E$. If $r(v) \leq r(u)$, then $v$ is connected to $w$. ◀

We already saw that for a region $S$ of the disk and a fixed vertex $v$ the probability with which $v$ is in $S$ is given by the measure of $S$, that is $\Pr[v \in S] = \mu(S)$. Note that in this notation the vertex $v$ is treated as the point $(r(v), \theta(v))$. By defining indicator random variables $X_v$ that are 1 if $v \in S$, we can express the expected number of vertices that lie in $S$ as $E[\sum_{v\in V} X_v] = \sum_{v\in V} E[X_v] = \sum_{v\in V} \Pr[X_v = 1] = n \cdot \mu(v \in S)$. Krohmer gives the following equation for the probability with which at least one vertex lies in $S$ [Kro17]

$$\Pr[\exists v \in S] = 1 - (1 - \mu(S))^n \geq 1 - e^{-n\cdot\mu(S)}, \tag{2.4}$$

where $\exists v \in S$ is used as informal notation for the event in which at least one vertex lies in $S$. Similarly, we can derive the probability for the event in which no vertex falls into $S$ as

$$\Pr\left[\nexists v \in S\right] = (1 - \mu(S))^n \geq 1 - n \cdot \mu(S), \tag{2.5}$$

where again $\nexists v \in S$ is used as informal notation for the event in which no vertex lies in $S$. The estimate used for the inequality is however rather pessimistic. We derive a more accurate inequality for a special case in the lemma below.

▶ **Lemma 2.4.** Let $S$ be a region of the hyperbolic disk $B_O(R)$ such that there is a constant $c$ with $\mu(S) \leq \frac{c}{n}$. Then the probability with which no vertex falls into $S$ is asymptotically at least constant and given as ◀

$$\Pr\left[\nexists v \in S\right] \geq \left(1 - \frac{c}{n}\right)^n \geq e^{-c}\left(1 - \frac{c^2}{n}\right) \in \Omega(e^{-c}) \tag{2.6}$$

*Proof.* The basic exponential inequality $\left(1 + \frac{x}{n}\right)^n \geq e^x\left(1 - \frac{x^2}{n}\right)$ is commonly known and holds for $n \geq 1$ and $|x| \leq n$. The claimed statement follows by setting $x$ to $-c$. ■

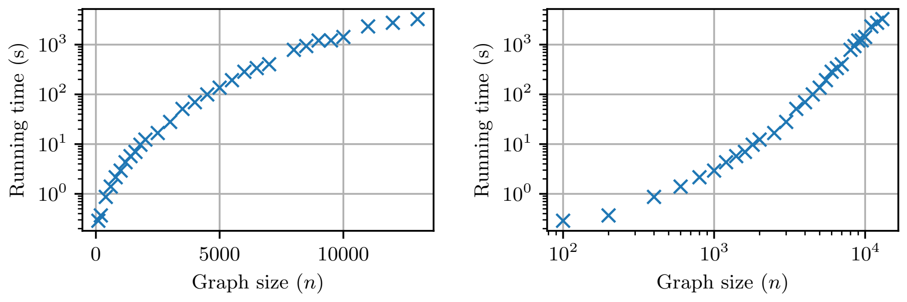
## 2.2 Empirical Running Time of PID-BT

When coming up with intuitions, hypotheses and ideas for proofs, it is always helpful to be able to quickly evaluate these ideas. In the case of hyperbolic random graphs, this can often be done experimentally by generating graphs and checking whether the assumption in question holds empirically. In the following, we present the basic set-up with which we conducted experiments and some of the main results in order to motivate the structure of the remainder of the thesis.

For the efficient generation of HRGs, we used the tool `girgs` [PW19], which is based on a paper by Bläsius et al. [Blä+19]. The implementation of PID-BT and its preprocessing was taken from the open-source repository [TO17] containing the original submission to the PACE challenge. The experiments were conducted on an ordinary laptop with 12GB RAM and Intel Core i7-5600U CPU.

The good performance of PID-BT on the instances of the PACE challenge already confirms the practical applicability of the algorithm to some extent. Still, good performance on more and realistic instances would make the algorithm even more interesting for further study. To that end, we conducted experiments about the performance of PID-BT with and without preprocessing on hyperbolic random graphs.

Figure 2.1 shows the run time of the PACE'17 implementation of PID-BT with the preprocessing enabled by default. For each $n$, the running time shown is the

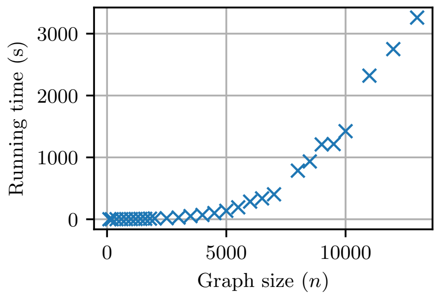


**(a)** Semilogarithmic plot. **(b)** Logarithmic plot.

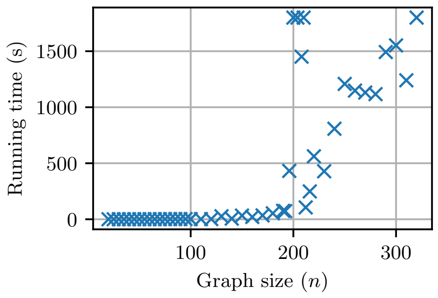
**Figure 2.1:** Running times of PID-BT with preprocessing on HRG instances with $n$ vertices, $\alpha = 0.75$, and average degree 10.

average over five random graphs generated with different random seeds. An exponential function would display as a straight in the semi-logarithmic plot. Thus the running time of the algorithm seems to be sub-exponential in $n$, as the observed running times depicted in Figure 2.1 (a) clearly follow a concave function. At the same time, polynomial functions of the form $f(n) = a \cdot n^b$ show as straight lines on the logarithmic plot. The observed run times depicted in Figure 2.1 (b) rather follow a convex function. This indicates that the running time of PID-BT is probably not polynomial. In summary, the runtime of the algorithm appears to be sub-exponential and super-polynomial. Given the NP-hardness of determining the treewidth of a graph, this confirms the algorithm as an interesting object of study.

In order to estimate how strongly the performance of the algorithm is influenced by the preprocessing, we conducted the same experiment without the preprocessing step. This was achieved by altering the source code of the PACE submission of PID-BT so that no safe separators are searched and found. We found that the algorithm's running time increases considerably if no preprocessing is applied. See Figure 2.2 for plots of the running times with and without preprocessing. In the experiments without preprocessing a time-out of 30 minutes was used and any execution time above that was truncated to 30 minutes. Note the range of graph sizes drawn on the x-axis of Figure 2.2 (b) is substantially smaller than in Figure 2.2 (a). It is clear that the preprocessing step plays an



**(a)** With preprocessing.



**(b)** Without preprocessing (time-out: 30m).

**Figure 2.2:** Comparison of running times of PID-BT with and without preprocessing on HRG instances with $n$ vertices, $\alpha = 0.75$, and average degree 10.

essential role in the performance of the algorithm. As a consequence, in this thesis we analyse the PID-BT algorithm and the preprocessing separately.

While Figure 2.2 shows the strong effect of the preprocessing, it does not provide any explanation of how this effect is achieved. To remedy this, our discussion of the preprocessing in Chapter 4 includes more fine grained experiments that show more clearly in which ways the preprocessing works well. Additionally, we include a few empirical results in our discussions of the theoretical analysis of the preprocessing in Section 4.4.

# 3 Positive-Instance Driven Dynamic Programming for Treewidth

The goal of this section is to present the PID-BT algorithm by Hisao Tamaki in a clean and intuitive way and to analyse its running time. We do so by viewing the algorithm as an extension of the PID-ACP algorithm described by Bannach and Berndt [BB19] that Tamaki submitted to the PACE challenge 2016. Both algorithms take a graph $G = (V, E)$ and a positive number $k$ and decide if the treewidth of $G$ is at most $k$. In the *yes* case, the algorithms also construct a tree decomposition of $G$. On a basic level, both algorithms are essentially dynamic programs that keep track of different combinatorial structures. The core idea that we apply to the descriptions of both algorithms is to view these combinatorial structures mainly as partial solutions of the problem. This way, the algorithms are understood as dynamic programs that build tree decompositions of subgraphs of the input instance. To the best of our knowledge, this perspective has so far not been explicitly used in the existing literature to understand or present these algorithms. Using this new perspective, we are able to derive interesting insights on the PID-ACP algorithm and describe the PID-BT algorithm in a very intuitive way.

In the remainder of this chapter, we first present the PID-ACP algorithm and derive a few insights about it by employing a partial solution perspective. We then proceed with our description of the PID-BT algorithm and give a short example to help the reader strengthen their intuitions about some of the combinatorial structures used by the algorithm. Finally, we conclude the section with a theoretical analysis that includes a super-polynomial lower bound on the expected running time of PID-BT on hyperbolic random graphs.

## 3.1 Graph Searching and Treewidth

The PID-ACP algorithm builds upon another equivalent characterisation of treewidth. This characterisation can be derived via a game in which cops try to catch a robber by jumping between vertices of a graph. At the beginning of the game, the cops player chooses (up to) $k$ vertices, which are occupied by cops. Afterwards, the robber player chooses a vertex on which the robber is placed. The game then proceeds in rounds until the robber is caught. In each round, the cop player chooses a subset of the cops that will be relocated, removes them from the graph, and announces the vertices on which they will be placed. The robber then may walk any number of steps along edges of the graph, while avoiding vertices currently occupied by cops. Subsequently, the cops are placed on the announced positions, and the next round begins. If at some point a cop is placed onto the vertex holding the robber, the robber is caught and the cop player wins. Furthermore, we say the cop player has a *winning-strategy* with $k$ cops on a graph $G$, if $k$ cops suffice to always catch the robber regardless of the robber players' moves. Seymour and Thomas [ST93] showed that a graph has treewidth at most $k$ if and only if there is a winning-strategy for $k + 1$ cops on that graph.

This equivalent characterisation of treewidth can be turned into an algorithm by simulating the states of the game and checking if there is a winning-strategy for the cops. This is the basis of the PID-ACP algorithm as described by Bannach and Berndt [BB19]. The ACP algorithm [ACP87] can also be seen as a simulation of the game even though this perspective is not the primary motivation behind the algorithm. The idea is to model states of the game as vertices of a *configuration graph* and moves as cop-edges and robber-edges in that graph. A state of the game can be represented as a *configuration*, a subset $C \subseteq V$ of vertices on which the robber can still move. This means that instead of the robber's exact position, only the components that may contain the robber are stored. In order to make sure that these configurations correspond to valid states of the game, only subsets $C$ with at most $k + 1$ neighbours (representing the positions of the cops) are allowed. The placement of a single cop is then modelled via a cop-edge from a configuration $C$ with $|N(C)| \leq k$ to $C' = C \setminus v$ for a vertex $v$. We call this a *jump move*. Note that turns in which multiple cops are moved are represented by chains of multiple cop-edges. The movement of the robber does not have to be modelled explicitly except when the robber's sub-graph is split into multiple components. Thus, for a configuration $C$ that consists of at least two connected components $C_1, \ldots, C_\ell$, there are robber-edges from $C$ to all $C_i$, $1 \leq i \leq \ell$. We call this a *reveal move*, as it reveals the component of the robber.

The *start configuration* of the game is the state containing all vertices of the original graph, as this represents a situation in which the robber can be on any vertex, and no cop has been placed. The *winning configurations* are all sets of single vertices with degree at most $k$. This is motivated by the fact that this represents a state of the game in which at least one cop is remaining and hence able to catch the robber.

We can check whether there is a winning strategy using a simple dynamic program that marks nodes in the configuration graph. We start by marking all winning configurations. Next, a configuration is marked if it has a cop-edge to an already marked configuration or if all of its outgoing robber-edges lead to marked configurations. This leads to the start configuration being marked, if and only if the cops have a winning strategy. In order to improve the performance, the algorithm does not explicitly compute the entire graph, but only the part of the graph that can be reached from a winning configuration.

In the worst-case, this can still lead to $\Omega(2^n)$ considered configurations, however it seems to be substantially less on some instances in practice. For instance, Bannach and Berndt show polynomial bounds for certain graph classes. As an example for a worst-case with $\Omega(2^n)$ configurations, consider a star with $n-1$ leaves. Here any subset of the leaves is valid as it has only one neighbour and it can be reached by a sequence of jump moves.

## 3.2 A Partial Solution Perspective on PID-ACP

The moves of the game that underlies the PID-ACP algorithm can be considered in the reverse direction, like in the dynamic program from the previous paragraph. This way, each configuration $C$ can be seen as a partial solution and each (reversed) move extends partial solutions. In particular, if a configuration $C$ is marked by the dynamic program, this guarantees the existence of a tree decomposition of width at most $k$ of $G[N[C]]$ with a bag containing $N(C)$ acting as an interface to the rest of the graph. Note that when taking this reversed perspective on the algorithm, then the winning configurations are what we start with and the start configuration is the target that the algorithm is working towards.

A reversed jump move represents adding one vertex $v \in V \setminus C$ to $C$ and can only take place if $|N(C \cup \{v\})| \leq k$. Looking at partial solutions, it now becomes clear why the size was limited by $k$ and not $k+1$. In order to construct a partial solution for $C \cup \{v\}$, the tree decomposition of $C$ is extended by connecting a new bag containing $N(C \cup \{v\}) \cup \{v\}$ to the old interface bag $N(C)$ and appending a new interface bag $N(C \cup \{v\})$. Only if $|N(C \cup \{v\})| \leq k$, it can be guaranteed that the intermediate bag contains at most $k + 1$ vertices. In analogy to nice tree decompositions, we call the reversed jump move an *insert* move.

A reversed reveal move simply takes two configurations $C$ and $D$, representing tree decompositions of $N[C]$ and $N[D]$ and creates a configuration that represents a tree decomposition of $N[C] \cup N[D]$, by simply joining the tree decompositions. For a reversed reveal move, $|N(C \cup D)| \leq k + 1$ has to hold as $N(C \cup D)$ is the new interface bag connected to the interface bags of the tree decompositions of $C$ and $D$. We call a reversed reveal move a *join* move.

The algorithm can now be seen as starting with the winning configurations and applying all possible insert and join moves until a configuration covering the whole graph (i.e., the start configuration) is found. Interestingly, for the computation of treewidth, a lot of these moves can be ignored.

In the following lemmas, we restrict the types of moves that are necessary to arrive at *reachable* configurations, i.e., configurations that can be reached from the winning configurations via a sequence of insert and join moves. Note that a configuration is reachable if and only if it is marked by the labelling procedure and that reachable configurations are vertex sets $C$ whose closed neighbourhood has a tree decomposition of width at most $k$ with a bag containing $N(C)$. In particular, we show that a reachable configuration can be reached by only using *adjacent* moves. We call an insert move adjacent if it adds a vertex $v$ to a configuration $C$ with $v \in N(C)$

▶ **Lemma 3.1.** Any configuration that is reachable can also be reached, when restricting insert moves to adjacent ones. ◀

*Proof.* Consider a configuration that can be reached by a sequence of moves $\mathcal{S}$. Suppose $\mathcal{S}$ contains a non-adjacent insert move that transforms a configuration $C$ to a configuration $C'$. Then $C' = C \cup \{v\}$ for some vertex $v \notin N[C]$ and $|N(C')| \leq k$. This means that $|N(v)| \leq k$ and thus $\{v\}$ is one of the winning configurations the game starts with. This means that the insert move can be replaced by a join move that joins $C$ and $\{v\}$. This can be done to any non-adjacent insert move, and thus the lemma follows. ■

Similarly, we call a join move adjacent, if it joins two configurations $C$ and $D$ with $N(C) \cap N(D) \neq \emptyset$.

▶ **Lemma 3.2.** Assume the input graph to be connected. If the start configuration is reachable, then it can also be reached, when restricting join moves to adjacent ones. ◀

*Proof.* Assume the start configuration can only be reached by using non-adjacent join moves. Among all sequences of moves leading to the start configuration with minimum number of non-adjacent join moves, choose a sequence $\mathcal{S}$ such that the number of moves after the last non-adjacent join move is minimal. In the following, we show that we can contradict the choice of $\mathcal{S}$ by lowering the number of non-adjacent join moves or the number of moves after the last non-adjacent join move $M$ in $\mathcal{S}$. Let $A$ and $B$ be the configurations joined in $M$. We have three cases:

1. The move after $M$ is an insert move. This means that $M$ adds a vertex $v$ to $A \cup B$. As $N(A) \cap N(B) = \emptyset$, $|N(A \cup \{v\})| \leq |N(A \cup B \cup \{v\})|$. This means that $v$ could also have been inserted into $A$ before the join with $D$, reducing the number of moves that follow after the join or making the join adjacent.

2. The move after $M$ is an adjacent join move that joins $A$ and $B$ with a configuration $C$ that is, without loss of generality, adjacent to $A$. Then, similarly to case 1, $|N(A \cup C)| \leq |N(A \cup B \cup C)|$. The joins can be rearranged, so that $A$ and $C$ are joined prior to the join with $B$, again reducing the number of non-adjacent joins or the number of moves that follow after $M$.

3. $M$ is the last move. In a connected graph, the result of a non-adjacent join is a configuration with non-empty neighbourhood and, thus, cannot be the start configuration. This means that a non-adjacent join cannot be the last move, making this case impossible.

Both in case 1 and case 2, the moves can be rearranged so that the next move after $M$ occurs before $M$. This always reduces the number of moves after $M$ and might make $M$ adjacent if the added vertices are adjacent to $C$ and $D$, contradicting the choice of $\mathcal{S}$. ■

▶ **Corollary 3.3.** Assume the input graph to be connected. If the start configuration is reachable, then it can also be reached using only adjacent moves. ◀

*Proof.* Let $\mathcal{S}$ be a sequence of moves leading to the start configuration. First use Lemma 3.1 to replace all non-adjacent insert moves in $\mathcal{S}$. Then use Lemma 3.2 to remove all non-adjacent join moves. This works, because the proof of Lemma 3.2

does not introduce new non-adjacent insert moves: the only step at which a non-adjacent insert move might be created is in case 1 of the proof. However if adding $v$ to $A \cup B$ is non-adjacent, then adding $v$ to either $A$ or $B$ must be non-adjacent as well, contradicting the statement that all insert moves are adjacent. ■

This means that the partial solution perspective helped us discover a way in which the graph searching algorithm could be sped up, by ignoring non-adjacent moves in the labelling procedure.

## 3.3 A Partial Solution Perspective on PID-BT

In this section, we explain the PID-BT algorithm Tamaki submitted in the 2017 PACE challenge. In our description, we show how the algorithm can be seen as an extension or adaptation of the PID-ACP algorithm described and analysed in the previous sections. The motivation behind the adaptations is to reduce the number of partial solutions the algorithm has to consider before finding a valid solution for the entire input instance or disproving the existence of such a solution. This is accomplished by two main ideas.

The first one is that a total ordering of the vertices is assumed and partial solutions are explored based on this order. This helps in restricting the number of ways in which a partial solution can be derived from other partial solutions.

The second idea is that the partial solutions themselves are structurally restricted. The graph searching algorithm discussed in the previous sections considers partial solutions whose interface corresponds to arbitrary reachable states of the cops and robber game. The PID-BT algorithm improves on this by restricting the interfaces of partial solutions to vertex sets with more narrow combinatorial properties. This restriction was already part of the BT algorithm by Bouchitté and Todinca [BT02]. The difference is that the BT algorithm considers arbitrary partial solutions that have the combinatorial properties, while PID-BT only considers positive partial solutions that additionally correspond to a sub-graph with treewidth at most $k$.

In the following, we explain the combinatorial property used by the BT and PID-BT algorithms and how it is used in combination with a vertex ordering to restrict the order in which partial solutions are explored. As a reminder, note that the algorithms receive a graph $G = (V, E)$ and an integer $k > 0$ as input and decide whether $\mathrm{tw}(G) \leq k$.

The combinatorial constraint placed on the interface of partial solutions is that they have to be *potential maximal cliques*. A vertex set $\Omega$ is called a potential maximal clique (PMC) if there is a minimal set of edges that completes $G$ into a chordal super-graph of $G$ in which $\Omega$ is a maximal clique. A tree decomposition $(T, \iota)$ of $G$ is called *canonical*, if every bag $\iota(t)$ is a potential maximal clique and for every pair of adjacent bags $\iota(t_1), \iota(t_2)$ the vertex set $\iota(t_1) \cap \iota(t_2)$ is a minimal separator in $G$. We later discuss how PMCs can be recognized in polynomial time, but first, we discuss a more fundamental statement. The following lemma is important both for the BT algorithm and PID-BT.

