Deterministic Properties for Algorithm Efficiency: Distances, Separators, and Cliques

Zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften

von der KIT-Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Marcus Herbert Wilhelm

Tag der mündlichen Prüfung:	19. Juni 2026
1. Referent:	T.T.-Prof. Dr. Thomas Bläsius
2. Referent:	Prof. Dr. Sebastian Siebertz

Abstract

Classical worst-case analysis of algorithms derives strong theoretical guarantees by assuming minimal constraints on the problem instance and considering any instance of a given size. Unfortunately, such running time bounds are often pessimistic and do not carry over to practice. For example, many problems appearing in practical applications are NP-hard, which makes the existence of algorithms that efficiently solve every possible instance unlikely. Despite this, practically occurring instances can often be solved efficiently.

Several methods have been developed to resolve this discrepancy between benign real-world instances and pessimistic worst-case bounds, each having individual limitations. Average-case analysis highly depends on the assumed probability distribution and the plausibility of the distribution's connection to real-world instances. Furthermore, average-case analysis only makes statements about the chosen probability distribution and not about individual problem instances. Parameterized analysis requires small, sufficiently powerful, and ideally efficiently computable parameters. For many practically relevant problems at least one of these three criteria fails. Additionally, it can be challenging to express the structure and complexity of practical instances with a single numerical parameter.

In this thesis we propose a complementary approach for the analysis of algorithms using deterministic structural properties, which subsume parameters but extend to richer combinations of parameters and other structural assumptions. We demonstrate the usefulness of this framework through multiple case studies considering different problems and algorithms, across both hard and efficiently solvable settings. We find that deterministic properties are a broadly applicable tool that addresses disadvantages of both average-case and parameterized analysis. Deterministic properties can be more flexible than univariate numerical parameters, while giving more tangible results than average-case analysis. More concretely, we find that our approach is useful to explain empirically observed algorithm efficiency, to develop new algorithms for specific types of instances, and to complement previously established parameterized as well as probabilistic input models. Overall, deterministic properties are a pragmatic approach that offers both theoretical insights and practical algorithmic improvements, thus contributing towards bridging the gap between the theoretical analysis of algorithms and practical algorithm performance.

Acknowledgments

Shortly after I had first moved to Karlsruhe to start as a PhD student, when my neighbor inquired about my job and how I liked it, I answered that it feels somewhat unreal because I simply continued doing what I enjoyed as a master student, while suddenly getting paid (more than for my previous student teaching assistant roles). To this, my neighbor's amused reply was that, from her experience, doing a PhD necessarily also involves at least some hardship. Looking back, it is clear that both views contain more than just a grain of truth. For this reason, I want to express my sincere gratitude both for the incredible opportunity to embark on this journey at all and for all the support I got from so many different sides.

First of all, I want to thank Thomas Bläsius for taking me in as his first PhD student at KIT, for his outstanding support and assistance as a supervisor and mentor, and for probably being the best possible scientific role model. I also want to thank Sebastian Siebertz for agreeing to review my thesis and for helping to realize the somewhat ambitiously scheduled defense date. Moreover, I am grateful to Martin Taraz, Wendy Yi, John Theurer, and Thomas Bläsius for providing valuable and constructive feedback on early drafts of this thesis and to various contributors including Martin Krejca for the thesis template.

Especially after the COVID pandemic, I carried out most of my work in the office, where I very much enjoyed a nice and friendly atmosphere that made it easy to stay excited about research and teaching. For this, I thank all my colleagues and seniors, in particular Thomas Bläsius, Christopher Weyand, Adrian Feilhauer, Maximilian Katzmann, Max Göttlicher, Michael Zündorf, Wendy Yi, Jean-Pierre von der Heydt, Martin Schirneck, Elly Schmidt, Dorothea Wagner, Michael Hamann, Lars Gottesbüren, Tim Zeitz, Matthias Wolf, Sascha Gritzbach, Jonas Sauer, Torsten Ueckerdt, Paul Jungeblut, Laura Merker, Miriam Goetze, Samuel Schneider, Kolja Kühn, Marvin Künnemann-Goos, Mirza Redzic, Geri Gokaj, Julian Stieß, and Sebastian Angrick. I especially appreciated your curiosity and willingness to engage in both serious and silly discussions, our joint attempts at making sushi and other food, and various other fun activities such as lifting weights, hiking, and bouldering. Special thanks also go to the lunchbox crew for sharing meals in a quieter environment (and for carrying my box, i.e., *Kiste*, all those times when I worked from home for too long to be there on time).

Furthermore, I thank Sándor Kisfaludi-Bak for inviting Thomas, Jean-Pierre, and

me to Aalto and, together with Geert van Wordragen, contributing to some of the most fun and engaging weeks of my doctoral studies. Similarly, I want to thank all my co-authors and collaborators for the nice work we did together. I also want to thank all the students I worked with, in particular Dennis Kobert, Annemarie Schaub, and Sven Geißler, whose theses and projects grew (or are growing) into joint papers. Besides my academic colleagues, I want to thank Ralf Kölmel, Tanja Wehrmann, Isabelle Junge, and Jennifer Mustapic for their support on technical and administrative matters, allowing me to focus on teaching and research.

My time at KIT would have been so much harder without the support I got from outside of university. I thank Alex, Benjamin, Hauke, Johannes, Stefan, Tassilo, and Titus, for helping me find friendship and community in a new city where I knew no one (especially during the COVID lockdowns) and for countless fascinating discussions on topics far beyond theoretical computer science. Moreover, I thank Malte, Michael, Sarah, Toni, Wendy, and all participants of KIT's unicycling course for giving me a place to find balance away from work, helping me practice my skills both in the gym and on the trails, and for encouraging me to join our performances. I am also grateful for all support from previously unmentioned friends, my family and especially my parents, without whom none of this would have been possible. Last but very much not least, I thank Sing-Huei for being the most important discovery of my doctoral studies.

Contents

1 Introduction

A vast number of tasks and questions relevant in everyday life can be formulated as computational problems. This gives computer science a tangible connection to the real world, where actual computers are routinely used to solve such problems. With this connection, theoretical computer science, the study of the mathematical foundations of computation, takes a similar place as theoretical physics, combining aspects of pure mathematics with aspects of natural science [Den07; DF13; Weg76]. From this perspective, one should expect progress in theoretical computer science to not only come in the form of proved theorems and deeper understanding of abstract computational structures. Rather, just like theoretical physics produces models that aim to better describe physical phenomena, theoretical computer science ought to produce better models of computational phenomena in the real world. A natural context in which to evaluate such models is the question of how efficiently a given problem can be solved. This touches upon core concepts within theoretical computer science such as (asymptotic) running times, the analysis of algorithms, and the (time-)complexity of problems.

The classical approach to these topics is motivated as follows. For the analysis of an algorithm, it is desirable to be independent of specific implementation details or the underlying hardware. Hence, for *asymptotic running times*, one studies the scaling behavior of algorithms depending on the size of inputs while ignoring constant factor differences using big-O notation. With this perspective, considering an algorithm on inputs of constant size becomes meaningless, so it is necessary to look at infinite families of inputs. Typically, when studying the performance of an algorithm on such an infinite family of inputs, one tries to use as few further assumptions as possible, bounding the running time only as a function of the input size. The resulting *worst-case analysis* thus yields very strong mathematical guarantees on the running time that are valid for *every* conceivable input. This framework forms the basis for the theoretical analysis of algorithms and also for computational complexity theory.

For many algorithms and problems, asymptotic running times derived via worst-case analysis are helpful tools that work reasonably well for the three main goals of algorithm analysis as identified by Roughgarden [Rou21]: predicting performance, comparing approaches, and guiding the development of new algorithms. However, like any model, worst-case analysis also has limitations. Consider for instance

the theory of NP-completeness. Following seminal results by Cook, Levin and Karp [Coo71; Kar72; Lev73] it was found that a large class of so-called NP-hard problems are unlikely to admit correct algorithms that solve every conceivable instance in polynomial time. This might seem discouraging, as many practically and economically relevant combinatorial optimization problems are NP-hard [GJ79]. However, it turns out that in many cases, typical real-world instances for these problems are much easier to solve than comparatively adversarial worst-case instances. Even though boolean satisfiability is perhaps the single most prominent NP-hard problem, modern SAT solving algorithms can routinely solve instances with millions of variables and clauses within reasonable time [GV21]. Similarly, some NP-hard graph problems can be solved rather efficiently on many real-world instances, see, e.g., [Del+18; DFH19]. Even among comparatively easy problems that can be solved in polynomial time, there are many examples for algorithms whose running time is highly instance-dependent and thus not reliably predicted by worst-case analysis [BCT17; BFW21; Blo+08]. It follows that typical instances from practice differ from worst-case instances in relevant ways that significantly affect the running times of algorithms [BF24], and worst-case analysis is thus not always a suitable model for predicting practical performance.

Consequently, in order to obtain more predictive models for the practical efficiency of algorithms, it is necessary to leverage additional assumptions about their inputs. Doing so means modeling the characteristics of typical inputs and tailoring the analysis of running times to take these characteristics into account. The two most prominent approaches attempting this are average-case analysis and parameterized analysis.[1] While both have proven fruitful, each comes with individual limitations. In this thesis, we argue for a complementary perspective based on deterministic properties, which we introduce after discussing these two approaches in more detail.

1.1 Average Case Analysis

As highlighted above, worst-case analysis fails to predict real-world performance of algorithms that are substantially slower on some inputs than on typical inputs in practice. The central idea behind *average-case analysis* is to model such typical inputs as random samples from an underlying probability distribution. Then, the algorithm's running time is a random variable with an expected value and concen-

1 Here, we focus on approaches that substantially aim to address the gap between theory and practice. For a broader collection of approaches going beyond the worst-case analysis of algorithms, see the book by Roughgarden [Rou21].

tration properties that can be analyzed asymptotically. Naturally, this approach crucially depends on the chosen probability distribution. Ideally, there should be reason to believe that this distribution accurately depicts the kinds of inputs that are encountered in practice. At the same time the distribution needs to be mathematically tractable, which often results in a conflict between richness allowing for empirical grounding and simplicity that enables rigorous analysis.

In this context, we also want to mention *smoothed analysis*, which considers worst-case instances under small random perturbations. One could thus see it as an alternative to average-case analysis requiring fewer assumptions, while still being able to explain the practically observed efficiency of algorithms, such as for instance, the simplex algorithm [ST04]. Essentially, smoothed analysis interpolates between worst-case and average-case analysis. This makes smoothed analysis most useful for cases where the average case is easy and worst-case instances are fragile outliers, while otherwise being less informative. Apart from that, like average-case analysis, smoothed analysis also heavily depends on the chosen random distribution and its plausibility.

The challenge of choosing a distribution that is both sufficiently tractable and empirically grounded is particularly apparent in the context of graph algorithms, where the choice of random model has been a subject of debate since the early days of average-case analysis. Historically, the default choice has been the Erdős-Rényi model [ER59; ER60; Gil59], where each edge exists independently with a fixed probability.[2] The simplicity of this model affords tractability [Bol01a], but fails to describe graphs that are meaningfully similar to typical real-world networks. As Karp observed already in 1983 [Kar83] and later confirmed in 2020, results obtained under such simplistic distributional assumptions tend to lack *external validity*, i.e., the insights derived from the analysis do not reliably transfer to practical instances [BF24].

One constructive response to this originates from a conversation between (theoretical) computer science, physics, and so-called network science, which has systematically studied structural properties of real-world networks and developed random network models that reflect them more faithfully. Roughly speaking, network science posits that there exists a coherent class of so-called *complex (real-world) networks* (also *scale-free* networks). These are sparse graphs that share three main characteristics: *locality*, *heterogeneity*, and the *small-world property*. Locality (sometimes also called: *clustering* or *community structure*) refers to non-random

2 Strictly speaking, this model was introduced by Gilbert [Gil59], while Erdős and Rényi's original model [ER59; ER60] fixes the number of edges. However, both variants are commonly referred to as the Erdős–Rényi model.

structural dependencies between vertices. For example, connected vertices are more likely to have common neighbors than unconnected vertices. Heterogeneity describes that complex networks often contain few vertices with disproportionately high degree, commonly modeled as *power-law distributed* vertex degrees. The small-world property states that most vertices have very low distance, typically formalized as a (sub-)logarithmic diameter or average distance between vertices. While there are exceptions and also other classes of networks (e.g., road networks), an abundance of real-world networks from different domains follow these patterns. Examples include social networks (e.g., citation networks, social media interactions), biological networks (e.g., neuronal connections, protein interactions), and technological networks (e.g., internet infrastructure, hyperlinks) [New03].

The Erdős-Rényi model indeed exhibits the small-world property [Bol01b], but lacks both clustering and heterogeneity. This motivated the proposal of random network models that capture these properties better. Models like the *Watts-Strogatz model* [WS98] and the *stochastic block model* [HLL83] have clustering and small diameters, but still result in homogeneous vertex degrees. The *preferential-attachment model* [BA99] conversely produces networks with a power-law degree distribution [Bol+01] and logarithmic diameter [DHH10], but without clustering. The same applies to *inhomogeneous random graphs*, *Chung-Lu random graphs*, and the *(soft) configuration model* [BJR07; CL02a; Hof24]. While these models and were already widely studied (see for example, [BJR07; Hof24]), it remained an open question whether the three hallmark properties of complex real-world networks could be achieved in a convincing and mathematically tractable model. While a number of proposals tried to address this, by, e.g., tweaking the preferential-attachment model or replacing individual vertices with small cliques [HK02; KE02; RB03], such approaches seem contrived and result in structural artifacts. In contrast, *Hyperbolic random graphs* (HRGs) are the first prominent network model that combines all three properties in a natural and mathematically elegant way [Kri+10]. The model is conceptually similar to *geometric random graphs* [Pen03], where vertices are represented by unit disks that are distributed uniformly at random in some ground space and edges are formed between each intersecting pair of disks. This directly results in networks with high clustering, but does not lead to a heterogeneous degree distribution and small distances. HRGs are essentially a (rather specific) translation of geometric random graphs into the hyperbolic plane. By carefully choosing the radius of the disks representing each vertex and the ground-space into which they are placed, clustering, heterogeneous vertex degrees and small distances emerge naturally via inherent properties of the hyperbolic plane [FK18; GPP12; MS19]. Interestingly, a very similar network model can be defined by combining geometric random graphs with vertex weights akin to the Chung-Lu model. The

resulting *geometric inhomogeneous random graphs* (GIRGs) can roughly be seen as generalizing HRGs [BKL19].

Encouragingly, more realistic models like GIRGs have been empirically validated as good proxies for real-world networks, not only with respect to structural properties but with respect to algorithm performance. Bläsius and Fischbeck [BF24] conducted a case study comparing algorithm performance on real-world networks and synthetic networks across multiple algorithmic problems. They identified locality and heterogeneity of networks as two dimensions that influence algorithm performance both on real-world networks as well as the tested network models. This provides evidence for the external validity of GIRGs, where locality and heterogeneity can be controlled using two parameters. Restricted to networks with low locality, this also extends to Chung-Lu random graphs. Moreover, thanks to an efficient generator [Blä+22c], GIRGs are also well-suited for empirical study and have been established as a viable testing ground for algorithm engineering [Wey23].

There are also several theoretical results that attempt to explain the real-world performance of algorithms using average-case analysis on HRGs. For instance, it has been shown that HRGs allow both NP-hard problems as well as problems in P to be solved faster than on general graphs [BFK18; Blä+22b; Blä+23a]. Compared to previous work studying Erdős-Rényi graphs (see, e.g., [BN19; Bol01a]) results on HRGs are often more convincing, especially when the arguments involved rely on empirically observed properties of real-world networks [Kat23]. In one case, the study of approximation algorithms for an NP-hard problem on HRGs has even led to the discovery of improvements to an existing algorithm for real-world networks [BFK23].

Overall, we find that average-case analysis using models for real-world inputs is a helpful tool to explain the practical tractability of both easy and hard problems. However, the probabilistic perspective also has inherent limitations: the choice of distribution is difficult to justify as each additional assumption about the input requires independent motivation, and the assumption of randomness itself is a strong one. Moreover, even using well-motivated models such as HRGs and GIRGs one can only make statements about the random distribution of typical instances and not about specific inputs. This makes it hard to transfer insights to individual real-world instances. Average-case analysis does not make predictions about any individual input, and it is often not clear whether the empirically observed performance of an algorithm is due to the same structural reason as in the theoretical analysis. There is also a risk of tailoring algorithms or analyses to quirks of the chosen distribution leading to results that do not generalize much beyond the studied model. These limitations motivate the search for approaches that do not rely on probabilistic input assumptions, such as the ones discussed next.

1.2 Parameterized Analysis

A theoretician's natural response upon finding that a computational problem is NP-hard is to consider its complexity on instances with restricted structural properties. This is often fruitful and has led to a wealth of results, especially in combination with combinatorics and graph theory [Lov05; LT79; RS04]. Such endeavors, however, arguably tend to be more about uncovering theoretical insights than about closing the gap between theory and practice in algorithm analysis. The most systematic attempt to bridge this gap using structural properties of inputs is *parameterized complexity* [DF13]. Downey and Fellows went as far as formulating this as an "aspiration and opportunity to engage mathematical science in algorithm design for real datasets with useful outcomes" [DF13]. The idea is to not only consider the asymptotic running time of an algorithm with respect to the size of the input, but also with respect to other parameters that describe relevant properties of the input.

Formally, a parameterized problem has instances (x, k) consisting of the actual input x and a parameter k. Then, a problem is *fixed-parameter tractable* (FPT) if it can be solved in time $f(k) \cdot n^c$, where n is the input size, k is the parameter, f is an arbitrary computable function, and c is an independent constant. This definition is useful especially for problems where no polynomial time algorithms are known, as it confines all super-polynomial complexity to depend only on the parameter instead of the input size. In fact, for constant k, the asymptotic complexity does not even depend on k. Beyond the study of hard problems, parameterized analysis can also be applied to polynomial time solvable problems [AVW16; GMN17].

Two examples for common parameters are the size of the solution or structural properties of the input. For example, vertex cover and feedback vertex set are NP-hard in general but become FPT when parameterized by solution size [Cyg+15; DF13]. Structural parameters such as graph width measures are also widely studied. For example, many hard graph problems are FPT with respect to treewidth [Cou90], a parameter measuring the similarity of a given graph to a tree.

In their textbook on parameterized algorithms, Cygan, Fomin, Kowalik, Lokshtanov, Marx, Pilipczuk, Pilipczuk, and Saurabh [Cyg+15] state that for a successful parameterization it is not only necessary that the studied problem admits an FPT algorithm with respect to some parameter, but additionally, one should have reason to believe that this parameter is typically small on relevant instances. It is unfortunate that the second question receives far less attention compared to the first, not only in the remainder of their book but also within the overall literature. This is a problem because many widely studied parameters are often rather large in practice. For instance, an experimental study found that the treewidth of real-world

graphs is rarely below $\sqrt{n}$, which makes most algorithms with a 2^k running time infeasible [MSJ19].

Even within the research community's own testing ground, the gap between theory and practice persists. The PACE challenge[3] is a yearly competition where participants implement algorithms that are evaluated and ranked based on their performance. While this competition has been successful in producing efficient implementations and even led to the development of new algorithms, the techniques used in the best solvers are often strikingly different from the ones that work well in theory. For example, even though one of the leading solvers for the 2019 PACE exact competition is based on interesting and deep combinatorial insights also described in a subsequent publication [Tam17], its good performance is mostly due to the use of effective heuristics in the pre-processing [Wil20]. Similarly, while the winning implementation of the 2021 PACE challenge [Blä+21] used a branching rule from the FPT literature [Böc+09; Böc12], its good performance is mostly due to the surprising effectiveness of reduction rules and heuristic upper and lower bounds [Blä+22a]. From a theoretical perspective, it is almost entirely opaque why these heuristics work so well.

In conclusion, the field of parameterized algorithms has produced valuable insights into why certain problems are hard, but its success in closing the gap between theoretical analysis and practical performance is limited. One possible reason for this could be that in many cases it is simply difficult to find a *univariate* parameterization that provably encapsulates the difficulty of instances and also fits well to practical algorithms and data.