▶ **Lemma 3.4 (Lemma 1 in [Tam19]).** Let $G$ be a graph with $\text{tw}(G) = k$. Then there is a canonical tree decomposition of $G$ with width $k$. ◀

This means that in order to show that a graph has treewidth at most $k$, it suffices to search for a canonical tree decomposition of width $k$. The BT algorithm does this by enumerating all potential maximal cliques of size up to $k + 1$ and all minimal separators of size up to $k$. The algorithm then tries to combine PMCs and minimal separators into valid partial solutions, which allows it to decide treewidth in time proportional to the product of the number of minimal separators, the number of PMCs and some polynomial factors in $n$. PID-BT works similarly, but does not need to list all PMCs and minimal separators. Instead, only relevant PMCs and minimal separators are considered that arise from already found valid partial solutions. This way, partial solutions are restricted to positive ones that correspond to sub-graphs of treewidth at most $k$, hence the name *positive-instance driven*. Below, we first explain the intuition behind how this is achieved using a total ordering of the vertices to orient the partial solutions, before going into more exact details, definitions, and lemmas.

Let $<$ be the aforementioned (strict) total order of $V$ and let $\min(U)$ stand for the minimum vertex of a vertex set $U \subseteq V$ according to $<$. We say that a vertex set $U_1$ *precedes* a vertex set $U_2$ if $\min(U_1) < \min(U_2)$. The goal is to let the algorithm process the graph and partial solutions in a descending way, first building partial solutions consisting of vertices high in the ordering and working towards partial solutions that include more minimal vertices.

To clarify this rough intuition, consider a PMC $\Omega$. We said that PMCs are the interfaces of partial solutions (compare Section 3.2). This means that $\Omega$ represents the interface of a partial solution. It remains to establish which other parts of the graph are included in that partial solution. We want the algorithm to go through the vertices in descending order. To that end, a connected set $C$ is called *inbound*, if there is a full block $(N(C), D)$ associated with $N(C)$ such that $D$ precedes $C$. Otherwise, $C$ is called *outbound*. Following the intuition, we want $\Omega$ to represent a partial solution that excludes the associated outbound components and includes the inbound ones. For technical reasons, we also have to exclude components whose neighbourhood is the subset of the neighbourhood of an outbound component. This is less complicated than it seems, because for two outbound components $A_1, A_2$ associated with a PMC, either $N(A_1) \subseteq N(A_2)$ or $N(A_2) \subseteq N(A_1)$ holds [Tam19, Lemma 4]. This means that $\Omega$ has an associated outbound component $A$ with maximal neighbourhood $N(A)$. This lets us define the *outlet* of $\Omega$, as $\text{outlet}(\Omega) = \emptyset$, if $\Omega$ does not have an associated non-full outbound component, and as $\text{outlet}(\Omega) = N(A)$ for the maximal neighbourhood $N(A)$ of any outbound component $A$ associated with $\Omega$. We call all other components, whose neighbourhood is not a subset of the outlet, the *support* of $\Omega$ ($\text{support}(\Omega)$) as these are the components that contribute to the partial solution represented by $\Omega$. See Figure 3.1 (c) for a visual depiction of a PMC with outlet and support. All components in the support are inbound. In summary, the PMC $\Omega$ represents a partial solution of $G$ without all components whose neighbourhood is a subset of the outlet, or equivalently of $\Omega$ and all components in the support of $\Omega$.

This partial solution is positive or valid if the graph induced by $\Omega$ and the components in its support has a tree decomposition of width at most $k$ that has a bag containing $\Omega$. Consequently, we call $\Omega$ $k$-*feasible* if $|\Omega| \leq k + 1$ and for each component $C_i \in \text{support}(\Omega)$, $G[N[C_i]]$ has a tree decomposition of width at most $k$ with a bag containing $N(C_i)$. This is motivated by the fact that partial solutions of the $N[C_i]$ can be joined together at their interface bags $N(C_i)$ with a new bag containing $\Omega$, resulting in a tree decomposition of $\Omega$ and its support. In the case in which $\text{outlet}(\Omega) = \emptyset$, the support of $\Omega$ contains all components associated with $\Omega$. This means that a valid partial solution represented by a PMC with an empty outlet is a valid solution for the whole graph. The following lemma confirms this intuition.

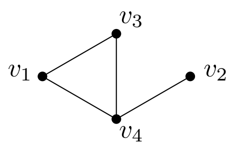
▶ **Lemma 3.5 (Lemma 5 in [Tam19]).** The treewidth of a graph is at most $k$ if and only if it contains a $k$-feasible potential maximal clique with empty outlet. ◀

In consequence, the goal of the algorithm is to either find a $k$-feasible PMC with empty outlet or to disprove the existence of such a PMC. The algorithm starts with trivially $k$-feasible PMCs consisting of the closed neighbourhood of a single vertex and tries to find new ones that cover larger parts of the graph. The core of the algorithm lies in recurrent relationships between PMCs and auxiliary structures called *I-blocks* and *O-blocks*. These recurrences make it possible to enumerate all $k$-feasible PMCs, in a way that relies only on already found $k$-feasible PMCs and $k$-feasible I-blocks and O-blocks. In the following, we define these structures, explain how they fit in with our understanding of PMCs, and describe the recurrences used by the algorithm.

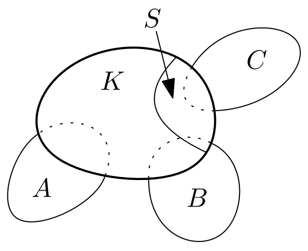
I-blocks and O-blocks are based on the definitions of inbound and outbound components and generalize our concept of partial solutions. We call a full block $(N(C), C)$ with $|C| \leq k$ an *I-block*, if $C$ is inbound. If otherwise $C$ is outbound, it is an *O-block*. Note that the neighbourhood of an inbound component $C$ is a minimal separator, as there must be another component $D$ with $N(D) = N(C)$ that precedes $C$. Consequently, if the neighbourhood of a component is not a minimal separator, then the component is outbound. Also, observe that a vertex set $S$ has at most one outbound associated full component $A$ with $N(A) = S$, while all other associated full components are inbound. We call an O-block *whole* if its neighbourhood is a minimal separator and *split* otherwise. The concept of a split O-block seems quite different from that of a whole O-block or I-block. Indeed, an outbound component that is associated with a PMC is always whole [Tam19, Lemma 3]. However, we still need the definition of split O-blocks, as they are needed in order to find all $k$-feasible PMCs. Also, split and whole O-blocks are given the same name in accordance with the notation in the original publication [Tam19].

Next, we extend the notion of feasibility to I-blocks and O-blocks. A connected vertex set $C$ is called $k$-feasible if there is a tree decomposition of $G[N[C]]$ of width at most $k$ with a bag containing $N(C)$. An I-block $(N(C), C)$ is called $k$-feasible, if $C$ is $k$-feasible. An O-block $(N(A), A)$ is $k$-feasible, if $N(A)$ is the union of the neighbourhoods of $k$-feasible inbound components. This way, $k$-feasible I-blocks and O-blocks naturally extend our intuition about the feasibility of PMCs and the concept of partial solutions.

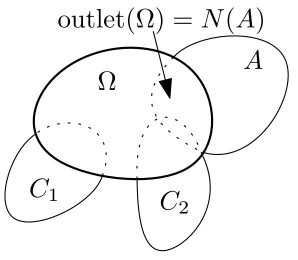
Before going into the details of the recurrences used by the algorithm, we show how PMCs can be identified in polynomial time and introduce a definition that makes it easier to refer to the vertices that are part of the partial solution represented by a PMC. A vertex set $S$ is called *cliquish* if for all pairs of distinct



**(a)** Examples for components and separators. $\{v_1\}$ is a full component associated with $\{v_3, v_4\}$, $\{v_2\}$ a non-full one. Also $\{v_2\}$ is a full component associated with the minimal separator $\{v_4\}$.



**(b)** Example for a crib, $\mathrm{crib}(S,K) = A \cup B \cup K \setminus S$. $C$ is not part of $\mathrm{crib}(S,K)$ as its neighbourhood is contained in $S$.



**(c)** Example a PMC $\Omega$, where $A$ is an outbound component assoc. with $\Omega$ s.t. $\mathrm{outlet}(\Omega) = N(A)$ and $\mathrm{support}(\Omega) = \{C_1, C_2\}$

**Figure 3.1:** Visual examples for some of the definitions needed for PID-BT.

vertices $u, v \in S$, $u$ and $v$ are adjacent, or there is a component $C$ associated with $S$ such that $u, v \in N(C)$.

▶ **Lemma 3.6 (Lemma 2 in [Tam19]).** A separator $S$ of $G$ is a potential maximal clique of $G$ if and only if (1) $S$ has no full component associated with it and (2) $S$ is cliquish. ◀

This can easily be checked in polynomial time. Next, consider a PMC $\Omega$ with outlet $S$. If $\Omega$ is $k$-feasible, this stands for a partial solution of $\Omega$ and all components associated with $\Omega$, whose neighbourhood is not a subset of $S$. In order to introduce a short notation for this, we generalise it as follows. For $K \subseteq V$ and $S \subset K$, we define the *crib* of $S$ with respect to $K$, written as $\mathrm{crib}(S,K)$, as the union of $K \setminus S$ and all components $C$ associated with $K$ whose neighbourhood $N(C)$ is not a subset of $S$. See Figure 3.1 (b) for a visual depiction of a crib. For a cliquish $K$ and $S \subset K$, $\mathrm{crib}(S,K)$ is a full component associated with $S$ [Tam19, Lemma 3]. We can now say that the feasibility of $\Omega$ stands for the existence of a partial solution of $\mathrm{crib}(\mathrm{outlet}(\Omega), \Omega)$.

This intuition is nicely confirmed by the following statement, which also gives us a recurrence between $k$-feasible I-blocks and PMCs.

▶ **Lemma 3.7 (Lemma 7 in [Tam19]).** An I-block $(N(C), C)$ is $k$-feasible if and only if there is some $k$-feasible PMC $\Omega$ with $\mathrm{outlet}(\Omega) = N(C)$ and $\mathrm{crib}(outlet(\Omega), \Omega) = C$. ◀

This way, we can find $k$-feasible I-blocks, based on found $k$-feasible PMCs. Any $k$-feasible whole O-block is also easily found, as the neighbourhood of any inbound component has an associated outbound component. Split O-blocks can be found by looking for full components associated with the union of the neighbourhoods of an inbound and an outbound component that already have been found.

In order to describe the recurrences used to find all $k$-feasible PMCs, we define *buildable* PMCs as a more general type of PMC that includes $k$-feasible PMCs, while being easier to construct. We say a PMC $\Omega$ with $|\Omega| \leq k+1$ is $k$-buildable if one of the following cases applies. Either

1. $\Omega = N[v]$ for a $v \in V$,
2. there is a subset $C$ of $support(\Omega)$ such that $\Omega = \bigcup_{D \in C} N(D)$ and every member of $C$ is $k$-feasible, or
3. $\Omega = N(A) \cup (N(v) \cap A)$ for some $k$-feasible O-block $(N(A), A)$ and a vertex $v \in N(A)$.

Tamaki proves that all $k$-feasible PMCs are $k$-buildable [Tam19, Lemma 9]. Thus, by finding all $k$-buildable PMCs the algorithm is guaranteed to also find all $k$-feasible PMCs.

With all the foundations set, we can now describe the algorithm in detail. The basic idea is to list all $k$-buildable PMCs and all $k$-feasible I-blocks and O-blocks, using previously found structures for the discovery of new ones. To that end, the algorithm maintains collections of all found PMCs, $k$-feasible PMCs, and $k$-feasible I-/ and O-blocks. In the following description of the algorithm, we assume that newly discovered structures are implicitly added to these collections and eventually processed by the applying procedures. To start, all case 1 $k$-buildable PMCs $\Omega$ with $|\Omega| \leq k+1$ are registered by checking for each $v \in G$ if $N[v]$ is a PMC.

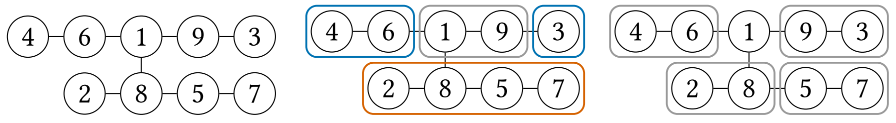
**Potential Maximal Cliques.** For every PMC $\Omega$ that is discovered this way or at a later step of the algorithm the same procedure is applied. If all components in support$(\Omega)$ are $k$-feasible (this includes the case where support$(\Omega) = \emptyset$), then $\Omega$ is registered as a $k$-feasible PMC.

**Feasible PMCs.** For every newly found $k$-feasible PMC $\Omega$, if $\operatorname{outlet}(\Omega) = \emptyset$ then the algorithm has found a witness for $\operatorname{tw}(G) \leq k$. In this case, the algorithm could return immediately, but in accordance with Tamaki's description of the algorithm, we assume that it returns only after listing all structures. Otherwise, $\operatorname{crib}(\operatorname{outlet}(\Omega), \Omega)$ is a potentially newly discovered $k$-feasible I-block.

**Feasible I-blocks.** For every newly found $k$-feasible I-block $(N(C), C)$ the following procedure is followed. First, for any already discovered O-block $(N(B), B)$ with $C \subseteq B$ the vertex set $K := N(C) \cup N(B)$ is constructed. If $|K| \leq k + 1$ and $K$ is a PMC, a new PMC is discovered. This corresponds to case 2 in the definition of $k$-buildable PMCs. If $|K| \leq k$ and there is a unique full component $A$ associated with $K$, then a new O-block $(N(A), A)$ is discovered. Next, the outbound full component $A$ associated with $N(C)$ is the component part of a potentially new O-block $(N(C), A)$. Subsequently for each O-block $(N(A), A)$ that had already been discovered before processing the current I-block and for each $v \in N(A)$, all PMCs $K$ with $|K| \leq k + 1$ of the form $K = N(A) \cup (N(v) \cap A)$ are registered. This step corresponds to case 3 of $k$-buildable PMCs. All newly discovered PMCs are processed according to the procedure above, before starting to process the next I-block in the collection of I-blocks.

If at some point all I-blocks have been exhaustingly processed, but no $k$-feasible PMC with empty outlet was found, the algorithm returns with negative result. Note that sometimes the processing of an I-block leads to the re-discovery of an already found PMC. In this case, the I-block could be the last missing $k$-feasible component in the support, which means that the PMC will be recognized as $k$-feasible.

The correctness of the algorithm is shown by inductively showing that all $k$-feasible I-blocks, $k$-feasible O-blocks, $k$-buildable PMCs and $k$-feasible PMCs of any size are discovered [Tam19, Theorem 1]. Let $\mathcal{I}_G^k$ and $\mathcal{O}_G^k$ denote the number of $k$-feasible I-/O-blocks for the given $k$ and let $\mathcal{P}_G^k$ be the number of $k$-buildable potential maximal cliques. As with similar notations we omit the subscript $G$ if the graph is clear from the context. Then, the running time of the algorithm on a graph $G$ is in $O^*(\mathcal{I}^k \cdot \mathcal{O}^k)$, where $O^*$ hides asymptotically polynomial factors. As all of these mentioned structures are listed exhaustingly, the algorithm further runs in $\Omega(\mathcal{I}^k + \mathcal{O}^k + \mathcal{P}^k)$ time.



**(a)** Example graph. **(b)** A PMC (gray) with inbound (blue) and outbound (orange) components. **(c)** The four case 1 1-buildable PMCs (gray).

**Figure 3.2:** Example graph with annotated structures.

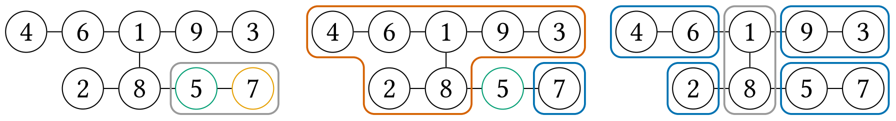
## 3.4 Example for PID-BT

Following the description of PID-BT we now discuss a brief example in order to strengthen our intuition of the algorithm. Insights from the example are then used as the foundation of formalized proofs of upper and lower running time bounds on elementary graph classes in Section 3.5.

We want to explore the behaviour of the algorithm on the graph depicted in Figure 3.2 (a) with $k = 1$. The vertex ordering is determined by the numbers inside the vertices.

We start with a few examples for PMCs, I-blocks and O-blocks. The set $\{1, 9\}$ is a PMC with three associated non-full components $\{4, 6\}$, $\{3\}$, $\{2, 8, 5, 7\}$, see Figure 3.2 (b). The components $\{4, 6\}$ and $\{3\}$ are inbound, because their neighbourhoods have an associated full component that contains a smaller vertex. The set $\{2, 8, 5, 7\}$ is a whole outbound component and $\{1, 2, 8, 5, 7\}$ is an example for a split outbound component. This means that $\{1\}$ is the outlet of the PMC $\{1, 9\}$ and $\{3\}$ is the only component in its support. If the PMC is 1-feasible, then there is a valid partial solution for $\operatorname{tw}(G) \leq 1$ of $\{1, 9, 3\}$ that has $\{1, 9\}$ as interface.

In the following, we write $X_p$ to signify that the vertex set $X$ is a PMC, and $X_i/X_o$ for a full I-block / O-block $(N(X), X)$. In the example graph there are four vertices whose closed neighbourhood has size only 2. These neighbourhoods are the case 1 1-buildable PMCs $\{2, 8\}_p$, $\{9, 3\}_p$, $\{4, 6\}_p$, $\{5, 7\}_p$ (see Figure 3.2 (c)) the algorithms identifies at the start. These PMCs consist of a leaf vertex and its neighbour, that is adjacent to an associated outbound component containing the vertex 1. This means that for each of these PMCs, the outlet consists of the non-leaf vertex. All of these PMCs have empty support as they either have no associated inbound components, or in the case of $\{2, 8\}_p$ the inbound



**(a)** A PMC (gray) with outlet (green) and crib of the outlet w.r.t. the PMC (yellow).

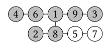
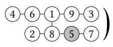
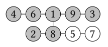
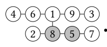
**(b)** Inbound (blue) and outbound (orange) component assoc. with outlet (green) of PMC $\{5,7\}_p$.

**(c)** PMC (gray) with empty outlet and assoc. inbound (blue) components.

**Figure 3.3:** More structures on the example graph.

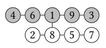
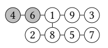
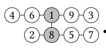
component $\{5,7\}_i$ is adjacent only to the outlet. Thus, the initial case 1 PMCs are all recognised as 1-feasible. Next, for each of the PMCs, the crib of the outlet with respect to the entire PMC is identified as the component part of a 1-feasible I-block. In all four cases, this is the degree 1 vertex of the PMC, as depicted in Figure 3.3 (a) for $\{5,7\}_p$ where $\text{crib}(\{5\},\{5,7\}) = \{7\}$.

We now follow the events related to the I-block $\{7\}_i$. During its processing, the 1-feasible full outbound component $\{4,6,1,9,3,2,8\}$ associated with $\{5\}$ is found (see Figure 3.3 (b)) and subsequently the case 3 1-buildable PMC $\{5,8\}_p$ is found by evaluating $N(\{4,6,1,9,3,2,8\}) \cup (N(5) \cap \{4,6,1,9,3,2,8\}) = \{5,8\}$:

$N($  $) \cup (N($  $) \cap$  $) =$  .

The outlet of $\{5,8\}_p$ is $\{8\}$ and $\{7\}$ is the only component in its support, making $\{5,8\}_p$ 1-feasible. By building $\text{crib}(\text{outlet}(\{5,8\}),\{5,8\})$, this leads to the discovery of the 1-feasible I-block $\{5,7\}_i$. When processing $\{5,7\}_i$, the algorithm discovers the next 1-feasible O-block $\{1,3,4,6,9\}_o$ and by using another case 3 construction for buildable PMCs, the PMC $\{1,8\}_p$, see Figure 3.3 (c). This PMC only has associated inbound components and thus an empty outlet. As soon as all of its inbound components $\{4,6\}$, $\{3,9\}$, $\{2\}$, and $\{5,7\}$ have been recognized as 1-feasible, $\{1,8\}_p$ is found to be 1-feasible, too. This way, the algorithm finds that the treewidth of the graph is indeed 1.

Note that this is just one example for how the algorithm can find $\{1,8\}_p$. Depending on the order in which the PMCs and I-blocks are found and processed, $\{1,8\}_p$ can also be constructed as a case 2 1-buildable PMC, by combining $\{4,6,1,9,3\}_o$ and $\{4,6\}_i$ as $\{1,8\}_p = N(\{4,6,1,9,3\}) \cup N(\{4,6\})$:

$N($  $) \cup N($  $) =$  .

This example already gives us a good intuition for the behaviour of the algorithm and the meaning of the various defined structures and their interplay. In the next section, we deepen this understanding by deriving bounds for the number of PMCs, I-, and O-blocks in graphs of different graph classes.

## 3.5 Running Time Bounds for PID-BT

Just as previously we use $\mathcal{I}^k$ and $\mathcal{O}^k$ for the number of $k$-feasible I-/O-blocks in a graph for a given $k$ and $\mathcal{P}^k$ for the number of $k$-buildable potential maximal cliques. Subscripts are added to this notation in case the graph that is referred to is not clear from the context.

In the following, we establish inequalities between the number of $k$-feasible I- and O-blocks and $k$-buildable PMCs and discuss their limitations. Subsequently, we derive upper and lower bounds for I- and O-blocks on basic graph classes for different values of $k$.

▶ **Lemma 3.8.** Let $G$ be a graph. Then the following inequalities hold.

1. $\mathcal{I}^k \leq n \cdot \mathcal{O}^k$
2. $\mathcal{I}^k \leq \mathcal{P}^k$
3. The number of $k$-feasible whole O-blocks is at most $n \cdot \mathcal{P}^k$
4. (a) $\mathcal{P}^k \leq (n+1)^{k+1}$, (b) $\mathcal{I}^k \leq (n+1)^{k+1}$, and (c) $\mathcal{O}^k \leq (n+1)^k$

◀

*Proof.* 1. The neighbourhood of an inbound component is a minimal separator. There can be at most $n$ (or actually $n-1$) full components associated with a minimal separator. The statement follows, as exactly one of these full components is outbound, while all others are inbound.

2. Follows directly from Lemma 3.7.

3. For a $k$-feasible whole O-block $(N(A), A)$ there exists a $k$-feasible I-block $(N(A), C)$ and by Lemma 3.7 there exists a PMC with outlet $N(A)$. For two outbound components $A_1$ and $A_2$ associated with a PMC, either $N(A_1) \subseteq N(A_2)$ or $N(A_2) \subseteq N(A_1)$ holds [Tam19, Lemma 4]. Further, $N(A_1) \neq$

$N(A_2)$ holds, because otherwise either $A_1$ or $A_2$ would be inbound. Thus, the number of outbound components per PMC is at most the size of the outlet which is at most $n$.

4. A potential maximal clique is a subset of $V$ of size up to $k+1$. There are no more than $(n+1)^{k+1}$ such subsets (a). The next inequality (b) follows directly, as $\mathcal{I}_G^k \leq \mathcal{P}_G^k$. O-blocks are uniquely identified by their neighbourhood, as no two outbound components can have the same neighbourhood. The neighbourhood of a $k$-feasible O-block consists of up to $k$ vertices, which gives the claimed inequality (c).

■

It would be useful to also find upper bounds for the number of $k$-feasible O-blocks in terms of the number of $k$-feasible PMCs. For whole $k$-feasible O-blocks, this is straightforward, as the neighbourhood of a full inbound component is also the neighbourhood of a unique full outbound component. For the number of $k$-feasible split O-blocks however, there does not seem to be such an easy upper bound in terms of the number of $k$-feasible I-blocks or PMCs. This allows the construction of graphs with $\Omega(n)^k$ $k$-feasible split O-blocks and linear number of $k$-feasible PMCs. Before giving an example of this and showing how it can be transformed to a lower bound for the running time of PID-BT on hyperbolic random graphs, we first analyse the number of $k$-feasible I-blocks and / or $k$-buildable PMCs on a few more elementary graph classes.

► **Proposition 3.9.** Let $T$ be a tree with $n$ vertices. For any $k > 0$, the number of $k$-feasible I-blocks in $T$ is in $\Theta(n)$. ◄

*Proof.* As $\mathcal{I}^k \leq \mathcal{P}^k$ it suffices to show an upper bound on the number of PMCs. A vertex set is a PMC, if there is no full component associated with it and it is cliquish. This means that any PMC must consist of two adjacent vertices because both a single vertex and a pair of non-adjacent vertices have an associated full component (unless $n = 1$) and more than two vertices are not cliquish. Thus, in a tree, we have $\mathcal{I}^k \in O(n)$ for any $k$.

We also have $\mathcal{I}^k \in \Omega(n)$ because for every edge that is not incident to the minimum vertex $v_{min}$, one of the two connected components separated by the edge is a $k$-feasible inbound component. ■

A similar result can be obtained for circles.

▶ **Proposition 3.10.** Let $C$ be a circle graph with $n$ vertices. For any $k \geq 2$, the number of $k$-feasible I-blocks in $C$ is in $\Theta(n^2)$ and the number of PMCs in $C$ is in $\Theta(n^3)$. ◀

*Proof.* Let $v_{min}$ be the minimum vertex in the total ordering of $V(C)$. Then every connected vertex set $D$ such that $v_{min}$ is not element of $N[D]$ is a $k$-feasible I-block. As there are only $\Theta(n^2)$ connected vertex sets in $C$ and inbound components are connected vertex sets, this means that $\mathcal{I}^k \in \Theta(n^2)$.

Regarding PMCs, first, we show that any PMC has 3 vertices. To see this, note that any set of at most 2 vertices has an associated full component. Further, any set with 4 or more vertices is not cliquish, because it contains a pair of vertices that are neither directly adjacent nor adjacent to the same component.

Next, consider a PMC $\Omega$ with three vertices of which none is $v_{min}$ or a neighbour of $v_{min}$. There are either one, two, or three components associated with $\Omega$, of which one contains $v_{min}$ and is thus outbound. If present, the other components are inbound and in the support of $\Omega$. The inbound components further are $k$-feasible, which means that $\Omega$ is $k$-feasible, too. As there are $(n-3)\cdot(n-4)\cdot(n-5)$ variants of such an $\Omega$, we conclude $P^k \in \Theta(n^3)$. ■

Interestingly while the number of PMCs is higher in circles than in trees, there are no split O-blocks in circles, and thus the number of $k$-feasible O-blocks in a circle is in $\Theta(n^2)$. Next, we look at slightly more complex graph classes. We call graph $G$ a *cactus graph*, if any two simple circles in $G$ share at most one vertex.

▶ **Proposition 3.11.** Let $G$ be a cactus graph. Then for any $k \geq 2$, the number of PMCs in $G$ is in $O(n^3)$ and the number of $k$-feasible I-blocks is in $O(n^2)$. ◀

*Proof.* First, we assume that $G$ does not contain degree 1 vertices and establish that the vertices of any PMC $\Omega$ lie on a single circle $C$ in $G$. Suppose otherwise that $\Omega$ is a PMC that lies on two circles but not on one. If $\Omega$ contains a cut vertex that belongs to both circles or separates the circles, then $\Omega$ is not cliquish as there would be a pair of non-adjacent vertices without common adjacent component. Otherwise, as $\Omega$ can not contain more than 2 vertices per circle, there is a full component associated with $\Omega$. Adding more vertices to $\Omega$ cannot resolve this issue, and thus we conclude that any PMC in $G$ lies on a single circle. Thus, by the same argumentation as in Proposition 3.10, the number of PMCs is upper bounded by $O(n^3)$.

An inbound component $D$ cannot be a separator, thus the part of $D$ that is adjacent to $N(D)$ must lie in a single circle. In that circle $D$ has (up to) two endpoints. This means that per circle $C$ there are no more than $O(|C|^2)$ inbound components, resulting in a total of at most $O(n^2)$ $k$-feasible I-blocks.

Both bounds also hold if we allow degree 1 vertices in $G$, as appending a tree only adds a linear amount of PMCs and I-blocks. ■

The invariance of the number of PMCs and I-blocks to higher values of $k$ does however not carry over to all graphs of larger treewidth. Consider for example a graph that is a rectangular grid with $a \times b$ vertices, which has treewidth $k = \min(a, b)$. On such a graph there are $2^{\Omega(k)}$ paths between certain vertices with distance linear in $k$ and all of these paths form $k$-feasible inbound components. Even more interestingly, such a lower bound can also be found on graphs of constant treewidth such as series parallel graphs, as shown in the next proposition.

▶ **Proposition 3.12.** For any $k \geq 2$ there is a series parallel graph $G$, such that $G$ contains $2^{\Omega(k)}$ $k$-buildable PMCs and $k$-feasible I-blocks. ◀

*Proof.* We construct $G$ by parallel composition of $k$ paths with 4 vertices. Let $s$ and $t$ be the vertices at which the paths have been joined and define $t$ as the minimum vertex in the total order of vertices.

Then any vertex set with $s$ and any combination of neighbours of $s$ forms an inbound component. Such a component is also $k$-feasible, because when searching for a tree decomposition of width $k$ even traversing all $k$ paths in parallel is valid. As each of these inbound components has at most $k$ neighbours, the described components and their neighbourhoods form $k$-feasible I-blocks. This yields a lower bound of $2^k$ on the number of $k$-feasible I-blocks in $G$.

Similarly, we find $2^{k-1}$ $k$-feasible PMCs by choosing the two vertices between $s$ and $t$ on one of the paths and one of these two vertices on all other paths. It is easy to see that such a vertex set has no full associated component and is cliquish. Further, the associated component containing $s$ is $k$-feasible and inbound, as established above. As for $k-1$ paths either the vertex adjacent with $s$ or the one adjacent with $t$ is chosen, this gives us $2^{k-1}$ $k$-feasible PMCs. ■

Instances that have an exponential number of O-blocks are even easier to construct. In the following example we show that even trees can have a number of O-blocks exponential in $k$.

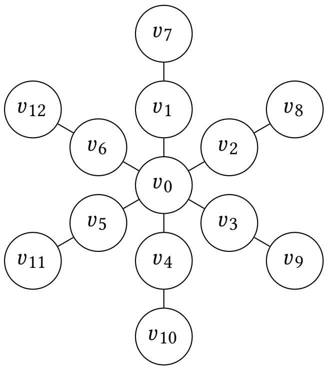


**Figure 3.4:** Acyclic graph $G^*_\ell$ with exponential number of O-blocks for $\ell = 6$.

▶ **Proposition 3.13.** For every $\ell \in \mathbb{N}$ there is a tree $G^*_\ell$ with $2\ell + 1$ vertices and contains $\Omega(\ell^k)$ $k$-feasible O-blocks and $O(\ell)$ $k$-feasible I-blocks for $k$ up to $\ell - 2$. ◀

*Proof.* We construct $G^*_\ell$ from a star with $v_0$ as central vertex and $v_1, \ldots, v_\ell$ as leaves by appending a vertex $v_{i+\ell}$ to $v_i$ for each $i \in 1, \ldots, \ell$, as depicted in Figure 3.4 for $\ell = 6$. We assume the total vertex ordering implied by the vertices' indices. For simplicity we call the length-2 paths connected to $v_0$ *arms*.

An inbound component $C$ cannot span multiple arms, because then it would have to contain $v_0$ which means that there cannot be another component $D$ with the same neighbourhood that precedes $C$. This means that all inbound components consist either of a leaf vertex or a complete arm (except the one containing $v_1$). Thus, there are only $O(\ell)$ $k$-feasible I-blocks in $G^*_\ell$.

In contrast to that, outbound components can span multiple arms. In fact, any union of $v_0$ and the vertices of up to $\ell - 2$ arms forms an outbound component, because there is no other full component associated with its neighbourhood. The neighbourhood of such an outbound component consists of degree-2 vertices that are the neighbourhoods of $k$-feasible inbound components. This gives us $\Omega(\ell^k)$ $k$-feasible O-blocks for $k$ up to $\ell - 2$. ■

Note that this example can be easily generalized to graphs of larger treewidth by replacing each $v_i$ by a bigger structure of connected vertices. In particular replacing, e.g. $v_{2\ell}$ by a $k$-clique increases the treewidth to $k - 1$. This serves as another example for an almost trivial instance on which PID-BT already has exponential running time in $k$.

We can express the generalized structure of the graphs $G^*_\ell$ in a lemma that we can then use to show a lower bound for the number of $k$-feasible O-blocks in hyperbolic random graphs. Consider the following generalisation of $G^*_\ell$. Let $G$ be a graph with some vertex ordering, $C$ be a connected vertex set containing the minimum vertex, and $C_1, \dots, C_\ell$ be the connected components associated with $C$. Further for $1 \le i \le \ell$ assume $C_i \setminus N(C) \neq \emptyset$ and let $S_i = N(C) \cap C_i$. In resemblance to the structure of $G^*_\ell$, we call the tuple $(C, (C_1, \dots, C_\ell), (S_1, \dots, S_\ell))$ a *star decomposition* $G$ with *centre* $C$, *arms* $C_1, \dots, C_\ell$ and *joints* $S_1, \dots, S_\ell$. As an example, consider the canonical star decomposition of $G^*_\ell$, where $\{v_0\}$ is the centre, all components associated with $\{v_0\}$ are arms and the joint of each arm consists of the degree-two vertex connected with $v_0$. A star decomposition is *$\Delta$-thin* if each joint $S_i$ has size at most $\Delta$. We can now formulate the lemma that generalizes the idea from Proposition 3.13 to graphs with suitable star decompositions.

▶ **Lemma 3.14.** Let $G$ be a graph and $(C, (C_1, \dots, C_\ell), (S_1, \dots, S_\ell))$ be a $\Delta$-thin star decomposition of $G$ for a constant $\Delta$. Suppose that for each arm $C_i$ the block $(S_i, C_i \setminus S_i)$ is a $k$-feasible I-block for the target value $k$ for the treewidth. Then $G$ contains at least $\Omega\big(\ell^{f(k)}\big)$ $k$-feasible O-blocks for some $f(k) \in \Omega(k)$. ◀

*Proof.* We consider the union $C$ of some arms and define the vertex set $\overline{C}$ as $V(G) \setminus C$. Assuming $N(\overline{C}) \le k$, we first show that $(N(\overline{C}), \overline{C})$ is a $k$-feasible O-block and then show that there are enough choices for such an $\overline{C}$ to prove the claimed bound. First note that $(N(\overline{C}), \overline{C})$ is an O-block because $\overline{C}$ is connected and its neighbourhood is distributed across multiple of its associated components and thus there exists no other full component associated with $N(\overline{C})$. We have $N(\overline{C}) \le k$ by assumption and also the neighbourhood of $\overline{C}$ is the union of some joints $S_i$. By assumption these $S_i$ are the neighbourhoods of $k$-feasible inbound components and hence $\overline{C}$ is a $k$-feasible O-block.

It remains to show that there are many such outbound components. If $C$ has $x$ arms, then its complement $\overline{C}$ has at most $\Delta \cdot x$ neighbours. Thus, there is an $f(k) \in \Omega(k)$ such that the complement of a union of $f(k)$ arms has a neighbourhood of size at most $k$. This means that there are at least $\binom{\ell}{f(k)} \in \Omega\big(\ell^{f(k)}\big)$ choices of $\overline{C}$. ■

In the following, we show that we can find a star decomposition suitable for the above lemma in hyperbolic random graphs, allowing a lower bound for the expected running time of PID-BT on HRGs to be derived.

The rough idea is that a HRG contains many vertices of constant degree that are connected to the giant component. We can also find a subset of these vertices that has disjoint neighbourhoods. To construct a star decomposition, we use the closed neighbourhoods of these vertices as arms and the remainder of the giant component as the centre.

Let $v \in \mathcal{G}$ stand for the event in which vertex $v$ is in the giant component of the HRG.

▶ **Lemma 3.15.** Let $v$ be a vertex in a hyperbolic random graph. There are constants $c_1$ and $c_r$ such that $\Pr[v \in \mathcal{G} \mid r(v) \geq R - c_r] \geq c_1$. ◀

*Proof.* There is a constant $c > 0$ such that with probability $1 - o(1)$ the largest component of a HRG contains at least $c \cdot n$ vertices [BFM13, Theorem 1.1]. Further, for a constant $c_r > 0$ the expected number of vertices with radius at most $R - c_r$ is a constant fraction of $n$. We can thus choose $c_r$ so that with constant probability only $\frac{n \cdot c_1}{2}$ vertices have radius smaller than $R - c_r$.

As we only want to derive a constant lower bound for $\Pr[v \in \mathcal{G} \mid r(v) \geq R - c_r]$ and the above events happen with probability $1 - o(1)$ and $\Theta(1)$, it suffices to show a constant lower bound for $v \in \mathcal{G}$ under the additional condition that the giant component contains $c \cdot n$ vertices and that at most $\frac{n \cdot c_1}{2}$ vertices have radius smaller than $R - c_r$.

Now, the remainder of the giant component has radius at least $R - c_r$. These are at least $\frac{n \cdot c_1}{2}$ vertices, which is a constant fraction of all vertices and therefore also at least a constant fraction of the vertices with radius at least $R - c_r$. We can choose $c_1 = \frac{c}{2}$ to derive the claimed bound. ■

Let $\mathcal{E}_r = r(v) \geq R - c_r$ be the event in which vertex $v$ has radius at least $R - c_r$. We now show statements that give us arbitrarily small probabilities for $v$ having large degree and for $v$ having a large sum of degrees of its neighbours.

▶ **Lemma 3.16.** Let $v$ be a vertex in a hyperbolic random graph that has radius at least $R - r_c$ for a constant $r_c$. Then there is a constant $\Delta_{R-c_r}$ such that for any $d > 0$ we have $\Pr[\deg(v) > d \mid \mathcal{E}_r] \leq \frac{\Delta_{R-c_r}}{d}$. ◀

*Proof.* The expected degree of a vertex grows with decreasing radius, so the worst case occurs if $v$ has radius $R - c_r$. Via Equation (2.3) we can calculate the expected degree as $\Delta_{R-c_r} = n \cdot \mu(B_O(R) \cap B_{R-c_r}(R)) \in \Theta(1)$. Thus, Markov's inequality gives us the claimed bound for any $d > 0$. ■

▶ **Lemma 3.17.** Let $v$ be a vertex in a hyperbolic random graph that has radius at least $R - c_r$ for a constant $c_r$. Then there is a function $f_\Sigma(n) \in O\big(n^{2-2\alpha}\big)$ such that for any $d > 0$ we have $\Pr\big[\sum_{u \in N(v)} \deg(u) > d \,\big|\, \mathcal{E}_r\big] \le \frac{f_\Sigma(n)}{d}$. ◀

*Proof.* The degrees of the neighbours of a vertex with radius $r$ sum to $O(ne^{-(\alpha-1/2)r})$ in expectation [Blä+18, Section 3.2.2], so the worst case occurs at $r(v) = R - r_c$. So for some $f_\Sigma(n) \in O(ne^{-(\alpha-1/2)(R-r_c)})$ we have $\mathrm{E}\big[\sum_{u \in N(v)} \deg(u) \,\big|\, \mathcal{E}_r\big] \le f_\Sigma(n)$. We get

$$\begin{aligned} f_\Sigma(n) &\in O\Big(ne^{-(\alpha-1/2)(R-r_c)}\Big) \\ &= O\Big(ne^{-(\alpha-1/2)(2\log n + C - r_c)}\Big) \\ &= O\big(n^{2-2\alpha}\big) \end{aligned}$$

and the claimed bound then follows via Markov's inequality. ■

▶ **Theorem 3.18.** Let $G$ be a hyperbolic random graph. In expectation, there are $\Omega(n^{2\alpha-1})^{\Omega(n^{1-\alpha})}$ $k$-feasible O-blocks in $G$ for $k = \mathrm{tw}(G)$. ◀

*Proof.* We want to show that the giant component of $G$ contains a thin star decomposition as required for Lemma 3.14. The basic idea is that we select vertices close to the boundary of the disk that are part of the giant component, have few neighbours and for which the sum of degrees of their neighbours is not too large. We can then find many non-overlapping closed neighbourhoods of these vertices and use these as the arms of a star decomposition to derive the claimed lower bound.