1.3 Deterministic Properties for Algorithm Analysis

A major advantage of the structural perspective discussed above is that combinatorially defined parameters and graph classes can at least in principle be verified empirically. This is opposed to average-case analysis, where, due to their probabilistic nature, assumptions are fundamentally difficult to verify, especially on individual instances. This suggests a natural direction: searching for more empirically grounded structural assumptions as models for the efficiency of algorithms in practice. It is generally difficult to find assumptions that are strong enough to have provable algorithmic consequences, while still fitting plausibly to practically observed inputs. To soften this trade-off, we argue that this search should not be limited to single numerical parameters, but also allow multiple parameters and

3 https://pacechallenge.org

more complex assumptions in general. Coming from the perspective of computing as a natural science, theory ought to provide predictive models for practical algorithm efficiency.

This thesis pursues exactly this direction: we seek deterministic structural assumptions that model what makes practical instances easy, allowing for richer descriptions than a single numerical parameter, while retaining per-instance guarantees that average-case analysis cannot provide. We use the term *deterministic properties* for such assumptions, including parameters, other deterministic structural assumptions, and combinations thereof. Note that this subsumes (univariate) parameterized analysis, but shifts the emphasis from complexity-theoretic aspects towards predictive modeling of practical performance. Our perspective follows related proposals such as *axiomatic analysis* [BCT17] and *distribution-free* models [Fox+20; Rou21]. The general idea has also been expressed within the context of parameterized complexity. Both the aspiration of using parameterized analysis to accurately predict practical algorithm performance and the use of more than one parameter were already proposed by Downey and Fellows [DF13] who also explicitly champion the term *multivariate algorithmics*. While our views regarding this naming are held lightly, our choice avoids the word "axiom" which carries connotations more related to the foundations of mathematics. It also frames the non-dependence on a probabilistic model more affirmatively than "distribution-free" does, while not strictly demanding multivariate parameterizations.

We briefly discuss the three most prominent previous examples that use deterministic properties to explain the surprising algorithmic tractability of practical instances. Interestingly, each of these can be seen as addressing one of three hallmark properties of complex networks.

Brach, Cygan, Łącki, and Sankowski define a somewhat involved deterministic condition for *power-law bounded* graphs [Bra+16] and show that such graphs allow multiple problems including maximum matching, triangle counting, and maximum clique to be solved faster than under worst-case assumptions. They complement these theoretical results with experiments in which they verify their definition of power-law boundedness on large heterogeneous real-world graphs. They also explicitly discuss their approach as an alternative to average-case analysis. While this certainly represents a valuable contribution, one critique is that the definition of power-law bounded graphs gives more of an upper bound for the heterogeneity (and also density) and does not seem to robustly capture the concept of heterogeneity as it is observed in practice. Indeed, any graph with constant maximum degree satisfies their definition, regardless of its actual degree distribution.

Borassi, Crescenzi, and Trevisan [BCT17] formulate a set of deterministic properties describing the metric structure of the graph, somewhat related to the small-

world phenomenon observed in complex real-world networks. They use these properties to analyze multiple practically relevant algorithms and heuristics for computing the graph diameter and a few other problems. The assumptions are validated both empirically and with an analysis on random graph models such as Chung-Lu random graphs. One downside of the properties and their analysis is that they essentially hinge on a property of random graphs with independently sampled edges, somewhat contradicting the commonly observed clustering properties of complex networks. To be a bit more concrete, the authors in [BCT17] roughly formalize a birthday paradox argument, where in an Erdős-Rényi graph two vertex sets of size n^x are likely to be connected with an edge for $x \geq 0.5$, but not for smaller x. Consequently, it is not clear whether the properties hold on, for example, graphs with underlying geometry.

Fox, Roughgarden, Seshadhri, Wei, and Wein introduce *(weak) c-closure* as a parameter that deterministically models networks with clustering [Fox+20], where vertices incident to the same edge are much more likely to have common neighbors than a random pair of vertices. They show the NP-hard problem of finding the maximum clique is in FPT with respect to this parameter, by bounding the number of inclusion-maximal cliques with respect to the parameter. Leveraging a polynomial time algorithm that determines the parameter on a given graph, they additionally conduct experiments showing that the parameter is often small on real-world graphs. While this is a highly compelling example for a successful parameterization, Bläsius and Fischbeck [BF24] empirically found that on many real-world networks, the parameter is largely uncorrelated with the number of maximal cliques, which is often drastically lower than predicted based on the parameterized analysis.

Even though the above works demonstrate the potential of using deterministic properties for the analysis of algorithms, they each only apply the idea to a specific setting or problem. Roughgarden and Seshadhri [RS21] go further and survey them alongside some related results as "distribution-free models for social networks". However, none of these treat deterministic properties as a distinct method for algorithm analysis in general or explicitly situate this perspective relative to both average-case and parameterized analysis.

1.4 Contribution and Outline

In this thesis, we establish deterministic properties as a general methodology for the analysis of algorithms. We demonstrate its usefulness and applicability through four case studies that each address downsides of previous parameterized or average-case approaches, and argue for its potential to close the gap between theoretical

analysis and practical algorithm performance. Our contributions are organized across four chapters as follows.

Clique-Partitioned Treewidth. Chapter 3 stays within the setting of univariate parameterized complexity and addresses the problem that in practice the treewidth of many graphs is prohibitively large [MSJ19]. Concretely, it proposes the new parameter *clique-partitioned treewidth* as a variant of treewidth with the goal of achieving smaller parameter values in practice. The new parameter adapts a technique previously used on geometric intersection graphs [Ber+20; Kis20], where instead of measuring the size of separators, one considers their structure via a weighted clique partition. The chapter shows that the parameter is still algorithmically useful and presents multiple exact and heuristic ways of computing such partitions and evaluates them on a large set of networks. These experiments show that the new parameter can indeed be notably smaller than treewidth in practice, especially on networks with a high clustering coefficient.

Overall, the chapter contributes to pushing parameterized analysis in a more empirically grounded direction and shows the feasibility of such endeavors.

Hyperbolic Uniform Disk Graphs and Independent Set. Next, in Chapter 4 we give a deterministic perspective on an established probabilistic model. Specifically, we study *hyperbolic uniform disk graphs* (HUDGs), a deterministic generalization of hyperbolic random graphs. Parameterized by a radius r, these graphs resemble Euclidean disk intersection graphs for small (subconstant) radii, but have very different structure for larger values of r. The chapter examines one algorithmic implication of these structural properties by describing an algorithm for the NP-hard Independent Set problem that runs in polynomial time for $r \in \Omega(\log n)$. As this regime also corresponds to hyperbolic random graphs, this adds a valuable new perspective to previous results showing polynomial-time average-case complexity on HRGs [Blä+23a].

This way, the chapter shows how deterministic properties provide a way to de-randomize a previously used random network model. Doing so, it exposes fundamental properties and deeper algorithmic implications that are already inherent to the probabilistic model.

On the Practical Efficiency of Bidirectional BFS. Chapter 5 uses deterministic properties to explain the performance of the bidirectional breadth-first search (BFS) algorithm on real-world networks, an algorithm used to find a

shortest path between two vertices in a graph. Here, the *bidirectional* variant has been observed to sometimes yield an asymptotic speed-up over the traditional *unidirectional* BFS, i.e., on some networks the bidirectional BFS outperforms the unidirectional BFS by a factor that grows with the network size. This chapter contrasts previous probabilistic explanations for this phenomenon with an analysis leveraging deterministic properties. In particular, it formulates a set of parameters capturing the exponential expansion of BFS search spaces, gives theoretical conditions for provably sublinear running times based on those parameters, and empirically validates the theory on a large set of real-world networks. Specifically, the conditions are often fulfilled in practice, and their fulfillment is correlated with faster running times.

The chapter shows deterministic properties as a method giving convincing explanations for the practical performance of algorithms. The empirical evaluation goes beyond those of previous studies [BCT17; Bra+16; Fox+20] by not only confirming that the assumptions used for the running time analysis often hold on real-world networks, but by also showing that they are predictive, in the sense that networks satisfying the assumptions actually exhibit faster running times.

Diameter Computation in (Random) Geometric Graphs. Chapter 6 considers the problem of computing the graph *diameter*, i.e., the largest distance between any two vertices. Even though conditional lower bounds rule out algorithms with subquadratic worst-case running time, the problem can often be solved much faster on many real-world networks. One exception to this practical tractability occurs on graphs with a toroidal or spherical geometric structure [BF24]. To address this, we present a framework consisting of deterministic properties and an algorithm that exploits them, thus enabling more efficient diameter computation. The properties robustly formalize combinatorial features of graphs with underlying geometry and also extend to torus-like geometry. We verify these properties on (torus-like) random geometric graphs and thus show that our algorithm achieves subquadratic running times. This is the first average-case analysis of the diameter problem on geometric random graphs and also beats recent results for the more general class of unit-disk graphs across a wide parameter regime [Cha+25].

These results demonstrate the usefulness of deterministic properties for the design of new algorithms. The use of deterministic properties makes transparent which structural properties of the input are algorithmically exploited, a level of insight that a traditional average-case analysis does not provide.

Overall, the four chapters demonstrate different facets of deterministic properties as a methodology for algorithm analysis: pushing parameterized analysis in a more empirically grounded direction, replacing probabilistic assumptions with deterministic ones, and providing more transparent explanations of practical algorithm performance. Together, the chapters show that pragmatically allowing more flexible and complex deterministic assumptions, rather than insisting on a single parameter or a probabilistic model, is a fruitful direction with largely unexplored potential for closing the gap between theoretical analysis and practical algorithm performance.

The remainder of the thesis is organized as follows. Chapter 2 introduces notation and definitions used throughout, while Chapters 3–6 cover our technical contributions across the four case studies described above. Finally, Chapter 7 concludes with a broader discussion and directions for future work.

2 Preliminaries

We introduce basic definitions and notational conventions used throughout this thesis. We write $\mathbb{N}$ for the *natural numbers*, i.e., the set of positive integers and $\mathbb{N}_0$ for the set of non-negative integers. Further, we write $[n] = \{1, \dots, n\}$ for the set of first n natural numbers. For a set X we write $2^X = \{X' \mid X' \subseteq X\}$ for the *powerset* of X and $\binom{X}{k} = \{X' \mid |X'| = k, X' \subseteq X\}$ for the set of k-element subsets of X.

We assume 2 to be the default base of logarithms, i.e., we write $\log_2 x$ as $\log x$. For the natural logarithm, we use $\ln x = \log_e x$.