From Lemma 3.15, there are constants $c_r$ and $c_1$ such that

$$\Pr[v \in \mathcal{G} \mid r(v) \ge R - c_r] \ge c_1.$$

Now let $\mathcal{E}_1(\Delta) = \deg(v) \le \Delta$ be the event that $v$ has degree at most $\Delta$ and $\mathcal{E}_2(\Delta) = \sum_{u \in N(v)} \deg(u) \le \Delta$ be the event that the sum of degrees of the neighbours of $v$ is at most $\Delta$. For suitable constants $\Delta_1, \Delta_2$ we have

$$\Pr[\mathcal{E}_1(\Delta_1) \wedge \mathcal{E}_2(\Delta_2) \wedge v \in \mathcal{G} \mid \mathcal{E}_r]$$

$$= 1 - \Pr[\neg\mathcal{E}_1(\Delta_1) \vee \neg\mathcal{E}_2(\Delta_2) \vee v \notin \mathcal{G} \mid \mathcal{E}_r]$$

and via a union-bound we derive

$$\begin{aligned}
&\geq 1 - \Pr[\neg\mathcal{E}_1(\Delta_1) \mid \mathcal{E}_r] - \Pr[\neg\mathcal{E}_2(\Delta_2) \mid \mathcal{E}_r] - \Pr[v \notin \mathcal{G} \mid \mathcal{E}_r] \\
&= 1 - \Pr[\neg\mathcal{E}_1(\Delta_1) \mid \mathcal{E}_r] - \Pr[\neg\mathcal{E}_2(\Delta_2) \mid \mathcal{E}_r] - 1 + \Pr[v \in \mathcal{G} \mid \mathcal{E}_r] \\
&= \Pr[v \in \mathcal{G} \mid \mathcal{E}_r] - \Pr[\neg\mathcal{E}_1(\Delta_1) \mid \mathcal{E}_r] - \Pr[\neg\mathcal{E}_2(\Delta_2) \mid \mathcal{E}_r] \\
&\geq c_1 - \Pr[\neg\mathcal{E}_1(\Delta_1) \mid \mathcal{E}_r] - \Pr[\neg\mathcal{E}_2(\Delta_2) \mid \mathcal{E}_r]. \qquad (3.1)
\end{aligned}$$

This means that we need to upper bound the probability for $\neg\mathcal{E}_1(\Delta_1)$ and $\neg\mathcal{E}_2(\Delta_2)$ by small constants, which we can do by choosing $\Delta_1$ and $\Delta_2(n)$ appropriately.

By Lemma 3.16 there exists a constant $\Delta_{R-c_r}$ such that

$$\Pr[\deg(v) > d \mid \mathcal{E}_r] \leq \frac{\Delta_{R-c_r}}{d}$$

for $d > 0$. This allows is to choose $\Delta_1$ sufficiently large such that

$$\Pr[\neg\mathcal{E}_1(\Delta_1) \mid \mathcal{E}_r] \leq \frac{c_1}{4}. \qquad (3.2)$$

Similarly, by Lemma 3.17 there is a $f_\Sigma \in O\big(n^{2-2\alpha}\big)$ such that

$$\Pr\left[\sum_{u \in N(v)} \deg(u) > d \,\middle|\, \mathcal{E}_r\right] \leq \frac{f_\Sigma(n)}{d}$$

for $d > 0$. This allows is to choose $\Delta_2$ as a function in $O\big(n^{2-2\alpha}\big)$ with sufficiently large constants such that

$$\Pr[\neg\mathcal{E}_2(\Delta_2) \mid \mathcal{E}_r] \leq \frac{f_\Sigma(n)}{\Delta_2(n)} \leq \frac{c_1}{4}. \qquad (3.3)$$

So, putting Equation (3.2) and Equation (3.3) into Equation (3.1), we get

$$\Pr\left[\deg(v) \leq \Delta_1 \wedge \sum_{u \in N(v)} \deg(u) \leq \Delta_2(n) \wedge v \in \mathcal{G} \,\middle|\, \mathcal{E}_r\right] \geq \frac{c_1}{2}.$$

As $\Pr[\mathcal{E}_r] = \Pr[r(v) \geq R - c_r] = 1 - \mu(B_O(R - c_r)) = 1 - e^{-\alpha c_r} \cdot (1 - o(1)) \geq c_2$ for some constant $c_2$, we get

$$\Pr\left[\deg(v) \leq \Delta_1 \wedge \sum_{u \in N(v)} \deg(u) \leq \Delta_2(n) \wedge v \in \mathcal{G} \wedge r(v) \geq R - c_r\right] \geq \frac{c_1 \cdot c_2}{2}.$$

So the expected number of vertices that have the above properties is at least $n \cdot \frac{c_1 \cdot c_2}{2} \in \Theta(n)$. Let $U$ be this set of vertices. We now show that we can select $\Theta(n^{2\alpha-1})$ vertices of $U$ such that they have disjoint neighbourhoods. The sum of the degrees of a vertex $v \in U$ is at most $O\left(n^{2-2\alpha}\right)$. So, in the worst-case, every vertex in the neighbourhood of $v$ only connects to other vertices in $U$. This means that by selecting $v$ we obstruct at most $O\left(n^{2-2\alpha}\right)$ other vertices in $U$ whose neighbourhood intersects $N(v)$. Thus, the expected number of vertices with degree at most $\Delta_1$ and radius at least $R - c_r$ and that belong to the giant component and that have disjoint neighbourhoods is at least

$$\Omega\left(\frac{n \cdot c_1 \cdot c_2}{4n^{2-2\alpha}}\right) = \Omega\left(n^{1-(2-2\alpha)}\right) = \Omega\left(n^{2\alpha-1}\right).$$

We can now verify that the star decomposition that uses the closed neighbourhoods of these vertices as arms and the remainder of the giant component as the centre has the desired properties. Suppose that the minimum vertex of the ordering of $V(G)$ is in the centre of the star decomposition. First of all, this construction indeed yields a star decomposition, because the selected vertices are part of the giant component and each closed neighbourhood is a component associated with the centre, because the neighbourhoods are disjoint. Next, the star decomposition is $\Delta_1$-thin because the selected vertices have constant degree at most $\Delta_1$. Then, for each selected vertex $v$ the block $(N(v), \{v\})$ is a $k$-feasible I-block, because $|N[v]| \leq \Delta_1 + 1$, while the treewidth $k$ of $G$ is in $\Theta(n^{1-\alpha})$ [BFK16, Theorem 9].

If the minimum vertex of the giant component is not in the centre of the star decomposition, we can add its closed neighbourhood to the centre without changing the asymptotic number of arms. So, in expectation we can select $\Omega\left(n^{2\alpha-1}\right)$ vertices with disjoint neighbourhoods as the arms of a $\Delta_1$-thin star decomposition, which lets us apply Lemma 3.14 to derive the claimed bound. ■

▶ **Corollary 3.19.** The expected running time of PID-BT on a hyperbolic random graph is in $2^{\Omega\left(n^{1-\alpha}\right)} \cdot \mathrm{poly}(n)$. ◀

*Proof.* Follows from Theorem 3.18 via a simple transformation

$$\begin{aligned}\Omega(n^{2\alpha-1})^{\Omega(n^{1-\alpha})} &= 2^{\Omega\left(n^{1-\alpha}\right)\cdot\log\left(\Omega(n^{2\alpha-1})\right)}\\ &= 2^{\Omega\left(n^{1-\alpha}\right)\cdot(2\alpha-1)\log(\Omega(n))}\\ &= 2^{\Omega\left(n^{1-\alpha}\right)} \cdot \mathrm{poly}(n)\end{aligned}$$

■

This affirms the experimental results in which PID-BT did not perform well without the preprocessing procedure.

# 4 Greedy Preprocessing

In addition to the PID-BT algorithm itself, Tamaki's submission for the PACE'17 challenge also performs a preprocessing step. This section explains the algorithms and heuristics used in the preprocessing and partially explains their effectiveness both empirically and theoretically.

The idea of the preprocessing is based on so-called *safe* separators which were introduced by Bodlaender and Koster [BK06]. A separator $S$ in $G$ is called safe for treewidth or simply safe if completing $S$ into a clique does not increase the treewidth of $G$. Safe separators are useful because they can be used to split $G$ into smaller sub-graphs $G[C \cup S] + \text{clique}(S)$ for each component $C$ associated with $S$. For a safe $S$, the treewidth of $G$ then is the maximum treewidth of $G[C \cup S] + \text{clique}(S)$ over all components $C$ associated with $S$. As tree decompositions of these sub-graphs have a bag containing $S$, an optimal tree decomposition of $G$ can be obtained by combining optimal solutions of the sub-graphs using a new bag $S$ to join the tree decompositions.

In consequence, the goal of the preprocessing step is to identify many safe separators so that the original problem is split into smaller and at best also easier sub-problems. Of course, this is no easy task, as the NP-hardness of deciding the treewidth of a graph directly transfers to the problem of deciding whether a separator is safe. For example, a treewidth instance $(G, k)$ can be reduced to a safe separator instance by appending a path of $k + 2$ vertices to $G$ and asking if the first $k + 1$ vertices of the path are a safe separator.

This means that in most practical settings, it is reasonable to rely on heuristics in order to identify safe separators. The preprocessing procedure proposed by Tamaki does this in both of its two steps. In the first step, multiple greedy heuristics are used to compute (possibly non-optimal) tree decompositions of the input graph. As the intersection of adjacent bags in a tree decomposition is a separator of the graph, the greedy tree decompositions are then used as sources of potentially safe separators. In the second step, the separators from the greedy tree decompositions are heuristically checked for safeness.

We now first explain how the preprocessing utilises the found safe separators in order to simplify the problem. Afterwards, we describe the greedy heuristics used to compute tree decompositions and different sufficient conditions that heuristically identify the safeness of separators.

As explained above, safe separators are used to split a given instance into smaller sub-instances. In addition to that, the safe separators can also be used to derive lower bounds for the treewidth of the graph. If for example $S$ is a safe separator in $G$, then $\mathrm{tw}(G) = \mathrm{tw}(G + \mathrm{clique}(S)) \geq |S|$. The highest lower bound found this way is then used to identify sub-instances that are already optimally solved by one of the greedy tree decompositions. To see how this works note that a tree decomposition of $G$ can be converted into an equally or less wide tree decomposition of a sub-graph $G'$ of $G$ by intersecting each bag with $V(G')$. This way, the preprocessing can already eliminate all safely separated sub-graphs for which the width of a converted greedy tree decomposition is below the lower bound. The PID-BT algorithm is then only called on the remaining safely separated sub-graphs.

## 4.1 Greedy Tree Decompositions

The basic idea for how the greedy heuristics compute tree decompositions is for example described by Bodlaender [Bod05]. The approach is closely related to an algorithm that can find the treewidth or a tree decomposition of chordal graphs in polynomial time. This is done via a perfect elimination ordering, that is an ordering $v_1, \ldots, v_n$ of the vertices such that $v_i$ is simplicial in $G[v_{i+1}, \ldots, v_n]$ for $i \in 1, \ldots, n-1$. Given such a perfect elimination ordering, the treewidth of $G = G_0$ is equal to the maximum degree of any $v_i$ in $G_i = G[v_{i+1}, \ldots, v_n]$ over $i \in 1, \ldots, n-1$. Even though non-chordal graphs do not have perfect elimination orderings, the procedure can be adapted to work with any ordering. That way, the solution becomes an upper bound for the treewidth and matches the exact solution, if the given ordering is a perfect elimination ordering. Just as described for chordal graphs, the adapted algorithm still iterates through the ordering and considers the degree of the current vertex $v_i$ in the remainder of the graph $G_i$. The difference is that in order to handle non simplicial vertices, $G_i$ is constructed from $G_{i-1}$ by not only removing $v_{i-1}$, but by additionally completing $N(v_{i-1})$ into a clique. See Algorithm 1 for a description of how this approach can be used to recursively compute an explicit tree decomposition.

Interestingly, this way of computing the treewidth / a tree-decomposition

**Algorithm 1:** Recursive computation of a tree decomposition of $G$

```
1 if $G$ consists of a single clique then
2     return tree decomposition with single bag $V(G)$;
3 select vertex $v \in V(G)$ according to an ordering (or some heuristic);
4 let $G' \leftarrow G[V(G) \setminus \{v\}] + \text{clique}(N(V))$;
5 recursively compute tree decomposition $T$ of $G'$;
6 construct $T'$ by appending new bag $N_G[v]$ to $T$ and connecting it to a bag
    containing $N_G(v)$;
7 return $T'$;
```

yields an optimal solution if and only if for each vertex $v$ in the ordering the neighbourhood $N(v)$ is a safe separator in the remaining graph $G_i$. This means that it is desirable to choose the next vertex in the ordering in a way that makes it likely for that vertex to have a safe neighbourhood. In the preprocessing employed in the PACE'17 submission of PID-BT, three different heuristics are used for this. The *Min-Degree-Heuristic* selects a vertex with minimum degree in the remaining graph. The *Min-Fill-Heuristic* selects a vertex with minimum *fill*, where $\text{fill}(v) = \left|E\left(\overline{G}[N(v)]\right)\right|$ is the number of missing edges in $G[N(v)]$. Lastly, the *Min-Defect-Heuristic* selects a vertex with minimum *defect*, where $\text{defect}(v) = |\{w \in N(v) \mid |N(v) \setminus N(w)| > 1\}|$ is the number of neighbours of $v$ that are not connected to all other neighbours of $v$. Intuitively it is clear that all three heuristics try to minimise the number of edges added into the graph. Note that the Min-Fill and Min-Defect-Heuristics always select a simplicial vertex, if the graph contains any, and are thus optimal on chordal graphs.

In the next section we look into safe separators, conditions for safeness and heuristics to identify safeness in more detail. This will also improve our intuition about the heuristics above.

## 4.2 Identifying Safe Separators

The decision problem of whether a separator is safe is NP-hard, as discussed above. Nevertheless there are sufficient conditions for safeness that allow to identify at least some safe separators. In the following we give a brief and incomplete selection of such conditions with the aim of explaining the preprocessing employed by Tamaki and some simpler heuristics that we also analyse theoretically. For a more complete introduction to conditions for safeness, see the articles by Bodlaender et al. [Bod+01] and Bodlaender and Koster [BK06].

Clearly, any separator that is a clique is safe. Bodlaender also extends this to separators that are what he calls *almost clique* separators, that is separators $S$ for which there is a $s \in S$ such that $S \setminus s$ is a clique.

▶ **Lemma 4.1 (Corollary 14 in [BK06]).** If $S$ is an almost clique minimal separator of $G$, then $S$ is safe for treewidth. ◀

While these first two conditions already are useful for our analysis on hyperbolic random graphs (see Section 4.4), we also want to discuss the safeness condition used in Tamaki's preprocessing. To that end we define a *labelled minor* of $G$ as a graph $G'$ that can be obtained from $G$ by performing deletions of vertices and edges or *labelled* edge contractions, that is contractions in which the new vertex keeps the label of one of the endpoints of the contracted edge.

▶ **Lemma 4.2 (Lemma 11 in [BK06]).** Let $S$ be a separator of $G$. If for each component $C$ associated with $S$, $G \setminus C$ contains a clique on $S$ as a labelled minor, then $S$ is safe. ◀

A separator $S$ that fulfils the above condition is called *minor-safe*. Note that minor-safeness also is an extension of the two sufficient conditions for safeness mentioned above in the sense that any clique separator or almost clique minimal separator is also minor-safe. In the preprocessing step used to speed up PID-BT in the PACE'17 challenge, minor-safeness is checked heuristically in a way that may miss some minor-safe separators but never declares a non-safe separator as safe. For each component $C$, the idea is to greedily contract edges in $G \setminus (C \cup S)$ in an attempt to maximise the number of vertices adjacent to both endpoints of missing edges in $S$. This way, the algorithm tries to build a labelled minor of $G$ in which $S$ is a clique. We refer to Tamaki's publication [Tam19] for more details, as this high-level view of the heuristic suffices for our purposes.

Combining this overview of the preprocessing and the results about upper and lower bounds without preprocessing from Section 3.5, we observe that intuitively the PID-BT algorithm and the preprocessing work tightly together. Consider for example an instance $G$ that consists of a complex graph $G_a$ with higher treewidth and a tree or series parallel graph $G_b$ that is appended to $G_a$.

If PID-BT tries to solve such an instance, the addition of $G_b$ to $G_a$ leads to an increase in the running time that is exponential in the treewidth of $G_a$. The reason for this is that PID-BT uses a value of $k$ that is determined by the complex graph $G_a$ when searching for I-blocks, O-blocks and PMCs in the appended simple graph $G_b$. This seems to be unnecessary and is indeed prevented by the preprocessing if the greedy heuristics are able to find a safe separator that separates the appended structure from the rest of the graph. In this case the PID-BT algorithm is then called separately on the $G_a$ and $G_b$. This way, the value of $k$ that is used on the simple graph is no longer determined by the treewidth of the more complex graph and thus the number of I-blocks, O-blocks and PMCs is drastically lower. In the case in which the appended $G_b$ is a tree, the greedy tree-decompositions obtained using the Min-Fill or Min-Defect heuristics even already contain an optimal solution of $G_b$, as trees are chordal graphs and have a perfect elimination scheme. This means that PID-BT does not have to be called on $G_b$ and is only called on $G_a$ or separated sub-graphs of $G_a$.

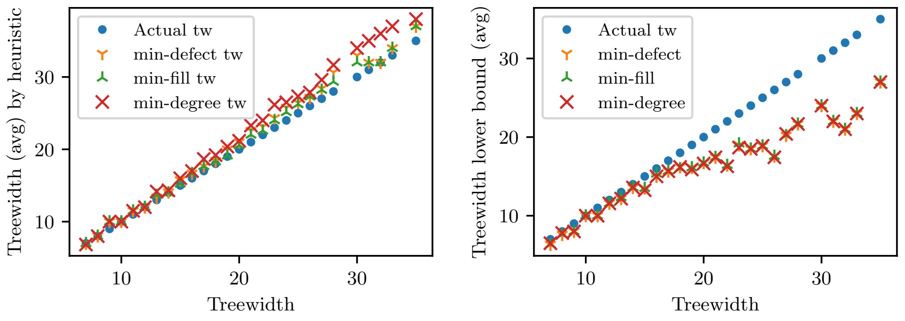
Overall, it seems like the preprocessing elegantly lets PID-BT bypass some of the cases in which otherwise the running time would increase significantly. This is confirmed by the experiments reported on in Section 2.2.

In the remainder of this section, we show the results of more fine-grained experiments that highlight the generation of greedy tree decompositions and the use of safe separators in order to split the graph. In Section 4.4 we proof interesting properties about the heuristics used in the preprocessing that partially explain some of our empirical observations and derive further new conditions for safeness.

## 4.3 Empirical Analysis

To investigate the effectiveness of the preprocessing on HRGs, we conducted some experiments, using the general set-up for the experiments as described in Section 2.2. We ran PID-BT with preprocessing on graphs of different sizes, with small adjustments to the source code in order to collect information about the greedy tree decompositions, found safe separators and the safely separated sub-instances. In order to reduce the effects of random noise, five graphs with different random seeds were generated for each graph size.

It was found that the greedy heuristics are very good at finding close to optimal tree decompositions. Figure 4.1 shows the average widths achieved by the greedy heuristics for treewidth and the lower bounds extracted from safe separators. We note that the data for the plot was obtained by running

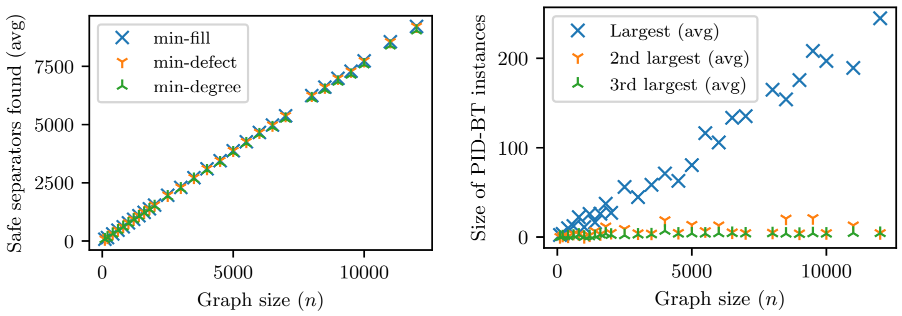


**(a)** Widths of greedy tree decompositions. **(b)** Lower bounds for treewidth.

**Figure 4.1:** Comparison of lower and upper bounds from treewidth heuristics in preprocessing of PID-BT on HRG instances with $\alpha = 0.75$, and average degree 10.

the adapted PACE'17 submission of PID-BT on graphs of different sizes with 5 different random seeds per graph size. The resulting instances were then grouped by treewidth, meaning that each data point shown in Figure 4.1 is the average width of greedy tree decompositions produced by one of the heuristics on a selection of graphs with the same treewidth and potentially heterogeneous vertex count. Especially the Min-Fill Heuristic is able to find very close to optimal tree decompositions even on the bigger instances with treewidth up to 30 and more than 10.000 vertices. The lower bounds derived from the safe separators found in the greedy tree decompositions do not differ based on the used greedy heuristic. This indicates that the different tree decompositions contain equally big detectable safe separators. Overall, the lower bounds do not match the actual treewidth of the graphs, especially on larger instances of higher treewidth. However, this does not seem to be a problem, as shown by the next experiments.