In Chapters 5 and 6 we are interested in polynomial differences between running times, e.g., $O(n^{1-\varepsilon})$ vs. $O(n)$ (for constant $\varepsilon > 0$), and thus use $\tilde{O}$-notation to suppress poly-logarithmic factors.

We write $\Pr[\mathcal{E}]$ for the *probability* of an *event* $\mathcal{E}$, and $\mathrm{E}[X]$ for the *expected value* of a random variable X. In the context of a graph with n vertices, we say an event occurs *asymptotically almost surely* (a.a.s.) if its probability is at least $1 - o(1)$ and *with high probability* if its probability is at least $1 - O(1/n)$.

Graphs and treewidth. A *graph* $G = (V, E)$ consists of a (finite) set of *vertices* V and *edges* $E \subseteq \binom{V}{2}$, i.e., we typically consider graphs to be simple and undirected. If the graph in question is clear, we often use n and m to refer to the number of vertices and edges, respectively. We also use $V(G)$ and $E(G)$ to refer to the vertex and edge set of G. A graph H with $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$ is a *subgraph* of G. The subgraph H is *induced* by $V(H)$ if for each $u, v \in V(H)$ with $\{u, v\} \in E(G)$ we also have $\{u, v\} \in E(H)$. For a set of vertices $X \subseteq V(G)$ we also write $G[X]$ to refer to the subgraph of G induced by X, i.e., $G[X]$ is an induced subgraph of G with $V(G[X]) = X$. Two vertices are *adjacent* if they appear together in the same edge and an edge is *incident* to its vertices. The *degree* of a vertex is the number of incident edges.

A *cycle* is a connected graph in which every vertex has degree 2. A *complete graph* is a graph where all vertices are pairwise adjacent, i.e., it contains $\binom{n}{2}$ edges. A set of vertices $X \subseteq V$ is called *clique* if it induces a complete subgraph.

A (simple) *path* from v to w in G is a non-repeating sequence of vertices $\langle v_1, \dots, v_k \rangle$ with $v_1 = v$ and $v_k = w$ where each subsequent pair is adjacent. The *length* of such a path is $k - 1$. The *(hop-)distance* between two vertices $v, w \in V(G)$, denoted by $d_G(v, w)$, is the minimum length of a path from v to w. If there is no path between v

and w, then $d_G(v,w) = \infty$. Otherwise, v and w are *connected*. A set of vertices, and by extension a (sub-)graph, is *connected*, if every pair of vertices is connected. A *connected component* is a connected vertex set that is inclusion-wise maximal with respect to being connected.

The *(open) k-neighborhood* of a vertex v is the set of vertices with distance exactly k from v. We write $N_G(v,k) = \{w \in V(G) \mid d_G(v,w) = k\}$ for the k-neighborhood of v. The *closed k-neighborhood* of a vertex v, written as $N_G[v,k]$, is the set of vertices with distance at most k from v. The 1-neighborhood of v is also simply called *neighborhood* of v, written as $N_G(v)$. We omit the subscript G from the notation for distance, neighborhoods, and similar concepts if the graph is clear from context.

A graph is called *acyclic* if no subgraph is a cycle. A connected acyclic graph is called a *tree*. Equivalently, a connected or acyclic graph is a tree if it has $n-1$ edges, and a graph is a tree if there is exactly one path between every pair of vertices. A *rooted tree* is a tree T together with a designated *root* $r \in V(T)$. We call $v \in V(T)$ a *descendant* of $w \in V(T)$ if w lies on the unique path from v to the root r. In this case, we call w an *ancestor* of v. If v and w are adjacent, w is the *parent* of v, written as $\text{parent}(v) = w$, and v is a *child* of w. A *leaf* is a vertex without children. For $v \in V(T)$ we define the *subtree below v* as the subgraph induced by v and the descendants of v.

A vertex set $S \subseteq V$ is a *separator* if V can be partitioned into non-empty subsets S, V_1, and V_2 such that there is no edge between V_1 and V_2. The separator is *β-balanced* if $|V_1|, |V_2| \leq \beta n$ for some $\beta < 1$.

A *tree decomposition* of G is a pair (T, B), for a tree T and a function B mapping vertices of T to subsets of V called *bags*, with the following three properties: (1) every vertex v of G is contained in some bag, i.e., there is a *node* $t \in V(T)$ with $v \in B(t)$, (2) for every edge $e = \{u,v\} \in E(G)$ of G, there is a bag containing both endpoints, i.e., there is a node $t \in V(T)$ with $\{u,v\} \in B(t)$, and (3) for any vertex v of G, the set of bags containing v forms a *subtree* of T, i.e., for $t_v = \{t \in V(T) \mid v \in B(t)\}$ the subgraph $T[t_v]$ is connected. The *width* of a tree decomposition is the size of the largest bag minus 1. The *treewidth* $\text{tw}(G)$ is the smallest width of any tree decomposition of G.

3 Partitioning the Bags of a Tree Decomposition Into Cliques

The following chapter is based on joint work with Thomas Bläsius and Maximilian Katzmann [BKW23]. The project was motivated by the use of clique-based separators on geometric intersection graphs [Ber+20; Kis20] and the question of whether such techniques could be applied on real-world networks.

Beyond minor cosmetic improvements, this thesis extends the conference version with a strengthened hardness result (Theorem 3.7) and a comparison of our new parameter with treewidth (Theorem 3.2). The idea for the latter result originated in a bachelor's thesis by Sven Geißler [Gei23].

In this chapter we introduce a deterministic property capturing structural simplicity in graphs with large cliques. The idea is that graphs with large cliques inevitably have large treewidth, even though their separators might be covered with few cliques. Concretely, we introduce *clique-partitioned treewidth* as a variant of treewidth. This new parameter is closely related to so-called $\mathcal{P}$-flattened treewidth, but has multiple advantages. We take a first step towards making this parameter practically usable by studying an underlying optimization problem related to clique-partitioning and evaluating heuristic approaches on real-world networks.

3.1 Introduction

Recall that the *treewidth*, a measure for how tree-like a graph is in terms of its separators, is defined via a *tree decomposition*, i.e., a collection of vertex separators called *bags* that are arranged in a tree structure. The size of the largest bag determines the *width* of the decomposition and the treewidth of a graph is the minimum width over all tree decompositions.

While treewidth has its origins in graph theory and is connected to deep structural insights [Lov05; RS90], it also has important algorithmic implications. Intuitively speaking, the separators of a tree decomposition split the graph into pieces that can be solved independently except for minor dependencies at the separators. This can be exploited using dynamic programming over the tree decomposition, and often yields algorithm that are FPT with respect to the treewidth as a parameter [Cyg+15].

As this is a versatile framework that can be applied to many problems, it comes to no surprise that there has been quite a bit of effort to develop algorithms for computing low-width tree decompositions (see, e.g., [Del+17; Del+18]).

A major obstruction for low treewidth are large cliques, which inevitably lead to large separators. This is particularly true for so-called complex networks, i.e., graphs with strong community structure and heterogeneous degree distribution, which appear in various domains such as communication networks, social networks, or web-graphs. One could, however, hope for two aspects that together mitigate this negative effect of large cliques. First, even though some separators are large, they might be structurally simple, e.g., by forming a clique or allowing coverage with few cliques. Second, separators that are large but structurally simple still let us solve the separated pieces individually with low dependence between them. The first hope is supported by structural results about hyperbolic random graphs [BFK16] and related graph classes [Blä+25; Blä+26; Kis20], as also discussed in Chapter 4. This indicates that cliques might indeed be a major obstruction for low treewidth in many kinds of networks. The second hope is supported by results of De Berg, Bodlaender, Kisfaludi-Bak, Marx, and Van der Zanden [Ber+20], who introduced the concept $\mathcal{P}$-flattened tree decompositions. There, the graph is partitioned into cliques and the width of the tree decomposition is measured in terms of the (weighted) number of cliques in a bag. Thus, the width does measure the complexity of separators rather than their size. Based on this definition, the authors then show that these structurally simple separators help to solve various problems more efficiently on geometric intersection graphs.

To the best of our knowledge, these ideas have not yet been studied from a practical perspective. In this chapter, we want to initiate this line of research by addressing two questions. First, can such clique-partitioned tree decompositions lead to substantially smaller parameter values than classical tree decompositions? Second, how can such tree decompositions be computed? For the second question, we design and evaluate different algorithmic strategies for computing a novel yet closely related variant of tree decompositions. Our experiments yield some interesting algorithmic insights and provide a good starting point for further development. Especially on networks with strong clustering, the constructed tree decompositions indeed have sufficiently low weight clique-partitions, to answer the first question affirmatively.

We believe that there is plenty of room for improvement in our approaches, which may yield even better insights into the applicability of the new parameter or variants thereof. In the following, we discuss related work before stating our contribution more precisely.

3.1.1 Related Work

There are multiple lines of research that investigate variants of treewidth where additional structural properties of separators are taken into account. As mentioned above, the authors in [Ber+20] propose a variant of tree decompositions where the initial graph is partitioned into cliques (or unions of constantly many connected cliques) that are contracted into weighted vertices, where the weight of a clique of size s is $\log(s+1)$. They then consider the weighted treewidth of the resulting weighted graph, where the weight of a bag is given by the sum of the vertex weights. Using this technique, the authors give subexponential algorithms for a range of problems on geometric intersection graphs, including Independent Set, Steiner Tree and Feedback Vertex Set. Kisfaludi-Bak [Kis20] applied the same algorithmic framework to intersection graphs of constantly sized objects in the hyperbolic plane, see also Chapter 4 for additional discussion.

A similar parameter called *tree clique width* has been proposed by Aronis [Aro21]. Here, the idea is to consider tree decompositions where each bag is annotated with an edge clique cover and where the size of the cover determines the width of a bag. The paper shows several hardness results and adapts common treewidth algorithms to the newly proposed parameter.

Another approach to capture graph structures that lead to high treewidth despite being structurally simple has been proposed by Dallard, Milanič, and Štorgel. They define the *independence number* of a tree decomposition as the size of the largest independent set of any of its bags and the *tree-independence number* of a graph as the minimum independence number of any tree decomposition [DMŠ24a]. This parameter connects to the more theoretical study of (tw, ω)-bounded graphs, i.e., graph classes in which the treewidth depends only on the clique number [DMŠ21; DMŠ24b]. This line of research is mostly concerned with the classification and characterization of the considered graph classes both in terms of graph theory and algorithmic exploitability. However, apart from a factor 8 approximation for the tree-independence number with running time $2^{O(k^2)} \cdot n^{O(k)}$ due to Dallard, Fomin, Golovach, Korhonen, and Milanič [Dal+26], we are not aware of any work that tries to actually design or even implement algorithms for this or related parameters.