Figure 4.2 (a) shows the number of detected safe separators in greedy tree decompositions of HRGs. The plot shows the average number over a sample of five graphs generated with different random seeds for each vertex count. The number of found safe separators appears to be linear in the size of the graph with for example about 7700 safe separators in a HRG with 10000 vertices. After splitting the graph along these safe separators, only small instances remain, for many of which the width of a greedy tree decomposition is already below the lower bound. As depicted in Figure 4.2 (b), the remaining instances, which are not solved by the preprocessing and have to be solved with PID-BT, are very



**(a)** Number of found safe separators. **(b)** Sizes of sub-instances after preprocessing.

**Figure 4.2:** Safe separators detected and size of remaining non-greedily solved sub-instances after preprocessing of PID-BT on HRG instances with $\alpha = 0.75$, and average degree 10.

small. The plot shows the average size of the largest, second largest and third largest remaining sub-instance over a sample of five HRGs with different random seeds for each vertex count. The size of the largest sub-instance seems to be roughly linear in the vertex count of the input graph with about 200 vertices in a 10000 vertex graph. In most cases there remains only one sub-instance of substantial size and the other remaining safely separated sub-instances contain only a negligibly small amount of vertices.

Overall, we conclude that both parts of the preprocessing perform very well on hyperbolic random graphs. First, especially the Min-Fill heuristic is able to construct almost optimal tree decompositions. Secondly, the greedy search for safe separators is able to identify many safe separators that allow the instance to be split into sub-instances of considerably reduced size. We conduct a theoretical analysis of the preprocessing in Section 4.4, partially explaining these observations.

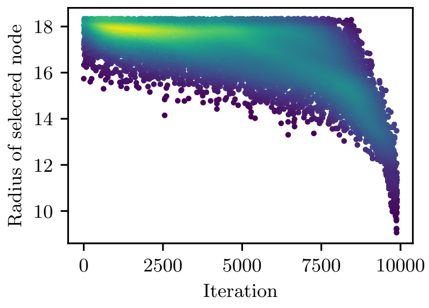
## 4.4 Theoretical Analysis

While the empirical analysis of HRG instances can already give some perspective on the preprocessing, our goal is to develop a more mathematically grounded understanding of how and why the preprocessing works on these graphs. The actual preprocessing procedure described in Chapter 4 is rather complex, so it is important to make simplifications where needed and formulate meaningful and clear directions of research. To the best of our knowledge the greedy heuristics for treewidth used in the preprocessing have so far not been studied on HRGs or similar models for real-world networks. This makes it necessary to first gain a basic understanding of these heuristics before trying to understand their role in some larger context of finding safe separators or even the whole treewidth computation.

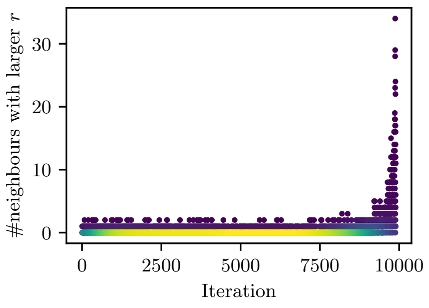
Computing a tree-decomposition of hyperbolic random graphs according to the Min-Fill, Min-Defect, or Min-Degree heuristics can be seen as a random process in which a random graph is drawn and subsequently modified according to the rules of the heuristic. Unfortunately such a random process is very hard to analyse, especially when one tries to make statements about deeper structures such as the computed treewidth or the number or even size of encountered safe separators. One simplification we therefore make is a simpler greedy heuristic based on the geometry of hyperbolic random graphs.

As HRGs are denser and have higher-degree vertices in the centre than in the outer regions of the disk, we can expect all of the heuristics mentioned above to rather pick vertices with larger radius in the beginning and vertices with smaller radius in the end. Following this intuition, we introduce the assumption of a heuristic that simply picks the remaining vertex with the largest radius. Under that assumption, the neighbourhood of each selected vertex in the remaining graph is equal to its inner neighbourhood. This is useful for our analysis, as it appears that the inner neighbourhood has more structure than the general neighbourhood of a vertex. We explore the structure of inner neighbourhoods throughout this section and in particular in Section 4.4.3. We want to stress that analysing a heuristic that selects the vertices in descending order of their radii does not seem to be a far stretch of what we can observe from, e.g. the Min-Fill heuristic. Indeed, the experiments reported on in Figure 4.3 confirm this. It can be seen that empirically the majority of selected vertices has no or only very few neighbours with larger radius at the time of their selection. This observation holds especially for smaller values of $\alpha$ and more in the earlier iterations than in the last ones. Still, overall we can conclude that the most important feature of our assumed heuristic seems to be shared by the original heuristics, at least before reaching the tightly connected core of the graph.

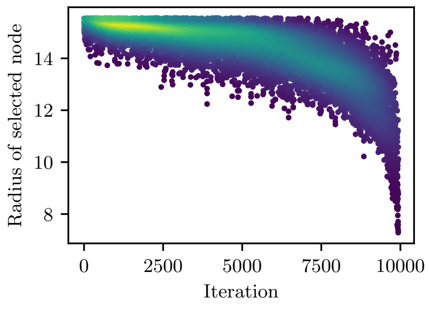
The remainder of this section discusses different approaches for proving that the inner neighbourhood of a vertex is a safe separator. In particular, we



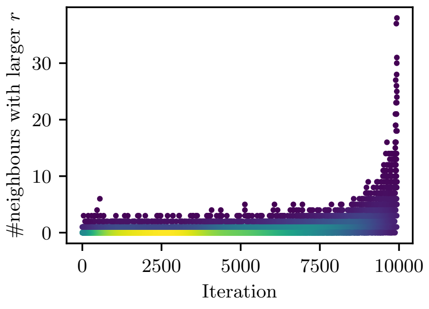
**(a)** HRG with $n = 10^4$, $\alpha = 0.55$.



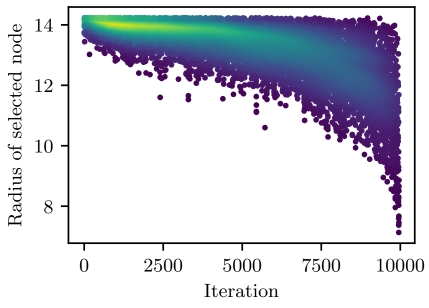
**(b)** HRG with $n = 10^4$, $\alpha = 0.55$.



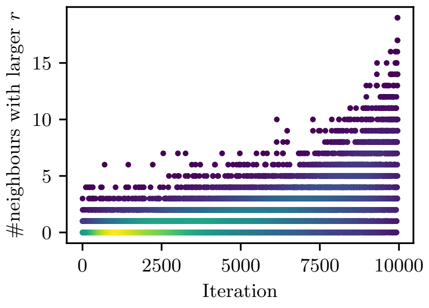
**(c)** HRG with $n = 10^4$, $\alpha = 0.75$.



**(d)** HRG with $n = 10^4$, $\alpha = 0.75$.



**(e)** HRG with $n = 10^4$, $\alpha = 0.95$.



**(f)** HRG with $n = 10^4$, $\alpha = 0.95$.

**Figure 4.3:** Experiment in which HRGs with $10^4$ vertices average degree about 23 and different values of $\alpha$ were generated and vertices were deleted according to the Min-Fill heuristic. The left column shows the radius of the vertex deleted in each iteration, the right column shows how many neighbours had a larger radius than the deleted vertex of each iteration. Brighter colours signify a higher density of points in the plots.

show that there is an expected linear number of vertices with clique (inner) neighbourhoods in Section 4.4.1 and give new conditions for the safeness of separators in Section 4.4.2.

### 4.4.1 Simplicial Vertices

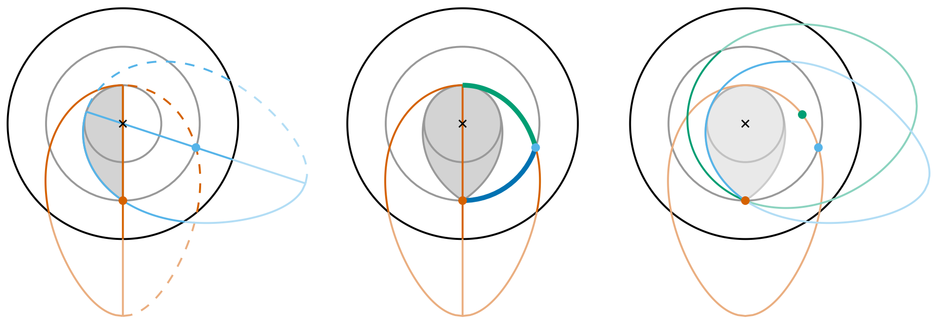
In resemblance to the definition of simplicial vertices, we call a vertex *inner simplicial* if its inner neighbourhood is a clique. In the following we want to show that asymptotically for large $n$, a constant fraction of all vertices is inner simplicial in expectation. In the proof for this, we consider different parts of the inner neighbourhood and argue that these parts are likely to be empty, form a clique, etc. In order to nicely formulate these statements, we establish labels for certain areas within the inner neighbourhood of a vertex.

Let $v$ be a vertex. First, we say that the *left half* of the neighbourhood of $v$ consists of all points in the neighbourhood of $v$ for which there exists a positive angle $\theta' \leq \pi$ such that $\theta(u) + \theta' = \theta(v)$. Equivalently, for points in the *right half* of $v$'s neighbourhood, there is a positive angle $\theta'$ such that $\theta(u) - \theta' = \theta(v)$. The *left inner* and *right inner* neighbourhood of $v$ are defined analogously.

Next, we assume that $r(v) > \frac{R}{2}$ and call the points with radius $r(v)$ and angle $\theta(v) \pm \theta(r(v), r(v))$ the left and right *corner points* of the inner neighbourhood of $v$. Regardless of their actual existence in a graph, it is sometimes useful to talk about vertices lying on these points. These hypothetical vertices are called left and right *corner vertices* of the inner neighbourhood of $v$ or simply left and right corner vertices of $v$. The corner vertices allow us to define a region in which all contained vertices form a clique. We call this the *central inner neighbourhood* of $v$ and define it as the subset of the inner neighbourhood of $v$ that

- has radius at most $R - r(v)$,
- or is part of the left inner neighbourhood of both $v$ and of $v$'s right corner vertex,
- or, symmetrically, is part of the right inner neighbourhood of $v$ and of $v$'s left corner vertex.

The left and right central inner neighbourhood are defined canonically as the intersection of the central inner neighbourhood and the left or right inner neighbourhood. See Figure 4.4 (a) for an illustration that visualises the construction



**(a)** Visualisation of the left central inner neighbourhood.

**(b)** The possible positions of $w$ in the first (blue) and second (green) case of the proof of Lemma 4.3.

**(c)** The second case of the proof of Lemma 4.3 ($w$: green, $c$: light blue).

**Figure 4.4:** Visualisations related to the central inner neighbourhood and Lemma 4.3.

of the left central inner neighbourhood of a vertex. As mentioned in the preliminaries, there is a direct correspondence between regions of the hyperbolic disk and the sets of vertices that are contained in such a region. This allows us to generalise the names of regions such as for example *left central inner neighbourhood* to also refer to the set of vertices that lie within that region. It will always be clear from the context if we are referring to a set of vertices or to a region of the hyperbolic disk. We can now prove the claimed statement about the central inner neighbourhood, which we later use to derive a lower bound for the probability with which a vertex is inner simplicial.

▶ **Lemma 4.3.** Let $v$ be a vertex with radius $r > \frac{R}{2}$ and let $u$ be a neighbour in the central inner neighbourhood of $v$. Then $u$ is adjacent to all vertices in the inner neighbourhood of $v$. ◀

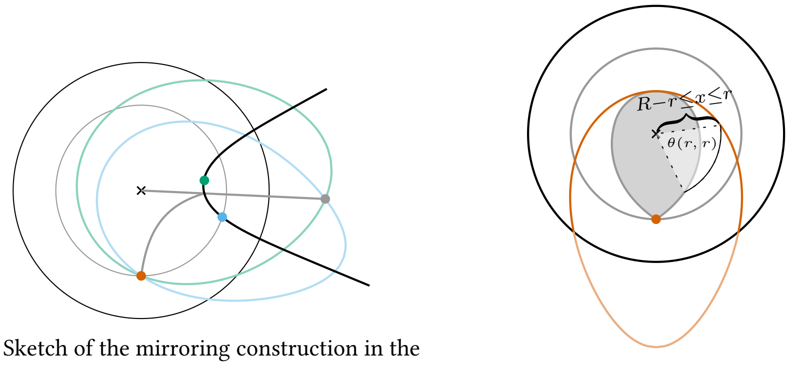
*Proof.* We want to show that any vertex $w$ in the inner neighbourhood of $v$ connects to $u$. As the left and right half of $v$'s inner neighbourhood form a clique, it remains to consider cases where $u$ and $w$ lie in opposite halves of $v$'s neighbourhood, so without loss of generality assume that $u$ is in the left and $w$ in the right part of the neighbourhood of $v$. Also, assume that $r > R/2$ as otherwise, the inner neighbourhood of $v$ forms a clique anyway.

Next, as decreasing the radius only grows the neighbourhood of $w$, it suffices to show that the statement holds for all $w$ on the outer boundary of $v$'s right inner neighbourhood.

First note that for the outermost point of the boundary, that is when $w$ lies on the position of the right corner vertex of $v$, $c = (r, \theta(v) + \theta(r, r))$, the statement holds, because either $r(u) \leq R - r$ or $u$ has distance at most $R$ from $w$ by the definition of the central inner neighbourhood. All other positions of $w$ on the boundary are characterised either by having radius $r(w) = r$ or by having distance exactly $R$ to $v$. In the first case, depicted a dark blue line in Figure 4.4 (b), it is clear that $u$ and $w$ are adjacent if $r(u) \leq R - r$. Otherwise $w$ lies between $c$ and $u$ and so $u$ and $w$ are adjacent by Lemma 2.3.

Now consider the second case in which $w$ has distance $R$ from $v$ (and thus lies on the green line in Figure 4.4 (b)). We want to show that the inner neighbourhood of $v$'s right corner vertex $c$ is contained in $N(w)$, as can depicted in Figure 4.4 (c). The circle with radius $R$ around $w$ intersects the one around $c$ in $v$. We proof that a second intersection is in some point with radius larger than $r$. To see this, observe that the position $x$ of the second intersection can be obtained by mirroring $v$ along the line $CW$ through $c$ and $w$ (see Figure 4.5 (a)). As the origin is on the same side as $v$ with respect to $CW$, the line segment from the origin to $x$ intersects $CW$ in some point with equal distance from $x$ as from $v$. By the triangle inequality this means that $r(x) \geq r$. Two (non-identical) circles intersect in at most two points, thus the circles around $w$ and $c$ have no other intersections. This means that if the neighbourhood of $w$ contains the neighbourhood of $c$ at radius $r$, then also for all radii up to $r$. This holds, since at radius $r$ the neighbourhood of $w$ goes from $\theta(v)$ to $\theta(v) + 2\theta(r, r(w))$ while the neighbourhood of $c$ goes from $\theta(v)$ to $\theta(v) + 2\theta(r, r)$, where $\theta(r, r(w)) > \theta(r, r)$ (see Figure 4.4 (c)). In consequence, $N(w)$ contains $c$'s inner neighbourhood. As $u$ is in the inner neighbourhood of $c$, it is adjacent to $w$. ■

We call the region that remains after subtracting the central inner neighbourhood from the inner neighbourhood of a vertex the *peripheral* inner neighbourhood. From the lemma, we deduce that a vertex is inner simplicial if its peripheral inner neighbourhood is empty, because in this case, only the central inner neighbourhood remains, which forms a clique. We can get a lower bound on the probability for this to happen by finding an upper bound on the measure of the peripheral neighbourhood.



**(a)** Sketch of the mirroring construction in the proof of Lemma 4.3.

**(b)** Illustration of the integral from Lemma 4.4.

**Figure 4.5:** Visualisations for the proofs of Lemma 4.3 and Lemma 4.4.

▶ **Lemma 4.4.** Let $v$ be a vertex with radius $r > \frac{R}{2}$ and let $S$ be its peripheral inner neighbourhood. Then, the measure of $S$ is

$$\mu(S) \leq \frac{2}{\pi} e^{-R(\alpha-\frac{1}{2})-r(1-\alpha)} \cdot \left(1 + \Theta\left(e^{R-2r}\right)\right) \cdot (1 - \Theta(e^{-\alpha r})).$$

◀

*Proof.* Let $S'$ be the area of the right peripheral inner neighbourhood. Then we have $2\mu(S') = \mu(S)$. We can compute $\mu(S')$ by integrating the probability density function over all points in $S'$. The minimum radius of any point in $S'$ is $R - r$. For any radius $x$ between $R - r$ and $r$, $S'$ spans all angles from the edge of the neighbourhood of the right corner vertex of $v$'s inner neighbourhood to the edge of the neighbourhood of $v$, that is from $\theta(r, x) - \theta(r, r)$ to $\theta(r, x)$. This is depicted in Figure 4.5 (b). The angular coordinates of $v$ and the corner vertices of its inner neighbourhood differ exactly by $\theta(r, r)$. Hence, we get

$$\mu(S') = \int_{R-r}^{r} \int_{\theta(r,x)-\theta(r,r)}^{\theta(r,x)} f(x) \mathrm{d}\theta \mathrm{d}x = \theta(r, r) \int_{R-r}^{r} f(x) \mathrm{d}x.$$

We substitute $\theta(r, r)$ according to Lemma 2.2 to obtain.

$$\mu(S') \leq \theta(r, r) \int_{0}^{r} f(x) \mathrm{d}x$$

$$= 2 \cdot e^{\frac{R-2r}{2}} \left(1 + \Theta\left(e^{R-2r}\right)\right) \cdot \int_0^r f(x)\mathrm{d}x$$

We note that the integral equals $\frac{1}{2\pi}\mu(B_0(r))$. Thus, we can simplify the integral according to Equation (2.2) and get

$$\begin{aligned}
&= 2 \cdot e^{\frac{R-2r}{2}} \left(1 + \Theta\left(e^{R-2r}\right)\right) \cdot \frac{1}{2\pi} e^{\alpha(r-R)} (1 - \Theta(e^{-\alpha r})) \\
&= \frac{1}{\pi} e^{-R\left(\alpha - \frac{1}{2}\right) - r(1-\alpha)} \cdot \left(1 + \Theta\left(e^{R-2r}\right)\right) \cdot (1 - \Theta(e^{-\alpha r})),
\end{aligned}$$

which matches the claimed bound. ■

Holding on to the notation from the above lemma, for $c > 0$ let $S_{R-c}$ be the peripheral inner neighbourhood of a vertex with radius $R - c$.