3.1.2 Contribution & Chapter Outline

In this chapter, we propose *clique-partitioned* treewidth as a parameter that captures structurally simple separators in graphs. It can be seen as a fractional and more local variant of $\mathcal{P}$-flattened treewidth [Ber+20], where we *first* compute a tree decomposition and *then* determine clique partitions of the subgraphs induced by

the bags. Thus, instead of using a global clique partition of the whole graph, we consider clique partitions that are local to a single bag.

The remainder of this chapter is structured as follows. In Section 3.2, we formalize our definition for clique-partitioned treewidth, demonstrate its algorithmic usefulness and compare it with $\mathcal{P}$-flattened treewidth. In Section 3.3, we present multiple approaches to compute low-weight clique partitions for the bags of an already computed tree decomposition. These include heuristic methods, as well as an exact branch-and-bound algorithm for which we propose several adjustments with the potential to improve its running time in practice. Afterwards, in Section 3.4 we combine an implementation of our approaches with existing methods for computing tree decompositions and study the upper bounds on the clique-partitioned treewidth of real-world networks. Furthermore, we evaluate the performance of the exact and heuristic clique partition solvers proposed in Section 3.3.

3.2 Clique-partitioned treewidth

In the following we define our parameter, compare it with two related parameters. Specifically, we show that clique-partitioned treewidth is upper bounded by the parameter introduced in [Ber+20], while being exponentially smaller on some instances. We additionally relate cp-treewidth to treewidth, showing that the two parameters are within an exponential factor of each other, which implies that FPT results transfer between them. Finally, we demonstrate that for independent set, the exponential blowup can be avoided entirely (Lemma 3.3).

Recall the definition of tree decompositions from Chapter 2. We define a *clique-partitioned* tree decomposition of a graph G as a tree decomposition (T, B) where for every $t \in V(T)$ we have a partition $\mathcal{P}_t$ of the subgraph induced by the corresponding bag (i.e., the graph $G[B(t)]$) into cliques. Following the authors in [Ber+20], we define the *weight* of a clique C as $\log(|C| + 1)$ and the weight of a bag $B(t)$ as the sum of weights of the cliques in its partition $\mathcal{P}_t$, i.e., $\sum_{C \in \mathcal{P}_t} \log(|C| + 1)$. The weight of a clique-partitioned tree decomposition is the maximum weight of any of its bags and the *clique-partitioned treewidth* (short: cp-treewidth) of G, denoted by $\mathrm{cptw}(G)$, is the minimum weight of any clique-partitioned tree decomposition.

As mentioned before, the clique-partitioned treewidth is closely related to the parameter defined in [Ber+20]. For a clique partition $\mathcal{P}$ of G, we say that a *$\mathcal{P}$-flattened tree decomposition* is a tree decomposition of the weighted graph obtained by contracting each clique C of $\mathcal{P}$ into a vertex v_C with weight $\log(|C| + 1)$. The weight of such a tree decomposition is then given by the sum of vertex weights for

each bag. In reference to the authors [Ber+20], we call the minimum weight over all clique partitions $\mathcal{P}$ and $\mathcal{P}$-flattened tree decompositions the BBKMZ-treewidth.

Roughly speaking, BBKMZ-treewidth and cp-treewidth rely on the same two central building blocks: a tree decomposition and a weighted clique partitioning. The difference is the order in which they are applied: while BBKMZ-treewidth first considers a partitioning and then a tree decomposition of the contracted graph, cp-treewidth first considers a tree decomposition and then a partitioning of each bag. There are two main advantages of cp-treewidth over BBKMZ-treewidth. First, with cp-treewidth the problem of finding a low weight clique partition is more independent from the problem of finding a tree decomposition. This makes it easy to develop heuristic upper bounds for the parameter, see Section 3.3. Second, cp-treewidth can be seen as a fractional and local variant of BBKMZ-treewidth, in the sense that a global clique partition $\mathcal{P}$ also induces local clique partitions for each bag, but BBKMZ-treewidth enforces that cliques of $\mathcal{P}$ are either fully contained in a bag or not, while cp-treewidth allows them to be split across multiple bags. Thus, the cp-treewidth of a graph is at most its BBKMZ-treewidth. Additionally, the cp-treewidth can also be substantially smaller than the BBKMZ-treewidth, as shown in the following lemma.

▷ **Lemma 3.1.** There is an infinite family of graphs $\mathcal{G}$ where each $G \in \mathcal{G}$ with n vertices has clique-partitioned treewidth in $O(\log \log n)$ and BBKMZ-treewidth in $\Omega(\log n)$. ◁

Proof. The family $\mathcal{G}$ contains for every $h \in \mathbb{N}$ one graph G_h. The graph G_h is a complete binary tree of height h, where additionally for every leaf ℓ we connect all h vertices that lie on a path between the root r and ℓ into a clique. Note that we have $h \in \Theta(\log n)$.

Let $\mathcal{P}_h$ be a clique partition of G_h. Then, via a simple induction over h, it is easy to see that in G_h there is a path between the root r and some leaf ℓ of G_h such that every vertex on the path belongs to a different partition class. Thus after contracting each partition, these vertices still form a path of length h. As they additionally form a clique, they are present in some bag of any $\mathcal{P}_h$-flattened tree decomposition of G_h (see also [Bod98, Lemma 4]). This bag thus contains all h partition classes on the path and has weight at least $h \cdot \log(1 + 1) \in \Omega(\log n)$.

At the same time we can construct a clique-partitioned tree decomposition (T, σ), that has one bag for every path between the root r and each leaf ℓ. Then, T forms a path. As every bag consists of a single clique on h vertices, there is a clique partition of this tree decomposition with weighted width $\log(h + 1) \in O(\log \log n)$. ■

In addition to showing that cp-treewidth is smaller than related parameters, we

show that it remains useful by enabling fixed-parameter tractability of NP-hard problems. To this end, we compare cp-treewidth with treewidth, obtaining both a lower and an upper bound.

▷ **Theorem 3.2 (See also [Gei23]).** For any graph G we have $\mathrm{cptw}(G) - 1 \leq \mathrm{tw}(G) \leq 2^{\mathrm{cptw}(G)}$. ◁

Proof. For the first inequality consider a bag X of a tree decomposition of G with width w. A trivial partitioning of X into cliques of size 1 has weight $|X| \cdot \log(1+1) = |X|$. As the width of a tree decomposition is the size of the largest bag minus 1, we have $|X| \leq w + 1$. Thus G has a clique-partitioned tree decomposition with weight at most $\mathrm{tw}(G) + 1$.

For the second inequality we consider the largest bag X of a minimum weight clique-partitioned tree decomposition of G. Then, X has a clique-partition $\mathcal{P}_X$ with weight at most $\mathrm{cptw}(G)$. We derive a bound on $|X|$ as follows. First, we have

$$|X| = \sum_{C \in \mathcal{P}_X} |C| \leq \sum_{C \in \mathcal{P}_X} (|C| + 1).$$

Now, for two integers x, y larger than 1, we have $x + y \leq x \cdot y$, so we get

$$\begin{aligned} |X| &\leq \prod_{C \in \mathcal{P}_X} (|C| + 1), \\ &\leq 2^{\log\left(\prod_{C \in \mathcal{P}_X} (|C|+1)\right)} = 2^{\left(\sum_{C \in \mathcal{P}_X} \log(|C|+1)\right)}. \end{aligned}$$

Here, the exponent is exactly the weight of the clique partition $\mathcal{P}_X$, which is at most $\mathrm{cptw}(G)$. Consequently,

$$|X| \leq 2^{\mathrm{cptw}(G)}.$$

As X is the largest bag of a tree decomposition, we have $\mathrm{tw}(G) \leq |X| - 1 \leq 2^{\mathrm{cptw}(G)}$. ■

Thus, an algorithm's running time of $O(f(\mathrm{tw}(G)) \cdot n^c)$ can also be expressed as $O(f(2^{\mathrm{cptw}(G)}) \cdot n^c)$, while a running time in $O(g(\mathrm{cptw}(G)) \cdot n^c)$ is also in $O(g(\mathrm{tw}(G)) \cdot n^c)$. Informally speaking, this means that a graph problem is FPT with respect to treewidth if and only if it is FPT with respect to cp-treewidth.

For some problems, the exponential increase of the running time's dependency on the parameter can be avoided. To show this, we consider the independent set problem that asks to find a subset of mutually non-adjacent vertices in a graph.

The following result, originally shown for BBKMZ-treewidth in [Ber+20], carries over to cp-treewidth.

▷ **Lemma 3.3.** Let G be a graph with a clique-partitioned tree decomposition (T, σ) of weight τ. Then a maximum independent set of G can be found in $O(2^\tau \cdot \text{poly}(n))$ time. ◁

Proof. We use a standard dynamic programming approach on tree decompositions based on *introduce*, *forget*, and *join* nodes (see for example [Cyg+15]). For each node $t \in V(T)$, we store a number of partial solutions for the subgraph of G induced by the bags of nodes in the subtree below t.

A partial solution consists of a subset of the vertices in the current bag as well as the size of the total partial independent set for the subgraph induced by the subtree below the current bag. This makes it easy to initialize partial solutions for leaf nodes in the tree decomposition.

In an introduce node, two new partial solutions are created, one where the new vertex is in the independent set and one where it is not. In a forget node, the removed vertex is removed from each partial solution. In a join node, compatible partial solutions from the child-nodes are combined by taking their union.

In a traditional tree decomposition of width k, this leads to at most 2^k partial solutions per bag. In a clique-partitioned tree decomposition, this is even smaller, as there are only $k + 1$ ways an independent set can intersect a clique of size k. Thus, assuming $\{\mathcal{P}_t \mid t \in V(T)\}$ denotes the clique partition of weight τ, the number of partial solutions that need to be considered per bag t are at most

$$\prod_{C \in \mathcal{P}_t} (|C| + 1) = 2^{\sum_{C \in \mathcal{P}_t} \log(|C|+1)} = 2^\tau.$$

As the number of bags as well as the time spent per partial solution are both polynomial, this concludes the proof. ■

3.3 The weighted clique partition problem

Having established that cp-treewidth is a useful parameter, we now turn to the question of computing it. In this section we give a heuristic upper bound as follows. We split the task of computing a clique-partitioned tree decomposition in two phases. First, we compute a tree decomposition, minimizing the traditional tree width. Secondly, fixing the structure and bags of this decomposition, we compute a clique partition for every bag. We note that this separation is already heuristic

as optimal solutions for the two subproblems do not guarantee an optimal clique-partitioned tree decomposition. However, we expect that small bags should also allow for low-weight clique partitions.

In the first phase, we use established algorithms for the computation of tree decompositions. Consequently, we focus on the second step in this section. To this end, we define the Weighted Clique Partition problem, short Clique Partition. For a given graph G and an integer w, decide if there is a partition of $V(G)$ into cliques $P_1, \ldots, P_k$ such that $\prod_{i\in[k]}(|P_i| + 1) \leq w$. Note that this function differs from the one in the definition of clique-partitioned treewidth, but is equivalent, as $\sum_{i\in[k]} \log(|P_i| + 1) = \log(\prod_{1\leq i\leq k}(|P_i| + 1))$ and the logarithm is monotonic.

In the following, we prove some technical lemmas that are useful throughout the section, before showing that Weighted Clique Partition is NP-complete (Section 3.3.1). Afterwards, we give different heuristic approaches (Section 3.3.2) and an optimal branch-and-bound algorithm in (Section 3.3.3). We start with following lemma, which intuitively states that the weight of a partition is smaller the more imbalanced the individual weights are, i.e., moving a vertex from a smaller to a larger clique reduces the total weight.

▷ **Lemma 3.4.** Let $a, b, c, d \in \mathbb{N}_0$ such that $a + b = c + d$ and $a \geq b$, $c \geq d$, $d > b$. Then $(a + 1)(b + 1) < (c + 1)(d + 1)$. ◁

Proof. There is an $x > 0$ such that $c = a - x$ and $d = b + x$. As $c \geq d$, x can be at most $(a - b)/2$. We derive

$$\begin{aligned}(c + 1)(d + 1) &= (a - x + 1)(b + x + 1)\\ &= ab - bx + b + ax - x^2 + x + a - x + 1\\ &= (ab + a + b + 1) + ax - bx - x^2\\ &= (a + 1)(b + 1) + x(a - b - x).\end{aligned}$$

We have $x(a - b - x) > 0$, as $0 < x \leq \frac{a-b}{2}$ and thus the claimed strict inequality follows. ■

With the above lemma (i.e., repeated applications thereof) we can compare the weight of two partitions.

▷ **Lemma 3.5.** Let $\langle s_1, \ldots, s_k\rangle$ and $\langle r_1, \ldots, r_\ell\rangle$ be two different non-increasing sequences of natural numbers such that $2 \leq k \leq \ell$, $\sum_{i\in[k]} s_i = \sum_{i\in[\ell]} r_i$, and $s_i \geq r_i$ for all $i \in [k - 1]$. Then $\prod_{i\in[k]}(s_i + 1) < \prod_{i\in[\ell]}(r_i + 1)$. ◁

Proof. This follows from repeatedly applying Lemma 3.4 to go from $R = \langle r_1, \ldots, r_\ell\rangle$ to $S = \langle s_1, \ldots, s_k\rangle$ while reducing the product in each step. To make this precise

let i be the first index where $s_i > r_i$. We adjust R by adding 1 to r_i and subtracting 1 from r_ℓ. Note that this maintains the sum. We apply Lemma 3.4 with $a = r_i + 1$, $b = r_\ell - 1$, $c = r_i$, and $d = r_\ell$. Then, we have $(a+1)(b+1) < (c+1)(d+1)$, i.e., the product of the adjusted sequence is smaller than that of the original sequence R. Moreover, after a finite number of steps, we reach S and thus the product for S is smaller than the product for R. ■

In the remainder of this section, we show that Weighted Clique Partition is NP-complete and further give different heuristic and exact ways of solving the optimization variant of Weighted Clique Partition.

For our algorithmic approaches, we make use of the fact that enumerating all (inclusion-)maximal cliques of a graph is not only output polynomial [JPY88], but also highly feasible in practice as shown by Eppstein, Löffler, and Strash [ELS13]. We use an implementation of their algorithm from the `igraph`[4] library. We also address this in our hardness proof, by showing that the problem remains hard even on graphs with few maximal cliques.

3.3.1 Hardness

We prove that Weighted Clique Partition is NP-complete and in particular remains NP-hard even on graphs with a polynomial number of maximal cliques. For this we reduce from Exact Cover by 3-Sets (X3C) as defined by Garey and Johnson [GJ79]: given a set X of $|X| = 3q$ elements and a collection C of 3-element subsets of X, decide if there is an exact cover, i.e., a subcollection $C' \subseteq C$ such that every $x \in X$ is contained in exactly one member of C'. By Garey and Johnson, this problem remains NP-hard even if no element $x \in X$ occurs in more than three subsets.

The rough idea for our reduction is to construct a graph with a vertex for every element of X and a sufficiently large clique for each element of C. By extending each of these large cliques to the respective 3-element subsets of the X3C-instance, we create a correspondence between exact covers and low-weight clique partitions.

We first show an inequality that later helps us exclude wayward clique partitions that do not correspond to exact covers.

▷ **Lemma 3.6.** For positive integers $k, c \in \mathbb{N}$ we have $(k+4)^c < (k+1)^c \cdot 2$ if $k > \frac{3c}{\ln 2} \approx 4.33c$. ◁

4 https://igraph.org/

Proof. From the above inequality we derive

$$\begin{aligned} k+4 &< (k+1)\cdot 2^{1/c} \\ k+4 &< k2^{1/c} + 2^{1/c} \\ k(1-2^{1/c}) &< 2^{1/c} - 4 \\ k &> \frac{4-2^{1/c}}{2^{1/c}-1}. \end{aligned}$$

It remains to show that $\frac{4-2^{1/c}}{2^{1/c}-1} < \frac{3c}{\ln 2}$ holds for $c > 0$. To that end, we set $t := \frac{\ln 2}{c} > 0$, then $2^{1/c} = e^t$, and so

$$\frac{4-2^{1/c}}{2^{1/c}-1} = \frac{4-e^t}{e^t-1} = \frac{3+1-e^t}{e^t-1} = \frac{3}{e^t-1} - 1.$$

Now use the standard inequality $e^t > 1 + t$ for $t > 0$. This implies

$$e^t - 1 > t,$$

hence

$$\frac{3}{e^t-1} - 1 < \frac{3}{t} - 1 < \frac{3}{t}.$$

Substituting back $t = \frac{\ln 2}{c}$, we obtain

$$\frac{4-2^{1/c}}{2^{1/c}-1} < \frac{3}{t} = \frac{3c}{\ln 2}.$$

To conclude, if $k > \frac{3c}{\ln 2}$, then in particular $k > \frac{4-2^{1/c}}{2^{1/c}-1}$, which yields

$$(k+4)^c < 2(k+1)^c.$$ ■

With this inequality established, we prove the main hardness result.

▷ **Theorem 3.7.** Weighted Clique Partition is NP-complete and remains NP-hard even on graphs with a polynomial number of maximal cliques. ◁

Proof. Membership in NP is easy to see as polynomial time verification of a solution is straightforward. For hardness, we reduce from X3C to Weighted Clique Partition. To that end, let $I = (X, C)$ be an instance of X3C where no element occurs in more than three subsets. We construct a graph G_I as follows. For each $x \in X$ we include a vertex v_x and call these vertices V_X. For each $c = \{x, y, z\} \in C$

we include vertices $V_c = \{v_c^1, \ldots, v_c^k\}$ for $k = 5 \cdot |C|$ that form a clique together with v_x, v_y and v_z. In the following we show that I has an exact cover if and only if G_I has a clique partition with weight at most $w = (k+4)^{|X|/3} \cdot (k+1)^{|C|-|X|/3}$.

First, suppose that I has an exact cover $C' \subseteq C$. We find a clique partition $\mathcal{P} = P_1, \ldots, P_{|C|}$ as follows. For every $c \in C'$ containing three elements $x, y, z \in X$, the vertices v_x, v_y, and v_z together with V_c form a clique that we select as a partition class. For every $c' \in C \setminus C'$, we select the clique $V_{c'}$ as a partition class. Thus, $\mathcal{P}$ has weight exactly $(k+4)^{|X|/3} \cdot (k+1)^{|C|-|X|/3}$.

For the other direction, suppose that G_I has a clique partition with weight at most w and among such partitions let $\mathcal{P}$ be one with minimum weight. We can assume that $|C| \geq |X|/3$ and that every $x \in X$ is contained in some $c \in C$, as otherwise there clearly is no exact cover of I and instead of constructing G_I we could have returned a trivial no-instance.

First, we note that each k-Clique V_c is contained in exactly one partition class of $\mathcal{P}$, as each vertex of V_c has the same neighbors in V_X and so two partition classes with vertices of V_c could be merged to achieve lower weight. Also, G_I contains no clique larger than $k+3$.

Next, we show that $\mathcal{P}$ contains exactly $|C|$ partition classes. Since every $x \in X$ is contained in some $c \in C$, we can assign each v_x to a partition class containing V_c for some $c \in C$ with $x \in c$. This yields a partition with exactly $|C|$ classes, each of size at most $k+3$, and thus weight at most $(k+4)^{|C|}$. Now suppose that $\mathcal{P}$ contains an additional partition class p beyond the $|C|$ classes containing the V_c's. Then p has size at least 1, contributing a factor of at least 2 to the weight, while the remaining $|C|$ classes each have size at least k, contributing a factor of at least $k+1$ each. Thus the weight of $\mathcal{P}$ is at least $(k+1)^{|C|} \cdot 2$. Using Lemma 3.6 and $k = 5 \cdot |C| > \frac{3|C|}{\ln 2}$, we have $(k+4)^{|C|} < (k+1)^{|C|} \cdot 2$, so $\mathcal{P}$ does not have minimum weight, a contradiction.

This means that $\mathcal{P}$ contains exactly $|C|$ partition classes of size between k and $k+3$. If fewer than $|X|/3$ partition classes have size $k+3$, then there are also partition classes of size $k+2$ or $k+1$. In this case, weight of $\mathcal{P}$ is larger than w by iterative application of Lemma 3.4. It follows that $\mathcal{P}$ contains exactly $|X|/3$ partition classes with exactly three vertices of V_X. This directly translates to an exact cover of I.