▶ **Corollary 4.5.** The measure $\mu(S_{R-c})$ of the peripheral inner neighbourhood of a vertex with radius $R - c$ can be upper bounded as

$$\mu(S_{R-c}) \leq \frac{2}{n\pi} \cdot e^{C/2 + c(1-\alpha)} \cdot \left(1 + \Theta\left(n^{-2} \cdot e^{2c-C}\right)\right)$$

◀

*Proof.* Using Lemma 4.4, we can compute an upper bound for the measure of $S_{R-c}$ as

$$\begin{aligned}
\mu(S_{R-c}) &\leq \frac{2}{\pi} e^{-R\left(\alpha - \frac{1}{2}\right) - (R-c)(1-\alpha)} \cdot \left(1 + \Theta\left(e^{-R+2c}\right)\right) \cdot \left(1 - \Theta\left(e^{-\alpha R + \alpha c}\right)\right) \\
&= \frac{2}{\pi} e^{-\frac{1}{2}R + c(1-\alpha)} \cdot \left(1 + \Theta\left(e^{-R+2c}\right)\right) \cdot \left(1 - \Theta\left(e^{-\alpha R + \alpha c}\right)\right) \\
&\leq \frac{2}{\pi} e^{-\frac{1}{2}(2\log n + C) + c(1-\alpha)} \cdot \left(1 + \Theta\left(e^{-R+2c}\right)\right) \cdot \left(1 - \Theta\left(e^{-\alpha R + \alpha c}\right)\right) \\
&= \frac{2}{n\pi} \cdot e^{C/2 + c(1-\alpha)} \cdot \left(1 + \Theta\left(n^{-2} \cdot e^{2c-C}\right)\right).
\end{aligned}$$

■

We can use this to prove that at least a constant fraction of vertices is inner simplicial.

▶ **Theorem 4.6.** Let $G$ be a hyperbolic random graph. Then, the expected number of inner simplicial vertices in $G$ is in $\Omega(n)$. ◀

*Proof.* By Lemma 4.3 a vertex $v$ is inner simplicial if its peripheral inner neighbourhood is empty. Assuming $r(v) \geq R - c$ for a constant $c$, we have $\mu(S_{R-c}) \leq \frac{2}{n\pi} \cdot e^{C/2+c(1-\alpha)}(1+o(1))$ from Corollary 4.5. This means that asymptotically there is a constant $c'$ such that $\mu(S_{R-c}) \leq \frac{c'}{n}$. Using the inequality from Lemma 2.4, we can give the following lower bound on the probability of the event of an empty peripheral inner neighbourhood

$$\begin{aligned}\Pr[v \text{ is inner simplicial} \mid r(v) \geq R - c] \geq \Pr\left[\nexists u \in S_{R-c}\right] &= (1 - \mu(S_{R-c}))^n \\ &\geq e^{-c'}\left(1 - \frac{c'^2}{n}\right) = e^{-c'}(1 - o(1)) \in \Omega(1).\end{aligned}$$

This means that $v$ is inner simplicial with at least constant probability. Note that this probability increases if the radius of $v$ is increased. For each vertex $v \in V$ we can define an indicator random variable $X_v$ to be 1 if $v$ is simplicial according to the above condition and 0 otherwise. In a hyperbolic random graph the probability for a vertex $v$ to have radius larger than $R - c$ is asymptotically constant:

$$\Pr[r(v) \geq R - c] = 1 - \mu(B_O(R - c)) = 1 - e^{-\alpha c} \cdot \left(1 - \Theta\left(e^{-\alpha R + \alpha c}\right)\right) \in \Theta(1).$$

Thus, we get $\Pr[X_v = 1] \geq \Pr[r(v) \geq R - c] \cdot \Omega(1) \in \Omega(1)$. The number $X$ of inner simplicial vertices in $G$ can be written as the sum of these random variables $X = \sum_{v \in V} X_v$. By the linearity of expectation, this implies that the expected number of inner simplicial vertices is at least $\Omega(n)$. ■

The following more general statement follows similarly.

▶ **Corollary 4.7.** Let $G$ be a hyperbolic random graph. In expectation $G$ has $\Omega(n)$ simplicial vertices. ◀

*Proof.* Let $c > 0$ be a constant and $v$ be a vertex with radius $R - c$. In the proof of Theorem 4.6 we use the upper bound on $\mu(S_{R-c})$ from Corollary 4.5 in order to derive a lower bound for the probability of a vertex to be inner simplicial. We can add the area $S_{\text{out}} = B_0(R) \cap B_{R-c}(R) \setminus B_0(R - c)$ of the outer neighbourhood of $v$ to that measure in order to enforce that $v$ has no other neighbours if the considered area contains no vertices. We get

$$\mu(S_{R-c} \cup S_{\text{out}}) \leq \mu(S_{R-c}) + 2\int_{R-c}^{R}\int_{0}^{\theta(R-c,R-c)} f(x)\mathrm{d}\theta\mathrm{d}x$$

$$= \mu(S_{R-c}) + 4e^{-\frac{1}{2}R+c}\left(1 \pm \Theta\left(e^{-R+2c}\right)\right) \int_{R-c}^{R} f(x)\mathrm{d}x$$

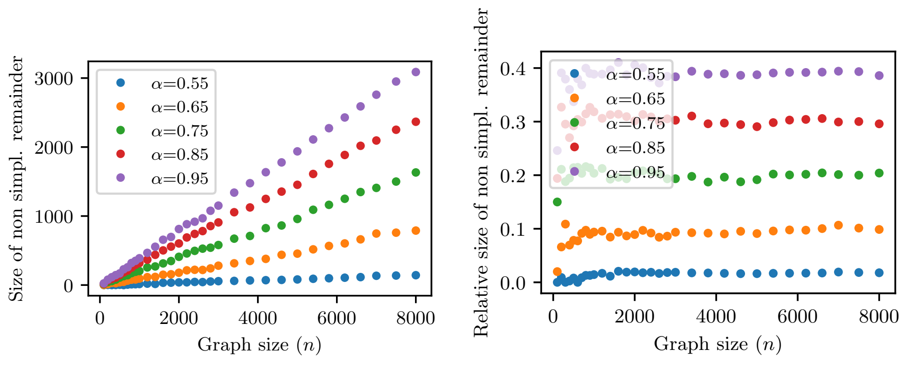
where $\int_{R-c}^{R} f(x)\mathrm{d}x = \frac{1}{2\pi}\mu(B_0(R) \setminus B_0(R-c)) \leq 1$. Thus, the term simplifies to

$$\leq \mu(S_{R-c}) + 4\frac{1}{n}e^{\frac{1}{2}C+c}\left(1 \pm \Theta\left(e^{-R+2c}\right)\right) \in O\left(\frac{1}{n}\right).$$

This means that the probability for a vertex with radius $R-c$ to be inner simplicial and have no outer neighbours is at least constant. The rest of the proof follows as in the proof of Theorem 4.6 by defining indicator random variables for the simpliciality of each vertex and using the linearity of expectation on their sum. ■

These two results offer a partial explanation for the effectiveness of the preprocessing procedure used in the implementation of PID-BT. As discussed in Section 4.1, the Min-Defect and Min-Fill heuristic choose simplicial vertices if the graph contains any. Corollary 4.7 implies that in expectation on a hyperbolic random graph, these heuristics first select and remove $\Omega(n)$ simplicial vertices before selecting any other vertex. For each removed simplicial vertex, the constructed tree decomposition contains two adjacent bags whose intersection is the neighbourhood of the simplicial vertex. This neighbourhood is subsequently recognised as a safe separator that separates the simplicial vertex from the remainder of the graph. Thus in expectation, the preprocessing removes at least $\Omega(n)$ vertices from the graph, which matches our empirical observations. We can expect the heuristics to find even more simplicial vertices after removing all simplicial vertices that are initially present in the graph. This is empirically confirmed by the experimental results presented in Figure 4.6. Here, we plot the number of vertices that remain after iteratively removing all simplicial vertices in HRGs of different size and power-law exponents. The set-up for this experiment is the same as in Section 4.3. It can be seen that iteratively removing simplicial vertices removes a constant fraction of vertices, that is larger for smaller values of $\alpha$.

We also conducted experiments to empirically verify our theoretical results. Figure 4.7 shows the number of simplicial vertices in GIRGs generated with different average degrees and power-law exponents. We used GIRGs instead of HRGs, because the HRG generator does not reliably control the average degree,



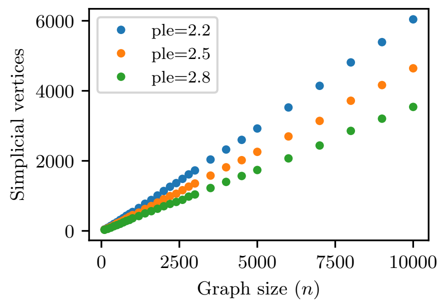
**(a)** Number of remaining vertices. **(b)** Relative share of remaining vertices.

**Figure 4.6:** Average number of remaining vertices after iteratively removing all simplicial vertices in HRG instances with $\alpha = 0.75$, and average degree 10.

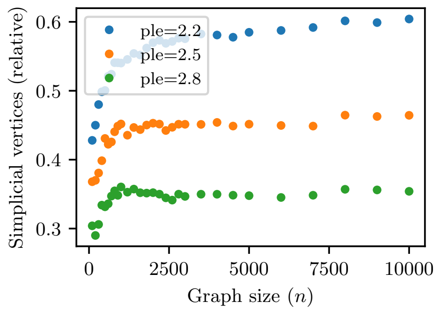
especially on instances with only a few thousand vertices. For the purpose of our experiment GIRGs and HRGs can be seen as equivalent, with the difference that instead of a parameter $\alpha$, we now control the power-law exponent directly. As in Section 4.3 each data point is the average of five GIRGs that were sampled with different random seeds.

The plots confirm our theoretical result by showing a roughly linear number of simplicial vertices. However, even though the above theorem and corollary seem to capture the asymptotic behaviour observed in our experiments, they also appear to miss some aspects. For example, note that the plots of Figure 4.7 (a) and Figure 4.7 (b) show a higher fraction of simplicial vertices in graphs with smaller power-law exponent, which represents smaller values of $\alpha$. Also Figure 4.7 (c) and Figure 4.7 (d) show more simplicial vertices in graphs with lower average degree, which represents smaller values of $C$. This stands in contrast to the dependency on $\alpha$ and $C$ that we find in Corollary 4.5. Here the lower bound for the probability of a vertex to be inner simplicial is higher for higher values of $\alpha$ and smaller values of $C$. This makes it seem like the (inner) simplicial vertices explained by our theorem and corollary are only a fraction of the (inner) simplicial vertices that are actually occurring. Still, even though our proof does not yet cut to the core of the phenomenon, we are able to asymptotically explain an expected linear amount of simplicial vertices.

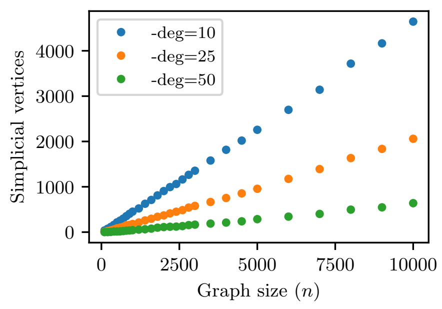
In summary, our theoretical analysis contributes to our understanding of the



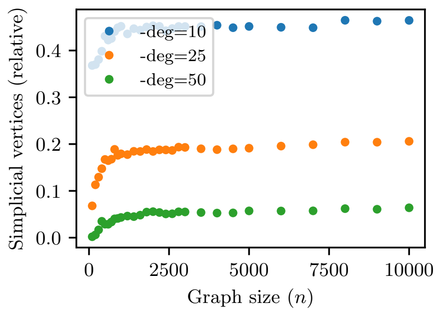
**(a)** Number of simplicial vertices.



**(b)** Relative share of simplicial vertices.



**(c)** Relative share of simplicial vertices.



**(d)** Relative share of simplicial vertices.

**Figure 4.7:** Average number of simplicial vertices in GIRG instances with different power-law exponents and average degree 10 ((a), (b)) or power-law exponent 2.5 and different average degrees ((c), (d)).

preprocessing procedure as explained above. Additionally, it also serves as the basis for a lower bound on the running time of PID-BT without preprocessing that we explore in Theorem 3.18. In the next section, we develop a deeper understanding of safeness and the inner neighbourhood of vertices in hyperbolic random graphs.

### 4.4.2 Safe Separators

In the following, we want to explore further conditions for the safeness of separators with the goal of gaining a deeper understanding for the preprocessing. We give a counterexample for an intuitively promising approach and develop a novel way of looking at safeness that allows to formulate new sufficient conditions for safeness.

Intuitively, one might suspect that small minimal separators are safe if they are a lot smaller than the treewidth of a graph. We can expect hyperbolic random graphs to contain many small minimal separators in the outer region of the disk, so it would be useful for our analysis if this intuition could be formulated as a general statement. Unfortunately, the following counterexample shows that this is not possible.

▶ **Lemma 4.8.** For any constant $C > 4 \in \mathbb{N}$, there exists a graph with a (connected) minimal separator $S$ with $|S| \leq \frac{1}{C}\text{tw}(G)$ that is not safe. ◀

*Proof.* We show the statement by constructing a graph $G$ of treewidth $C^2$ with a connected minimal separator $S$ of size $C$ that is not safe for treewidth.

Let $G$ contain a clique with $C^2$ vertices, another clique with $C(C-1)+2$ vertices, a single vertex $v$ and vertices $S_1, \ldots, S_C$ that form the separator $S$ (as depicted in Figure 4.8). The vertices of $S$ form a path and additionally each $S_i$ is connected to $v$ and to all vertices in the $C(C-1)+2$-clique. Also, $v$ is connected to all vertices in the $C^2$-clique.

$G$ has treewidth at least $C^2$, because the $C^2$-clique and $v$ form a clique of size $C^2+1$. Further, $G$ has treewidth at most $C^2$, because the remainder of $G$ can easily be decomposed by forming bags $B_i$ containing $v$, the $C(C-1)+2$-clique, $S_i$ and $S_{i+1}$ for adjacent $S_i$, $S_{i+1}$. As $C \geq 5$, $|B_i| \leq C^2$. These bags along with a bag of the other clique and $v$ can be connected to a bag of just $v$.

$S$ is not safe for treewidth, because when completed into a clique, $S$ and the $C(C-1)+2$ clique form a $C^2+2$ clique, meaning the graph would no longer

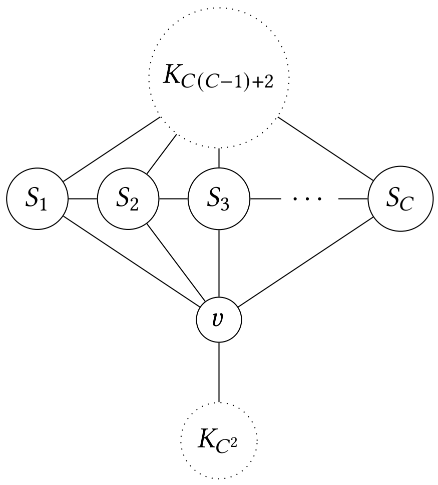


**Figure 4.8:** Sketch of the graph constructed in the proof of Lemma 4.8.

have treewidth $C^2$. As $S$ is connected and $|S| = C \leq (C^2 + 1)/C = \text{tw}(G)/C$, this proves the statement. ■

Note that the example in Lemma 4.8 can easily be extended to a separator $S$ with small constant pathwidth or treewidth. If the pathwidth of $S$ is, e.g. increased to $k$, $k$ vertices $S_i, \ldots, S_{i+k-1}$ have to be together in a bag with the upper clique and $v$. Following the above proof, we need to make sure that these bags do not increase the treewidth $C^2$ of $G$, because otherwise, $S$ would no longer be unsafe. This holds if $C$ is big enough that $C(C-1)+2+k+1$ is still at most $C^2 + 1$.

Consequentially, we need to look into safeness from a different perspective. In the following, we restate some basic facts about safe separators, which allow us to derive new conditions for safeness.

In the original paper on safe separators Bodlaender and Koster [BK06] give the following useful lemma.

▶ **Lemma 4.9 (Lemma 5 in [BK06]).** For every graph $G$, and every separator $S$ of $G$, the treewidth of $G$ is at most the maximum over all components $Z$ of $G \setminus S$ of the treewidth of $G[S \cup Z] + \text{clique}(S)$. ◀

This follows easily by combining the tree decompositions of the individual components into a tree decomposition of $G$. This view directly implies the following corollary, giving an equivalent characterisation of safeness.

▶ **Corollary 4.10 (see Definition 4 in [BK06]).** Let $S$ be a separator in $G$. $S$ is safe if and only if the treewidth of $G$ is at least the maximum over all components $Z$ of $G \setminus S$ of the treewidth of $G[S \cup Z] + \text{clique}(S)$. ◀

*Proof.* If $S$ is safe then $\text{tw}(G) = \text{tw}(G + \text{clique}(S))$ and there is a tree decomposition of $G$ with a bag containing $S$ of width $\text{tw}(G)$. This tree decomposition can be turned into equally or less wide tree decompositions of $G[S \cup Z] + \text{clique}(S)$ for each component $Z$ associated with $S$ by intersecting each bag with $S \cup Z$.

Analogically, if there are tree decompositions of $G[S \cup Z] + \text{clique}(S)$ for each component $Z$ associated with $S$ with a maximum width at most $\text{tw}(G)$, these can be joined at a bag containing $S$. This gives an optimum width tree decomposition of $G$ and also $G + \text{clique}(S)$, showing that $S$ is a safe separator. ■

Hence, in order to show the safeness of a separator $S$, it is sufficient to show that $\text{tw}(G[S \cup C] + \text{clique}(S)) \leq \text{tw}(G)$ for all components $C$ associated with $S$. We can break this down to the individual components and say that a separator $S$ is *safe with respect to* $C$, if $\text{tw}(G[S \cup C] + \text{clique}(S)) \leq \text{tw}(G)$. Then $S$ is safe if and only if it is safe with respect to all components associated with it. This lets us for example restate the sufficient condition from Lemma 4.2 in a per component perspective.

▶ **Lemma 4.11.** Let $S$ be a separator of $G$ and $C$ be a component associated with $S$. $S$ is safe with respect to $C$ if $G \setminus C$ contains a clique on $S$ as a labelled minor. ◀

*Proof.* As $G \setminus C$ has a minor with $S$ as a clique, $G[S \cup C] + \text{clique}(S)$ is a minor of $G$. Thus, $\text{tw}(G[S \cup C] + \text{clique}(S)) \leq \text{tw}(G)$, as for any minor $H$ of $G$, $\text{tw}(H) \leq \text{tw}(G)$. ■

While this may already be interesting on its own, the real power of safeness with respect to a component is that we can conclude that a separator is safe from the fact that it is safe with respect to all of its components even though for each component the safeness might be due to different sufficient conditions. In the following, we therefore want to introduce additional sufficient conditions for the safeness of a separator with respect to a component. The intuition behind our approach for this is somewhat related to the refuted approach discussed in the beginning of this section. Basically, the idea comes from the fact that hyperbolic random graphs are dense in the centre and sparse in the outer regions and this should make it possible to conclude the safeness of small separators lying in the outer part of the disk. With the following definition we can clarify what is meant by sparse and formulate sufficient conditions, that rely on knowing a lower bound for the treewidth in the graph.

▶ **Definition 4.12.** We say that a vertex set $C \subseteq V(G)$ has *subtree-width at most* $k$, if there is an optimum width tree decomposition of $G$ with a subtree $\tau$ such that the union of all bags in $\tau$ contains $C$ and each bag in $\tau$ has at most $k + 1$ vertices. The smallest $k$ such that $C$ has subtree-width at most $k$ is called the *subtree-width* of $C$. ◀

Informally, if the subtree-width of a set of vertices is $k$ then the set of vertices can safely be put into bags of size $k + 1$ and there are no bigger bags in between. The following statements follow easily, but are useful to derive our sufficient conditions for safeness with respect to a component.

▶ **Lemma 4.13.** Let $S$ be a safe separator in $G$. Then $S$ has subtree-width at most $|S| - 1$. ◀

*Proof.* There is an optimum width tree decomposition of $G$ with a bag containing $S$. We can connect a bag containing only $S$ to this bag to see that $S$ has subtree-width at most $|S| - 1$. ■

▶ **Lemma 4.14.** Let $D \subseteq V(G)$ be a vertex set in $G$ and let $C \subseteq D$ be a subset of $D$. Then the subtree-width of $C$ is at most the subtree-width of $D$. ◀

*Proof.* Follows directly because if $D$ is contained in the bags of a subtree of a tree decomposition, then these bags also contain $C$. ■

▶ **Corollary 4.15.** Let $C \subseteq V(G)$ be a vertex set in $G$. Then the subtree-width of $C$ is at most the treewidth of $G$. ◀

*Proof.* Follows from Lemma 4.14 because the subtree-width of $V(G)$ equals the treewidth of $G$. ■

If a separator has small subtree-width, it can serve as a good interface to the rest of the graph even if it contains many vertices. The following lemma constitutes the most important insight about subtree-width.