It remains to show that G_I only contains a polynomial number of maximal cliques. To that end we first consider the subgraph $G_I[V_X]$. Here, as each $x \in X$ is only contained in at most three subsets, each $v_x \in V_X$ is contained in at most three triangles and has no other edges. Thus, $G_I[V_X]$ has maximum degree 6, which implies a polynomial number of maximal cliques. All additional maximal cliques

in G_I consist of a triangle in V_X together with a clique V_c for $c \in C$. In total the number of maximal cliques in G_I is polynomial in $|X| + |C|$. ■

3.3.2 Heuristic approaches

We now present different heuristic approaches for computing low weight clique partitions.

Maximal clique heuristic. Recall from Lemma 3.4 that the weight function favors imbalanced clique sizes over more balanced ones. It therefore makes sense to try to find few large cliques that cover all vertices. A basic greedy heuristic that tries to achieve this works as follows. First, we enumerate all maximal cliques C of the graph. Then we iteratively add one clique to the partition by greedily selecting the clique with the largest number of remaining uncovered vertices. We call this the *maximal clique heuristic*.

In order to efficiently implement this heuristic, we use a priority queue to fetch the largest clique and keep track of the cliques $C_v \subseteq C$ that a vertex v is part of. This way, after choosing the remaining vertices of a clique $C \in C$ as a partition, we have to update the sizes of $O(\sum_{v \in C} |C_v|)$ cliques. The total number of such updates throughout the whole algorithm is at most the sum of clique sizes in C. Thus, using a Fibonacci Heap, a total running time of $O(|V| \log |C| + \sum_{C \in C} |C|)$ can be achieved. In our implementation we use a binary heap due to it being faster in practice. This costs an additional factor of $\log |C|$ for the second term.

Repeated maximal clique heuristic. Note that the MC heuristic does not recompute the maximal cliques of the remaining graph after selecting a clique. As deleting the vertices of one clique can have the effect that a non-maximal clique becomes maximal, the MC heuristic might miss a clique we would want to select. The *repeated maximal clique heuristic* recomputes the set of maximal cliques after each decision, i.e., it selects a maximum clique of the remaining graph in each step.

Set Cover heuristics. Observe that for the Weighted Clique Partition problem, we have to choose a set of cliques of minimum weight that cover all vertices. Thus, we essentially have to solve a weighted Set Cover problem. As there are reasonably efficient solvers for Set Cover (or the equivalent Hitting Set problem), it seems like a promising approach to use those. However, this has the disadvantage, that we would need to list all cliques and not only the maximal

(a) Unweighted set cover. **(b)** Weighted set cover.

Figure 3.1: Counter-examples for the optimality of the set cover heuristics.

cliques. Nonetheless, it seems like a good heuristic to just consider maximal cliques and find a minimum set cover (unweighted or weighted).

The heuristic consists of two steps. First, we compute a minimum set cover, using the maximum cliques as sets and the vertices as elements. We consider two variants for this steps; weighted (a set of size k has weight $\log(k+1)$) and unweighted (each set has weight 1). Afterwards, in the second step, we convert the cover into a partition by assigning the overlap between selected cliques to only one clique. We call the resulting two approaches the *(maximal clique) set cover* and *(maximal clique) weighted set cover* heuristics.

For the first step, i.e., solving Set Cover, we use a state of the art branch-and-bound solver [Blä+22d] for the unweighted case. Additionally, for the weighted case, we use the straight-forward formulation of set cover as an ILP and solve it with Gurobi [Gur23]. To the best of our knowledge, ILP solvers are currently the state-of-the-art for weighted set cover.

For the second step, we have to compute clique partitions from the resulting set covers by assigning each vertex that is covered by multiple cliques to a single one of these cliques. The goal is to minimize the weight of the resulting cliques, i.e., by Lemma 3.4, we want to distribute them as unevenly as possible. We employ a simple greedy heuristic, assigning each vertex to the largest clique it is part of and braking ties arbitrarily in case of ambiguity.

At a first glance it seems possible that doing both steps optimally (solving set cover and resolving the overlaps) could yield an overall optimal solution. However, this is not the case, as shown by the following two counter examples.

▷ **Observation 3.8.** There are graphs on which the minimum size clique cover cannot give an optimal clique partition. ◁

Proof. Consider a clique on k vertices for even k where half of the vertices are connected to one additional vertex and the other half to another additional vertex, as illustrated in Figure 3.1 (a). Then, for $k \geq 6$ the partition into three cliques of sizes 1, 1, and k has lower weight than the partition into two cliques of size $\frac{k}{2}+1$, which corresponds to the optimal solution of the set cover instance. ■

For the minimum weight set cover, we can use the fact that the weights of the set cover instance correspond to the size of the whole clique and do not reflect the potential overlap between multiple selected cliques.

▷ **Observation 3.9.** There are graphs on which the minimum weight clique cover cannot give an optimal clique partition. ◁

Proof. For the weighted approach, consider the instance depicted in Figure 3.1 (b). The small circles represent the vertices of a graph and the regions mark maximal cliques. The optimal clique partitioning uses cliques of sizes 6, 2, and 1 (the dotted clique plus the remainders of the two solid cliques). In the set cover instance these cliques have (partly overlapping) sizes 6, 4, and 4, which is more expensive than the set cover solution with sizes 5, 4, and 4 (using the dashed clique instead of the dotted one), which results in a solution with sizes 5, 3, 1. ■

The above problem could be avoided by extending the set cover instance to also include all non-maximal subsets of each clique that can be obtained by removing vertices shared with some subset of overlapping cliques. This would, however, lead to an exponential blowup of the set cover instances, which is not feasible even with state of the art solvers.

3.3.3 Exact branch-and-bound solver

Next, we present an exact branch-and-bound algorithm. The idea is to iteratively add cliques to a potential solution partition and to branch on the decision of which clique to select next. In the following, we first explain the branching, before introducing two different lower bounds as well as an additional reduction rule.

Branching. The following lemma enables us to branch on the maximal cliques.

▷ **Lemma 3.10.** Let $\mathcal{P}$ be a minimum weight clique partition of a graph G and let $C \in \mathcal{P}$ be a maximum size clique of $\mathcal{P}$. Then C is a maximal clique in G. ◁

Proof. Assume that C is a non-maximal clique. That is, there is a vertex $v \in V(G) \setminus C$ with $C \subseteq N(v)$. Let $C' \in \mathcal{P}$ be the clique containing v. We construct a clique partition $\mathcal{P}'$ by removing v from C' and adding it to C. As C was a largest clique in $\mathcal{P}$, via Lemma 3.4 we have $(|C| + 2)(|C'|) < (|C| + 1)(|C'| + 1)$, contradicting the optimality of $\mathcal{P}$. ■

Thus, even though not all cliques of an optimal solution might be maximal, we at least know that the largest one is. We can use the decision of which maximal

clique to select as the largest one as the branching decision of our algorithm. This way, we can solve the optimization variant of Weighted Clique Partition, i.e., the algorithm takes a graph G and finds a minimum weight clique partitioning.

After a clique C has been selected as the largest one, the remaining problem is to find a clique partition of $G[V \setminus C]$ that does not use any clique larger than C. This means that we can view our algorithm as a simple recursive subroutine that solves the same problem at every node of the recursion tree. As input it gets the graph G and the cliques $\langle C_1, \ldots, C_i \rangle$ that have already been selected by previous recursive calls. It then tries to compute an optimal clique partition of the remaining graph $G' := G \setminus \bigcup_{j \in [i]} C_j$. This is done by either returning a trivial solution if G' can be covered with a single clique or by branching on the decision of which maximal clique to select as the largest one for the partition of G'. Note that for this decision, only maximal cliques that are at most as large as any of the previously selected cliques $\langle C_1, \ldots, C_i \rangle$ need to be considered. The result of the subroutine call is then the cheapest solution found in any of the branches.

In order to quickly obtain a good upper bound, we explore branches corresponding to larger cliques first. This way, the first leaf of the search tree constructs the same solution as the repeated maximal clique heuristic.

Lower bounds. The *size lower bound* given by following lemma.

▷ **Lemma 3.11.** Let G be a graph with n vertices and $\mathcal{P}$ be a clique partition of G consisting of cliques of size at most s. Then $\mathcal{P}$ has weight at least $(s+1)^{\lfloor n/s \rfloor} \cdot ((n \bmod s) + 1)$. ◁

Proof. The stated minimum weight is achieved by a partitioning $\mathcal{P}'$ that uses as many cliques of size s as possible and one clique with all remaining vertices. Any other partitioning $\mathcal{P}$ using only cliques of size at most s is at least as expensive, as it can be transformed into $\mathcal{P}'$ by Lemma 3.5. ■

The size lower bound can trivially be evaluated in constant time. Even though it is rather basic, we expect this lower bound to be effective at pruning branches in which very small cliques are selected early on.

Note that the size lower bound optimistically assumes that there are $\lfloor n/s \rfloor$ non-overlapping cliques of size s. This yields a bad lower bound if, e.g., there is only one clique of size s while all other cliques are much smaller. In the following, we describe an improved bound based on this observation. We note that we have to be careful when considering what clique sizes are available for the following reason. Assume the branching has already picked a clique of size s, i.e., subsequent selected cliques have to have size at most s. Then it seems natural to derive a lower bound

by summing over the sizes of all maximal cliques of size at most s. However, we have to account for the fact that selecting (and deleting) one clique can shrink a maximal clique that was larger than s to become a clique of size s. Thus, there might be more cliques of size s available than initially thought. In order to formalize this, we first introduce a different problem that considers only sizes of the cliques without making any assumptions on the overlap between the cliques.