▶ **Lemma 4.16.** Let $S$ be a separator in $G$, $C$ be a component associated with $S$ and $\ell \leq \mathrm{tw}(G)$ be a lower bound for the treewidth in $G$. If the neighbourhood of $S$ in $C$, $N(S) \cap C$, has subtree-width at most $k$ in $G[C]$ such that $|S| + k \leq \ell + 1$, then $S$ is safe with respect to $C$. ◀

*Proof.* There is an optimum width tree decomposition of $G[C]$ in which the vertices of $N(S) \cap C$ all appear in a subtree $\tau$ with bags of size at most $k + 1$. We construct a tree decomposition of $G[C \cup S] + \mathrm{clique}(S)$ by adding the vertices of $S$ to the bags of $\tau$. This is indeed a tree decomposition as the bags are connected as a tree, all vertices of $C \cup S$ appear in some bag, both for edges in $C$ and for edges between $S$ and $C$ there is a bag containing the endpoints, and the bags of each vertex form a subtree. This tree decomposition has width at most $\mathrm{tw}(G)$, as the modified bags have size at most $|S| + k \leq \ell + 1$ and thus shows that $S$ is safe with respect to $C$. ■

We derive two corollaries that capture interesting special cases of the above lemma.

▶ **Corollary 4.17.** Let $S$ be a separator in $G$, $C$ be a component associated with $S$ and $\ell \leq \mathrm{tw}(G)$ be a lower bound for the treewidth in $G$. If the neighbourhood of $S$ in $C$ $N(S) \cap C$ is safe for treewidth and $|S| + |N(S) \cap C| \leq \ell + 1$ then $S$ is safe with respect to $C$. ◀

*Proof.* Follows from Lemma 4.13 and Lemma 4.16, because $N(S) \cap C$ is safe. ■

Note that the condition in the above corollary is in particular fulfilled by a separator with clique neighbourhood in the considered component. Building upon Corollary 4.15, the following statement gives a safeness condition with respect to small components.

▶ **Corollary 4.18.** Let $S$ be a separator in $G$, $C$ be a component associated with $S$ and $l \leq \mathrm{tw}(G)$ be a lower bound for the treewidth in $G$. If $|S| + |C| \leq l + 1$ then $S$ is safe with respect to $C$. ◀

*Proof.* Follows from Lemma 4.16 and Corollary 4.15, because in $G[C]$ the set $N(S) \cap C$ has subtree-width at most $tw(G[C]) \leq |C| - 1$. ■

This condition is especially relevant for hyperbolic random graphs as it intuitively tells us that the inner neighbourhood of vertices with poly-logarithmic degree is safe with respect to all components except one.

▶ **Conjecture 4.19.** Let $c > 0$ be a constant and let $v$ be a vertex with radius at least $R - c \log \log n$. We conjecture that with high probability, the inner neighbourhood of $v$ has one associated component of linear size while all other associated components have poly-logarithmic size. ◀

The motivation behind the conjecture is that for a vertex adjacent to the inner neighbourhood of $v$, there are two cases: either it is connected to the giant component or not in which case it is part of a different associated component. A lemma by Bläsius et al. [Blä+18, Lemma 6] states that with high probability there exists a vertex adjacent to every other vertex in $B_O(1/\alpha(\log n - \log \log n)$. Thus, with high probability a component associated with the inner neighbourhood of $v$ that is not the giant component does not contain any vertex with radius less than $\frac{1}{\alpha}(\log n - \log \log n)$, because otherwise, it would be part of the giant component. This means that all vertices in this component have an expected poly-logarithmic degree. Similarly, we also expect the vertices of the component to lie in a narrow angular sector that exceeds the angular width of the inner neighbourhood of $v$ by at most a constant factor. Intuitively this suggests that the conjecture probably holds, even though some more work is required to rigorously formulate statements about components and their relationship to the presumed giant component.

If the conjecture holds, then it immediately follows that the inner neighbourhood of $v$ is safe with respect to the poly-logarithmic components. As the treewidth of hyperbolic random graphs is polynomial, the poly-logarithmically sized inner neighbourhood and component are asymptotically smaller than the treewidth and so the safeness follows via Corollary 4.18.

Intuitively it seems also plausible that Lemma 4.16 may be applicable for the safeness of the inner neighbourhood with relation to the giant component inside. Let $S$ be the inner neighbourhood of a vertex with radius $R - c \log \log n$ for some constant $c$ that maybe needs to be chosen suitably. The size of $S$ is poly-logarithmic with high probability, so for the application of Lemma 4.16 it would suffice to show that the neighbourhood of $S$ in the giant component has poly-logarithmic subtree-width. We conjecture that this is the case with at least constant probability. This conjecture is motivated by the fact that a vertex with radius $R - c \log \log n$ does not have any neighbour with radius smaller than $R - c \cdot c' \cdot \log \log n$ for some constant $c' > 1$ asymptotically almost surely. Thus, the neighbourhood of $S$ in the giant component also consists of vertices with large radius and is presumably also located in a small region of the disk. Intuitively, this supports the conjecture considerably.

### 4.4.3 Structure of the Inner Neighbourhood.

In the remainder of this section, we want to summarise our findings about the structure of the inner neighbourhood. Based on these insights, we want to motivate a possible approach to explain why the inner neighbourhoods of vertices with large radius are likely to be safe separators. The exact details of such an approach are however not yet clear and have to be left for future work.

We already know from Lemma 4.3 that all vertices in the central inner neighbourhood are connected to the entire inner neighbourhood, which makes the central inner neighbourhood a clique.

Using Lemma 2.3 we can derive that both the left and right half of the inner neighbourhood also form a clique.

▶ **Lemma 4.20.** Let $u$ be a vertex in a hyperbolic random graph. The right and left half of the inner neighbourhood of $u$ each form a clique. ◀

*Proof.* We show the statement for the right half of the inner neighbourhood, as the other half follows analogously. Let $v$ and $w$ be two vertices in the right inner neighbourhood of $u$ such that $v$ without loss of generality lies between $u$ and $w$. As $u$ and $w$ are adjacent and $r(v) \leq r(u)$, $v$ and $w$ are also adjacent by Lemma 2.3. ■

This means that the inner neighbourhood of any vertex can be decomposed into (at most) two overlapping cliques: the union of the left inner neighbourhood and the central inner neighbourhood and the union of the right inner neighbourhood and the central inner neighbourhood. We can also show that for vertices with small enough radius neither the central inner neighbourhood nor the peripheral inner neighbourhood are empty with high probability. We first compute the measure of the central inner neighbourhood in the next lemma.

▶ **Lemma 4.21.** Let $v$ be a vertex and $I$ be the region of its central inner neighbourhood. Then the measure of $I$ is given as

$$\mu(I) = \frac{2}{\pi(2\alpha - 1)} e^{-R(\alpha-\frac{1}{2})-r(1-\alpha)} \cdot \left(1 \pm O\left(e^{-\alpha R}\right) - O\left(e^{-(\alpha-\frac{1}{2})(2r-R)}\right)\right). \quad (4.1)$$

◀

*Proof.* $I$ can be divided into an outer part with radius at least $R-r$ and an inner part with radius up to $R-r$, as depicted in Figure 4.4 (a). We compute the measure of $I$ by computing the measures of these parts separately, and adding them

$$\begin{aligned}\mu(I) =&\mu(B_0(R-r)) + 2\cdot\int_{R-r}^{r}\int_0^{\theta(r,x)-\theta(r,r)} f(x)\mathrm{d}\theta\mathrm{d}x\\ =&\underbrace{\mu(B_0(R-r))}_{\text{A}} +2\cdot\underbrace{\int_{R-r}^{r}\theta(r,x)f(x)\mathrm{d}x}_{\text{B}} -2\cdot\underbrace{\int_{R-r}^{r}\theta(r,r)f(x)\mathrm{d}x}_{\text{C}}.\end{aligned}$$

Summand A can be derived via Equation (2.2) as

$$\mu(B_0(R-r)) = e^{-\alpha r}(1-\Theta(e^{-\alpha r})).$$

Fur summand B we get

$$\begin{aligned}\int_{R-r}^{r} f(x)\theta(r,x)\mathrm{d}x &= \frac{\alpha}{\pi}\int_{R-r}^{r} e^{\alpha(x-R)+\frac{R-r-x}{2}}\cdot\left(1\pm\Theta\left(e^{R-r-x}\right)\right)\left(1+\Theta\left(e^{-\alpha R}-e^{-2\alpha x}\right)\right)\mathrm{d}x\\ &= \frac{\alpha}{\pi}\left(1+\Theta\left(e^{-\alpha R}\right)\right)\int_{R-r}^{r} e^{\alpha(x-R)+\frac{R-r-x}{2}}\cdot\left(1+O\left(\pm e^{R-r-x}-e^{-2\alpha x}\right)\right)\mathrm{d}x\end{aligned}$$

As $O\left(\pm e^{R-r-x}-e^{-2\alpha x}\right) = O\left(\pm e^{R-r-x}\right)$, this term can be transformed to

$$= \frac{\alpha}{\pi}\left(1+\Theta\left(e^{-\alpha R}\right)\right)\int_{R-r}^{r} e^{\alpha(x-R)+\frac{R-r-x}{2}}\cdot\left(1\pm O\left(e^{R-r-x}\right)\right)\mathrm{d}x.$$

We now compute the integral without the error term and later compute and add the error term.

$$\begin{aligned}&\frac{\alpha}{\pi}\left(1+\Theta\left(e^{-\alpha R}\right)\right)\int_{R-r}^{r} e^{\alpha(x-R)+\frac{R-r-x}{2}}\mathrm{d}x\\ =&\frac{\alpha}{\pi}\left(1+\Theta\left(e^{-\alpha R}\right)\right)\left[\frac{1}{\alpha-\frac{1}{2}}e^{x\left(\alpha-\frac{1}{2}\right)}\right]_{R-r}^{r} e^{-\frac{r}{2}-R\left(\alpha-\frac{1}{2}\right)}\\ =&\frac{\alpha}{\pi\left(\alpha-\frac{1}{2}\right)}\left(1+\Theta\left(e^{-\alpha R}\right)\right)\underbrace{\left(e^{r\left(\alpha-\frac{1}{2}\right)}-e^{R\left(\alpha-\frac{1}{2}\right)-r\left(\alpha-\frac{1}{2}\right)}\right)}_{e^{r\left(\alpha-\frac{1}{2}\right)}\cdot\left(1-e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right)} e^{-\frac{r}{2}-R\left(\alpha-\frac{1}{2}\right)}\end{aligned}$$

$$= \frac{\alpha}{\pi\left(\alpha - \frac{1}{2}\right)} \cdot \left(1 + \Theta\left(e^{-\alpha R}\right)\right) \cdot e^{-r(1-\alpha)-R\left(\alpha-\frac{1}{2}\right)} \cdot \left(1 - e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right)$$

For the error term we obtain

$$\begin{aligned}
&\int_{R-r}^{r} O\left(e^{x\left(\alpha-\frac{3}{2}\right)-\frac{3}{2}r-R\left(\alpha-\frac{3}{2}\right)}\right)\mathrm{d}x \\
=&O\left(\left[e^{x\left(\alpha-\frac{3}{2}\right)}\right]_{R-r}^{r} e^{-\frac{3}{2}r-R\left(\alpha-\frac{3}{2}\right)}\right) \\
=&O\left(e^{-r(3-\alpha)-R\left(\alpha-\frac{3}{2}\right)} - e^{-\alpha r}\right) \\
=&e^{-r(1-\alpha)-R\left(\alpha-\frac{1}{2}\right)} \cdot O\left(e^{-2r-R} - e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right).
\end{aligned}$$

Note that $-2r - R \leq -(2r - R) \leq -\left(\alpha - \frac{1}{2}\right)(2r - R)$, which simplifies the error term. Together with the rest we obtain

$$\int_{R-r}^{r} f(x)\theta(r,x)\mathrm{d}x = \frac{\alpha}{\pi} e^{-r(1-\alpha)-R\left(\alpha-\frac{1}{2}\right)} \cdot \left(1 + \Theta\left(e^{-\alpha R}\right)\right)\left(1 - O\left(e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right)\right).$$

For summand C we get

$$\begin{aligned}
&\int_{R-r}^{r} \theta(r,r) f(x)\mathrm{d}x \\
&= 2e^{\frac{R}{2}-r}\left(1 \pm \Theta\left(e^{R-2r}\right)\right) \cdot \frac{\alpha}{2\pi} e^{-\alpha R} \int_{R-r}^{r} e^{\alpha x}\left(1 + \Theta\left(e^{-\alpha R} - e^{-2\alpha x}\right)\right)\mathrm{d}x \\
&= \frac{\alpha}{\pi} e^{-R\left(\alpha-\frac{1}{2}\right)-r}\left(1 \pm \Theta\left(e^{-(2r-R)}\right)\right)\left(1 + \Theta\left(e^{-\alpha R}\right)\right) \int_{R-r}^{r} e^{\alpha x}\mathrm{d}x \\
&= \frac{1}{\pi} e^{-R\left(\alpha-\frac{1}{2}\right)-r}\left(1 \pm \Theta\left(e^{-(2r-R)}\right)\right)\left(1 + \Theta\left(e^{-\alpha R}\right)\right) \underbrace{\left[e^{\alpha r} - e^{\alpha(R-r)}\right]}_{e^{\alpha r}\left(1-e^{-\alpha(2r-R)}\right)} \\
&= \frac{1}{\pi} e^{-R\left(\alpha-\frac{1}{2}\right)-r(1-\alpha)}\left(1 \pm \Theta\left(e^{-(2r-R)}\right)\right)\left(1 - e^{-\alpha(2r-R)}\right)\left(1 + \Theta\left(e^{-\alpha R}\right)\right)
\end{aligned}$$

Where depending on $r$ either $\left(1 \pm \Theta\left(e^{-(2r-R)}\right)\right)$ or $\left(1 + \Theta\left(e^{-\alpha R}\right)\right)$ is the dominant error term.

Omitting summand A and putting together B and C, we obtain

$$\mu(S) = e^{-\alpha r}(1 - \Theta(e^{-\alpha r}))$$

$$
\begin{aligned}
&+ 2 \cdot \frac{2\alpha}{\pi\left(\alpha - \frac{1}{2}\right)} e^{-r(1-\alpha)-R\left(\alpha-\frac{1}{2}\right)} \cdot \left(1 + \Theta\left(e^{-\alpha R}\right)\right)\left(1 - O\left(e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right)\right) \\
&- 2 \cdot \frac{1}{\pi} e^{-R\left(\alpha-\frac{1}{2}\right)-r(1-\alpha)}\left(1 \pm \Theta\left(e^{-(2r-R)}\right)\right)\left(1 - e^{-\alpha(2r-R)}\right)\left(1 + O\left(e^{-\alpha R}\right)\right) \\
\geq & \frac{1}{\pi(2\alpha-1)} e^{-R\left(\alpha-\frac{1}{2}\right)-r(1-\alpha)} \\
&\cdot \left(0 + 2 \cdot \frac{\alpha\pi(2\alpha-1)}{\pi\left(\alpha-\frac{1}{2}\right)}\left(1 + \Theta\left(e^{-\alpha R}\right)\right)\left(1 - O\left(e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right)\right)\right. \\
&\left. - 2 \cdot \frac{\pi(2\alpha-1)}{\pi}\left(1 + O\left(e^{-\alpha R}\right)\right)\left(1 + \Theta\left(e^{-(2r-R)}\right)\right)\left(1 - e^{-\alpha(2r-R)}\right)\right) \\
= & \frac{2}{\pi(2\alpha-1)} e^{-R\left(\alpha-\frac{1}{2}\right)-r(1-\alpha)} \cdot \left(1 \pm O\left(e^{-\alpha R}\right) - O\left(e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)}\right)\right).
\end{aligned}
$$

■

▶ **Corollary 4.22.** Let $v$ be a vertex with radius $\frac{R}{2} < r \leq R - c - \frac{1}{1-\alpha}\log\log n$ for a constant $c = \frac{1}{1-\alpha}\log\left(\frac{\pi(2\alpha-1)}{e^{C/2}}\right)$. Then with high probability the left (respectively right) half of the central inner neighbourhood of $v$ contains at least one vertex. ◀

*Proof.* We use Lemma 4.21 as a lower bound for the measure of the left/right central inner neighbourhood which we benote by $I_{half}$. We have

$$
\Pr\left[\exists v \in I_{half}\right] = 1 - (1 - \mu(I_{half}))^n \geq 1 - \frac{1}{e^{n \cdot \mu(I_{half})}}.
$$

Where for $n \cdot \mu(I_{half})$ we obtain

$$
\begin{aligned}
n \cdot \mu(I_{half}) &= n \cdot \frac{1}{\pi(2\alpha-1)} e^{-R\left(\alpha-\frac{1}{2}\right)-r(1-\alpha)} \cdot (1 \pm O(e^{-\alpha R}) - O(e^{-\left(\alpha-\frac{1}{2}\right)(2r-R)})) \\
&= \frac{n}{\pi(2\alpha-1)} \cdot e^{-R\left(\alpha-\frac{1}{2}\right)-R(1-\alpha)+\left(c-\frac{1}{1-\alpha}\log\log n\right)(1-\alpha)} \cdot (1 \pm o(1)) \\
&= \frac{n}{\pi(2\alpha-1)} \cdot (e^{-(2\log n+C)/2+\left(c-\frac{1}{1-\alpha}\log\log n\right)(1-\alpha)}) \cdot (1 \pm o(1)) \\
&= \frac{1}{\pi(2\alpha-1)} \cdot e^{C/2} e^{\left(c-\frac{1}{1-\alpha}\log\log n\right)(1-\alpha)} \cdot (1 \pm o(1)) \\
&= \log n \cdot (1 \pm o(1))
\end{aligned}
$$

by the choice of $c = \frac{\log\left(\frac{\pi(2\alpha-1)}{e^{C/2}}\right)}{1-\alpha}$. This gives us $\Pr[\exists v \in X] = 1 - O\left(\frac{1}{n}\right)$. ■

Note that the restriction $r > \frac{R}{2}$ is not really a restriction, because in the other case the inner neighbourhood has diameter less than $R$ and forms a clique anyway. As the measure of the peripheral inner neighbourhood is also in $\Theta(e^{-R(\alpha-\frac{1}{2})-r(1-\alpha)})$ by Lemma 4.4, it is clear that the statement in the corollary above also applies for the peripheral inner neighbourhood.

In summary, we know a considerable collection of facts about the inner neighbourhood of a vertex. We know that the left and right inner neighbourhood and the central inner neighbourhood form cliques. Further, every vertex in the central inner neighbourhood is adjacent to every other vertex in the inner neighbourhood. From Section 4.4.1, we know that vertices with constant distance to the outer boundary of the disk are simplicial with at least constant probability. Moreover, for vertices with distance at least $\Omega(\log\log n)$ from the boundary of the disk the left and right peripheral and also central inner neighbourhood contain vertices with high probability.

Additionally, in the following lemma, we show that the treewidth of the central inner neighbourhood of a vertex is equal to its pathwidth.

▶ **Lemma 4.23.** Let $N^{<r}(v)$ be the inner neighbourhood of a vertex $v$. Then there is an optimal width tree decomposition of $G[N^{<r}(v)]$ that is a path decomposition and $\operatorname{tw}(G[N^{<r}(v)]) = \operatorname{pw}(G[N^{<r}(v)])$. ◀

*Proof.* Let $(T, \tau)$ be a optimal tree decomposition of $G[N^{<r}(v)]$. Assuming that $(T, \tau)$ is not a path decomposition, we show that it can be transformed into one. Let $L \subseteq\subset N^{<r}(v)$ denote the left half of the inner neighbourhood and $R$ the right half. As both $L$ and $R$ form a clique, there must be bags $\tau_\ell$ and $\tau_r$ that contain all vertices of $L$ and $R$, respectively. If $\tau_\ell$ and $\tau_r$ are the same bag, then this bag alone already is an optimum width tree decomposition that is a path decomposition. Otherwise, $\tau_\ell$ and $\tau_r$ lie on a path $P$ in the tree decomposition. As by assumption $(T, \tau)$ is no path decomposition, there are bags that are not part of $P$. Consider any bag $B$ that does not lie on $P$. There must be a path in the tree decomposition that connects $B$ to a vertex $B'$ on $P$. As any vertex in $B$ is either part of the left or right half of the inner neighbourhood, it must also be contained in $\tau_\ell$ or $\tau_r$. This means that $B \subseteq B'$ and therefore omitting $B$ (and all other bags in the subtree below it when rooting on $B'$) from the tree decomposition does not invalidate it. This way, an equally wide path decomposition of $N^{<r}(v)$ can be obtained. Thus,

$\mathrm{pw}(G[N^{<r}(v)]) \leq \mathrm{tw}(G[N^{<r}(v)])$ and as any path decomposition is already a tree decomposition $\mathrm{pw}(G[N^{<r}(v)]) = \mathrm{tw}(G[N^{<r}(v)])$. ■

This means that computing the treewidth of the inner neighbourhood of a vertex is just as hard as computing its pathwidth. However, the inner neighbourhood has a very particular structure, consisting of two distinct cliques that are connected by edges and contain a number of vertices that connect to every other vertex. This already tells us a lot about how optimal path/tree decompositions of the graph induced by the inner neighbourhood can look like. For example, they could start and end with bags that contain the left or respectively right clique together with all vertices that connect to the entire inner neighbourhood. In between those bags, there are bags in which vertices from the right clique are introduced and vertices from the left clique are forgotten. It is however unclear how exactly these bags in between can or have to look like depending on the structure of the graph.

A generalisation of the structure of the inner neighbourhood would be to view the graph induced by the inner neighbourhood as an arbitrary co-bipartite graph (i.e. the complement of a bipartite graph). Doing so gives a pessimistic outlook on understanding tree or path decompositions of the inner neighbourhood, as, by the proof for the NP-hardness of treewidth by Arnborg et al. [ACP87], even deciding the treewidth of co-bipartite graphs is NP-hard. This makes it appear unlikely that we can get a thorough understanding of path decompositions of the inner neighbourhood.

A different perspective, from which we can maybe learn more, consists of asking how the vertices of the inner neighbourhood of a vertex $v$ are distributed in the bags of optimal tree decompositions of the entire graph $G$. This is especially interesting in relation to the safeness of the inner neighbourhood of $v$. Clearly, there is an optimal tree decomposition of $G$ in which the inner neighbourhood of $v$ appears in a common bag, if and only if it is safe. One interesting question we can ask is therefore which properties a tree decomposition of $G$ has to have in order to allow or forbid rearranging it in a way that lets the inner neighbourhood appear in a single bag. This approach, together with other directions for future work, is discussed in the conclusion of this thesis.

# 5 Conclusions & Outlook

In this thesis, we described and analysed the PID-BT algorithm and its preprocessing, focussing on hyperbolic random graphs as a model of realistic inputs. This way, we make the algorithm more accessible to future improvements and contribute to the understanding of practical treewidth computation in general. We found that PID-BT without its preprocessing performs rather poorly, as shown experimentally and by a super-polynomial lower bound for the expected running time on HRGs. With the preprocessing, however, we observed vastly lower running times. We showed both empirically and theoretically that on hyperbolic random graphs the preprocessing can reduce the instance size by a constant fraction and that this is at least partially due to a linear number of simplicial vertices. We also introduced the concept of subtree-width and safeness of a separator with respect to a component. This allowed us to derive new sufficient conditions for safeness that, if a conjecture holds, apply to sub-linear components associated with the inner neighbourhoods of vertices with radius $R - \Theta(\log \log n)$ in a HRG.

There are many possible directions for future work. The most immediate ones are of course trying to improve the bounds for the number of feasible O-blocks and the number of simplicial vertices in hyperbolic random graphs. Feasible O-blocks are rather complex structures, and it is already a good first step to have any lower bound on them in hyperbolic random graphs. However, the current proof hides extremely small constants behind asymptotic notation, and it would be interesting to derive a tighter bound. Similarly, we hope that the proof for the number of (inner) simplicial vertices can be improved. As discussed at the end of Section 4.4.1, the current lower bound for the number of simplicial vertices depends on the parameters $\alpha$ and $C$ in a way that is not consistent to our empirical observations. Thus, there is probably another way in which vertices can be simplicial that dominates the one we considered. Investigating this might lead to an entirely different way of deriving a linear lower bound that might have greater explanatory capacity. Besides, while a linear number of simplicial vertices is enough to derive how the preprocessing is able reduce the instance size by a linear amount, this does not yet explain how the Min-Fill heuristic is able to construct almost optimal tree decompositions. This makes it even more worthwhile to investigate safeness more directly.

We want to point out one promising direction for this, building upon the insights from Section 4.4. The idea is to consider a separator $S$ consisting of the inner neighbourhood of some vertex $v$ of a HRG $G$ and how it is distributed within the bags of a tree decomposition of $G$. Given what we know about the structure of $S$, is it possible to find properties that allow or disallow for $S$ to be placed in a single bag? $S$ is safe if and only if there is an optimal tree decomposition of $G$ with a bag containing $S$, so this is a very interesting question. Intuitively it is clear that in many cases, $S$ can be safe even if it is not a clique. The goal is to either find a sufficient condition that allows the construction of a bag containing $S$ or a necessary condition that forbids $S$ to be placed in a single bag. Then, it remains to show that on HRGs the sufficient condition is likely to apply or that the necessary condition is unlikely to apply. One way to find such a condition could be to consider the bags in which $S$ appears. As $S$ consists of two overlapping cliques $S_\ell$ and $S_r$, any tree decomposition of $G$ contains a path $\mathcal{P}$ of adjacent bags, whose endpoints contain $S_\ell$ and $S_r$. Additionally, every bag on $\mathcal{P}$ contains the vertices of the central inner neighbourhood of $v$. Given this rich structure, what ways are there to manipulate the tree decomposition in order to construct a bag containing all vertices of $S$? Intuitively, it seems that if $v$ has small degree, as in for example (poly)logarithmic, then it should be unlikely that there are local structures or properties in the tree decomposition that prevent the safeness of $S$. Thus the problem probably lies in understanding how global properties of the graph or tree decomposition relate to $S$ and its safeness locally.

Next, Tamaki formulated the question of whether there is a way to bound the number of feasible O-blocks in terms of the number of feasible potential maximal cliques. In Lemma 3.8 we show that the number of whole O-blocks is at most $n$ times the number of feasible PMCs. It remains open, whether a similar bound for split O-blocks can be achieved.

This directly leads to the next open question: how can the PID-BT algorithm be improved? In our analysis in Section 3.5 we found that especially split O-blocks lead to exponential running times in cases in which this seems unnecessary. Maybe there is a way to replace split O-blocks by improved structures that restrict the search more strongly. Similarly, so far it was assumed that the algorithm exhaustively lists all feasible PMCs, I-blocks and O-blocks, even though in practice it would terminate at the first feasible PMC with an empty outlet. How pessimistic is this assumption? Are there cases in which it is asymptotically better to stop at the first PMC with empty outlet? Maybe it is possible to derive feasible PMCs in a more restricted order that terminates more quickly.

Another direction that we could only explore to a limited extent in this thesis is subtree-width. In this thesis, we only used subtree-width to derive conditions for the safeness of separators, but we argue that it would be interesting to investigate the concept more thoroughly. In their empirical study on the treewidth of real-world graphs Maniou et al. [MSJ19] report that similarly to hyperbolic random graphs, many real-world networks consist of a *densely connected core* and a *tree-like fringe*. They further find that this structure allows for low-width tree decompositions of considerably large sub-graphs. We conjecture that this means that there are also considerably large vertex sets that have a small subtree-width. It would be worthwhile to explore if there is a general way to exploit this algorithmically for different applications.

On a more abstract level, one could also ask whether treewidth is small enough in practice to be such a dominating concept in theory. One of the goals of parametrized complexity is to explain why certain instances of hard problems are sometimes efficiently solvable. Arguably, treewidth has only small explanatory capacity in this respect, and while hyperbolic random graphs and similar models seem to do better at explaining practical tractability, they are limited to very narrow domains. Maybe it is possible to identify better models or more realistic parameters in an attempt to bridge the gap between theory and practice.

# Bibliography

[ACP87] Stefan Arnborg, Derek G. Corneil, and Andrzej Proskurowski. **Complexity of Finding Embeddings in a $k$-Tree**. *SIAM. J. on Algebraic and Discrete Methods* (Apr. 1987), 277–284. DOI: 10.1137/0608024 (see pages 1, 2, 4, 18, 72).

[BB19] Max Bannach and Sebastian Berndt. “Positive-Instance Driven Dynamic Programming for Graph Searching.” In: *Algorithms and Data Structures*. Ed. by Zachary Friggstad, Jörg-Rüdiger Sack, and Mohammad R Salavatipour. Vol. 11646. Springer International Publishing, 2019, 43–56. DOI: 10.1007/978-3-030-24766-9_4 (see pages 5, 17, 18).

[BB72] Umberto Bertelè and Francesco Brioschi. **Nonserial dynamic programming**. Mathematics in science and engineering v. 91. New York: Academic Press, 1972. ISBN: 978-0-12-093450-8 (see page 3).

[BFK16] Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. **Hyperbolic Random Graphs: Separators and Treewidth**. In: *24th Annual European Symposium on Algorithms (ESA 2016)*. Vol. 57. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 15:1–15:16. DOI: 10.4230/LIPIcs.ESA.2016.15 (see pages 6, 40).

[BFK18] Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. **Cliques in Hyperbolic Random Graphs**. en. *Algorithmica* 80:8 (Aug. 2018), 2324–2344. DOI: 10.1007/s00453-017-0323-3 (see page 6).

[BFM13] Michel Bode, Nikolaos Fountoulakis, and Tobias Müller. “On the giant component of random hyperbolic graphs.” en. In: *The Seventh European Conference on Combinatorics, Graph Theory and Applications*. Pisa: Scuola Normale Superiore, 2013, 425–429 (see pages 6, 37).

[BK06] Hans L. Bodlaender and Arie M.C.A. Koster. **Safe separators for treewidth**. *Discrete Mathematics* 306:3 (Feb. 2006), 337–350. DOI: 10.1016/j.disc.2005.12.017 (see pages 5, 43, 46, 62, 63).

[BK07] H. L. Bodlaender and A. M. C. A. Koster. **Combinatorial Optimization on Graphs of Bounded Treewidth**. en. *The Computer Journal* 51:3 (Nov. 2007), 255–269. DOI: 10.1093/comjnl/bxm037 (see page 4).

[BKL17] Karl Bringmann, Ralph Keusch, and Johannes Lengler. **Sampling Geometric Inhomogeneous Random Graphs in Linear Time**. In: *25th Annual European Symposium on Algorithms (ESA 2017)*. Vol. 87. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 20:1–20:15. DOI: 10.4230/LIPIcs.ESA.2017.20 (see page 6).

[Blä+18] Thomas Bläsius, Cedric Freiberger, Tobias Friedrich, Maximilian Katzmann, Felix Montenegro-Retana, and Marianne Thieffry. **Efficient Shortest Paths in Scale-Free Networks with Underlying Hyperbolic Geometry**. In: *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Vol. 107. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 20:1–20:14. DOI: 10.4230/LIPIcs.ICALP.2018.20 (see pages 6, 38, 66).

[Blä+19] Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. **Efficiently Generating Geometric Inhomogeneous and Hyperbolic Random Graphs**. In: *27th Annual European Symposium on Algorithms (ESA 2019)*. Vol. 144. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 21:1–21:14. DOI: 10.4230/LIPIcs.ESA.2019.21 (see pages 6, 13).

[Blä+20] Thomas Bläsius, Philipp Fischbeck, Tobias Friedrich, and Maximilian Katzmann. **Solving Vertex Cover in Polynomial Time on Hyperbolic Random Graphs**. In: *37th International Symposium on Theoretical Aspects of Computer Science (STACS 2020)*. Vol. 154. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, 25:1–25:14. DOI: 10.4230/LIPIcs.STACS.2020.25 (see page 6).

[Bod+01] Hans L. Bodlaender, Arie M.C.A. Koster, Frank van den Eijkhof, and Linda C. van der Gaag. **Pre-processing for Triangulation of Probabilistic Networks**. eng. Tech. rep. 01-39. URN: urn:nbn:de:0297-zib-6655. Takustr. 7, 14195 Berlin: Zuse Institute Berlin (ZIB), 2001 (see page 46).

[Bod+06] Hans L. Bodlaender, Fedor V. Fomin, Arie M. C. A. Koster, Dieter Kratsch, and Dimitrios M. Thilikos. **On Exact Algorithms for Treewidth**. In: *Algorithms – ESA 2006*. Ed. by Yossi Azar and Thomas Erlebach. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, 672–683. DOI: 10.1007/11841036_60 (see page 5).

[Bod05] Hans L. Bodlaender. **Discovering Treewidth**. In: *SOFSEM 2005: Theory and Practice of Computer Science*. Ed. by Peter Vojtáš, Mária Bieliková, Bernadette Charron-Bost, and Ondrej Sýkora. Vol. 3381. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, 1–16. DOI: 10.1007/978-3-540-30577-4_1 (see pages 5, 44).

[Bod93] Hans L. Bodlaender. **A Linear Time Algorithm for Finding Tree-Decompositions of Small Treewidth**. In: *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*. STOC '93. San Diego, California, USA: Association for Computing Machinery, 1993, 226–234. DOI: 10.1145/167088.167161 (see page 4).

[Bod98] Hans L. Bodlaender. **A partial k-arboretum of graphs with bounded treewidth**. en. *Theoretical Computer Science* 209:1-2 (Dec. 1998), 1–45. DOI: 10.1016/S0304-3975(97)00228-4 (see page 3).

[BPK10] Marián Boguñá, Fragkiskos Papadopoulos, and Dmitri Krioukov. **Sustaining the Internet with hyperbolic mapping**. *Nature Communications* 1:62 (Dec. 2010), 8. DOI: 10.1038/ncomms1063 (see page 5).

[Bri+17] Karl Bringmann, Ralph Keusch, Johannes Lengler, Yannic Maus, and Anisur Rahaman Molla. **Greedy Routing and the Algorithmic Small-World Phenomenon**. en. In: *Proceedings of the ACM Symposium on Principles of Distributed Computing*. Washington DC USA: ACM, July 2017, 371–380. DOI: 10.1145/3087801.3087829 (see page 6).

[BT02] Vincent Bouchitté and Ioan Todinca. **Listing all potential maximal cliques of a graph**. en. *Theoretical Computer Science* 276:1-2 (Apr. 2002), 17–32. DOI: 10.1016/S0304-3975(01)00007-X (see pages 2, 4, 22).

[CC16] Chandra Chekuri and Julia Chuzhoy. **Polynomial Bounds for the Grid-Minor Theorem**. en. *J. ACM* 63:5 (Dec. 2016), 1–65. DOI: 10.1145/2820609 (see page 4).

[Cou90] Bruno Courcelle. **The monadic second-order logic of graphs. I. Recognizable sets of finite graphs**. en. *Information and Computation* 85:1 (Mar. 1990), 12–75. DOI: 10.1016/0890-5401(90)90043-H (see page 1).

[Del+17] Holger Dell, Thore Husfeldt, Bart M. P. Jansen, Petteri Kaski, Christian Komusiewicz, and Frances A. Rosamond. **The First Parameterized Algorithms and Computational Experiments Challenge**. In: *11th International Symposium on Parameterized and Exact Computation (IPEC 2016)*. Vol. 63. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 30:1–30:9. DOI: 10.4230/LIPIcs.IPEC.2016.30 (see page 5).

[Del+18] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. **The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration**. In: *12th International Symposium on Parameterized and Exact Computation (IPEC 2017)*. Vol. 89. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 30:1–30:12. DOI: 10.4230/LIPIcs.IPEC.2017.30 (see page 5).

[DF99] R. G. Downey and M. R. Fellows. **Parameterized Complexity**. Ed. by David Gries and Fred B. Schneider. Monographs in Computer Science. New York, NY: Springer New York, 1999. ISBN: 978-1-4612-6798-0 978-1-4612-0515-9. DOI: 10.1007/978-1-4612-0515-9 (see page 4).

[DHS06] Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. **Computing treewidth with LibTW**. Tech. rep. University of Utrecht, 2006. URL: http://www.treewidth.com/treewidth/ (see page 5).

[FK15] Tobias Friedrich and Anton Krohmer. **On the Diameter of Hyperbolic Random Graphs**. In: *Automata, Languages, and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, 614–625. DOI: 10.1007/978-3-662-47666-6_49 (see page 6).

[GPP12] Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. **Random Hyperbolic Graphs: Degree Sequence and Clustering**. In: *Automata, Languages, and Programming*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, 573–585. DOI: 10.1007/978-3-642-31585-5_51 (see pages 6, 10–12).

[Hal76] Rudolf Halin. **S-functions for graphs**. *Journal of Geometry* 8:1-2 (Mar. 1976), 171–186. DOI: 10.1007/BF01917434 (see page 3).

[Kri+10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. **Hyperbolic geometry of complex networks**. *Phys. Rev. E* 82:3 (Sept. 2010), 036106. DOI: 10.1103/PhysRevE.82.036106 (see page 5).

[Kro17] Anton Kromer. **Structures & Algorithms in Hyperbolic Random Graphs**. PhD thesis. University of Potsdam, May 2017. URL: https://publishup.uni-potsdam.de/opus4-ubp/files/39597/krohmer_diss.pdf (see pages 11, 12).

[MS19] Tobias Müller and Merlijn Staps. **The diameter of KPKVB random graphs**. *Adv. Appl. Probab.* 51:2 (June 2019), 358–377. DOI: 10.1017/apr.2019.23 (see page 6).

[MSJ19] Silviu Maniu, Pierre Senellart, and Suraj Jog. **An Experimental Study of the Treewidth of Real-World Graph Data**. In: *22nd International Conference on Database Theory (ICDT 2019)*. Vol. 127. LIPIcs. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 12:1–12:18. DOI: 10.4230/LIPIcs.ICDT.2019.12 (see pages 4, 5, 75).

[PS97] Andreas Parra and Petra Scheffler. **Characterizations and algorithmic applications of chordal graph embeddings**. *Discrete Applied Mathematics* 79:1-3 (Nov. 1997), 171–188. DOI: 10.1016/S0166-218X(97)00041-3 (see pages 3, 9).

[PW19] Manuel Penschuck and Christopher Weyand. *girgs*. https://github.com/chistopher/girgs/. 2019 (see page 13).

[Röh98] Hein Röhrig. **Tree decomposition: A feasibility study**. English. Diplomarbeit. Saarbrücken: Max-Planck-Institut für Informatik in Saarbrücken, Sept. 1998. URL: http://edoc.mpg.de/518016 (visited on 08/06/2020) (see page 5).

[RS86] Neil Robertson and P.D Seymour. **Graph minors. II. Algorithmic aspects of tree-width**. en. *Journal of Algorithms* 7:3 (Sept. 1986), 309–322. DOI: 10.1016/0196-6774(86)90023-4 (see pages 3, 4).

[RS95] N. Robertson and P.D. Seymour. **Graph Minors .XIII. The Disjoint Paths Problem**. en. *Journal of Combinatorial Theory, Series B* 63:1 (Jan. 1995), 65–110. DOI: 10.1006/jctb.1995.1006 (see page 4).

[RST94] N. Robertson, P. Seymour, and R. Thomas. **Quickly Excluding a Planar Graph**. en. *Journal of Combinatorial Theory, Series B* 62:2 (Nov. 1994), 323–348. DOI: 10.1006/jctb.1994.1073 (see page 4).

[ST93] P.D. Seymour and R. Thomas. **Graph Searching and a Min-Max Theorem for Tree-Width**. en. *Journal of Combinatorial Theory, Series B* 58:1 (May 1993), 22–33. DOI: 10.1006/jctb.1993.1027 (see pages 3, 18).

[Tam19] Hisao Tamaki. **Positive-instance driven dynamic programming for treewidth**. en. *Journal of Combinatorial Optimization* 37:4 (May 2019), 1283–1311. DOI: 10.1007/s10878-018-0353-z (see pages 5, 23–28, 31, 46).

[Tho98] Mikkel Thorup. **All Structured Programs Have Small Tree Width and Good Register Allocation**. *Information and Computation* 142:2 (May 1998), 159–181. DOI: 10.1006/inco.1997.2697 (see page 4).

[TO17] Hisao Tamaki and Hiromu Ohtsuka. *PACE2017-TrackA*. https://github.com/TCS-Meiji/PACE2017-TrackA. 2017 (see page 13).

# Declaration of Authorship

I hereby declare that this thesis is my own unaided work. All direct or indirect sources used are acknowledged as references.

Potsdam, August 24, 2020

Marcus Wilhelm