In the Valuable Sequence problem, we are given a multi-set A of natural numbers and a natural number n. The task is to construct a sequence of total value n and minimum weight. Such a sequence $S = \langle s_1, s_2, \ldots, s_k \rangle$ consists of elements $s_i \in A$ such that each number is repeated at most as often as it appears in A. In the following, we define value and weight of a sequence and give additional restrictions to what constitutes a valid sequence. To this end, let $S_i = \langle s_1, \ldots, s_i \rangle$ for $i \leq k$ denote a prefix of S. We define a value $\mathrm{val}(s_i)$ for each s_i in the sequence as follows. The first element s_1 has value $\mathrm{val}(s_1) = s_1$. For subsequent elements s_{i+1}, we have $\mathrm{val}(s_{i+1}) = \min\{s_{i+1}, \mathrm{val}(s_i), n - \mathrm{val}(S_i)\}$, where $\mathrm{val}(S_i) = \sum_{j \in [i]} \mathrm{val}(s_j)$ is the total value of the prefix S_i.[5] If $\mathrm{val}(s_i) = s_i$, we say that the element contributes *fully* to the sequence. Otherwise, it contributes *partially*. The *weight* of S is $\prod_{i \in [k]} (\mathrm{val}(s_i) + 1)$. For the subsequence S_i, we call the next element s_{i+1} *eligible* if $s_{i+1} - \mathrm{val}(S_i) \leq \mathrm{val}(s_i)$; s_1 is always eligible. The sequence S is *valid* if each element is eligible.

To make the connection back to Weighted Clique Partition, interpret the numbers in A as the clique sizes. The total value n corresponds to the number of vertices that have to be covered. The value $\mathrm{val}(s_i)$ corresponds to the number of vertices from the maximal clique of size s_i in G that have not been covered by previous cliques, i.e., the number of vertices that are newly covered in step i. Note that in step $i + 1$, at least $s_i - \mathrm{val}(S_i)$ new vertices are covered as only $\mathrm{val}(S_i)$ have been covered previously. Thus, the eligibility requirement ensures that the number of vertices covered in step $i + 1$ is not larger than the number of vertices covered in step i (recall, that we can assume the chosen cliques to form a non-increasing sequence). Moreover, for the definition of $\mathrm{val}(s_{i+1})$, note that the minimum with $\mathrm{val}(s_i)$ ensures that the sequence of values is non-increasing and the minimum with $n - \mathrm{val}(S_i)$ ensures that the total value is n.

The following two lemmas formalize this connection between Valuable Sequence and Weighted Clique Partition. Afterwards, we discuss how Valuable Sequence can be solved optimally.

5 Note that $\mathrm{val}(s_{i+1})$ only depends on values of previous elements in S, i.e., the definition is not cyclic.

▷ **Lemma 3.12.** Let $\mathcal{P}$ be a minimum weight clique partition of a graph G and let $\mathcal{C}$ be the set of maximal cliques in G. Then, any mapping $f : \mathcal{P} \to \mathcal{C}$ with $P \subseteq f(P)$ for each $P \in \mathcal{P}$ is injective and there exists at least one such mapping. ◁

Proof. There are mappings from $\mathcal{P}$ to $\mathcal{C}$, because each clique $P \in \mathcal{P}$ is either a maximal clique or a subset of a larger maximal clique. Assume that a mapping $f : \mathcal{P} \to \mathcal{C}$ is not injective. Then, there are two partition classes P and P' that are mapped to the same clique $C \in \mathcal{C}$. Thus, these partition classes could be merged into a single clique of lower weight, contradicting the optimality of $\mathcal{P}$. ■

▷ **Lemma 3.13.** Let G be a graph with maximal cliques $\mathcal{C} = \{C_1, \ldots, C_k\}$ and let (A, n) with $A = \{|C_1|, \ldots, |C_k|\}$ and $n = |V(G)|$ be an instance of Valuable Sequence. The weight of a minimum solution of (A, n) is a lower bound for the weight of every clique partition of G. ◁

Proof. We now show that for a minimum clique partition $\mathcal{P}$ of G, we find a solution S of (A, n) whose weight is at most the weight of $\mathcal{P}$.

Let $P_1, \ldots, P_{k'}$ be the cliques of $\mathcal{P}$ sorted by size in decreasing order. We can think of $\mathcal{P}$ as constructed iteratively in that order, so that each P_i is a maximal clique in $G[V \setminus (\bigcup_{j \in [i-1]} P_j)]$.

We construct S iteratively until $\mathrm{val}(S) = n$. For each element s_i in the sequence, we prove by induction that $\mathrm{val}(s_i) \geq |P_i|$ except for the last element. This then lets us use Lemma 3.5 to obtain that the weight of S is at most the weight of $\mathcal{P}$. For $i = 1$, we simply choose $s_1 = |P_1| \in A$. Since the first element always contributes fully, we have $\mathrm{val}(s_1) = |P_1|$.

Assuming we constructed the sequence until i, we continue with step $i + 1$ as follows. If P_{i+1} is one of the initial maximal cliques in $\mathcal{C}$, then we can simply choose $s_{i+1} = |P_{i+1}| \in A$. Note that s_{i+1} is eligible, as $s_{i+1} = |P_{i+1}| \leq |P_i| \leq \mathrm{val}(s_i)$, which in particular implies $s_{i+1} - \mathrm{val}(S_i) \leq \mathrm{val}(s_i)$. In this case, s_{i+1} contributes fully, i.e., $\mathrm{val}(s_{i+1}) = |P_{i+1}|$, which implies the claim.

Otherwise, if P_{i+1} is not in $\mathcal{C}$, it is at least a subset of some clique $C \in \mathcal{C}$ such that $P_{i+1} = C \setminus \bigcup_{j \in [i]} P_j$. We choose $s_{i+1} = |C|$. The eligibility of s_{i+1} follows from the facts that the cliques in $\mathcal{P}$ are ordered non-increasingly, i.e. $|P_i| \geq |P_{i+1}|$, and that $\mathrm{val}(s_j) \geq |P_j|$ holds by induction for all $j < i + 1$:

$$\mathrm{val}(s_i) \geq |P_i| \geq |P_{i+1}| = \left|C \setminus \bigcup_{j \in [i]} P_j\right| \geq |C| - \sum_{j \in [i]} |P_j|$$
$$\geq s_{i+1} - \sum_{j \in [i]} \mathrm{val}(s_j) = s_{i+1} - \mathrm{val}(S_i).$$

Note that s_{i+1} contributes partially, i.e., $\text{val}(s_{i+1}) = \text{val}(s_i)$ unless this is the last item in S. As we just argued, we have $\text{val}(s_i) \geq |P_{i+1}|$ and thus $\text{val}(s_{i+1}) \geq |P_{i+1}|$, proving the claim.

To conclude, observe that our construction of S implicitly defines a mapping from $\mathcal{P}$ to C as in Lemma 3.12. As such a mapping is injective, no number in A is chosen twice. Moreover as we have $\text{val}(s_i) \geq |P_i|$ for $i < k$, but both sum to n, the weight of $\mathcal{P}$ is at least the weight of S by Lemma 3.5. ■

Valuable Sequence can be solved optimally with a simple greedy algorithm. We call the resulting lower bound the *valuable sequence* bound.

▷ **Theorem 3.14.** An instance (A, n) of Valuable Sequence can be solved in $O(|A| + n)$ time. ◁

Proof. We construct a solution $S = s_1, \ldots, s_i$ by iteratively choosing an eligible and not yet chosen number $a \in A$ with maximum value, until the value of the sum reaches n.

We note that this greedy strategy maximizes how many numbers in A are eligible, as the corresponding upper bound $\text{val}(S) + \text{val}(s_i)$ decreases as slowly as possible. The optimality of the produced sequence $S = s_1, \ldots, s_i$ follows, again, via Lemma 3.5 as for $j \in [i]$, the value $\text{val}(s_j)$ is at least as large as the value of any other number that can be chosen in round j.

Regarding the running time, the greedy strategy can be implemented by sorting the numbers in A (in $O(|A| + n)$ time) and keeping track of the largest unchosen number that is eligible and contributes fully, as well as the smallest unchosen number that can contribute partially (which is larger than the ones that can contribute fully). Both of these values can be updated in constant time each time a number has been chosen. ■

Sufficient weight reduction. To speed-up the computation of clique partitions for all bags of a tree decomposition, we additionally apply the *sufficient weight reduction*. Here, we immediately accept the first solution that is lighter or equally light as the largest weight of any of the already considered bags.

3.4 Evaluation

With our evaluation, we aim to answer the following questions.

1. How do the different algorithms compare in regards to run time and quality?

2. How do the algorithms scale?

3. What is the impact of the lower bounds and the reduction rules on the performance of the exact branch-and-bound solver?

4. How do different network properties influence the performance of the algorithms?

5. How do the resulting upper bounds on the clique-partitioned treewidth compare to traditional treewidth?

Experimental setup. Our implementation is written in Python. The source code along with all evaluation scripts and results is available on our public GitHub repository.[6] The experiments were run with Python 3.10.1 on a Gigabyte R282-Z93 (rev. 100) server (2250MHz) with 1024GB DDR4 (3200MHz) memory.

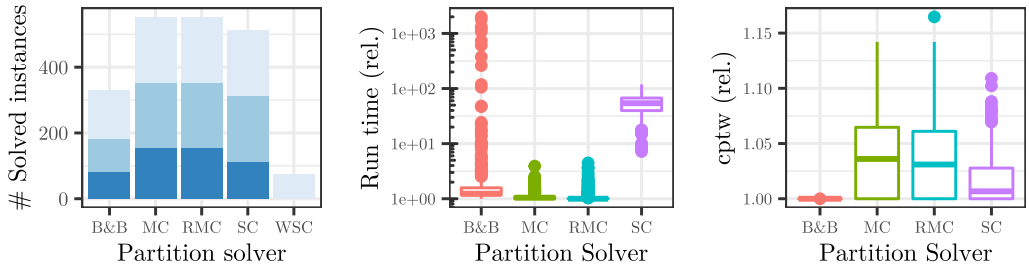
For each input graph, we perform the following two steps. First, we compute a tree decomposition using the heuristics implemented in the HTD library [AMW17]. Specifically, we use the min-fill-in heuristic, which is known to provide a good tradeoff between run time and solution quality [MSJ19]. Secondly, we solve the Weighted Clique Partition problem for each bag of the tree decomposition using all algorithms proposed in Section 3.3.

We use a time limit of five minutes for the heuristic computation of low-weight tree decompositions with the HTD library. For the Weighted Clique Partition algorithms, we set a time limit of three minutes per bag and five minutes in total.

To discern the different solvers from Sections 3.3.2 and 3.3.3, in our plots, we use the following abbreviations: branch and bound solver (B&B), maximal clique set cover heuristic (SC), maximal clique weighted set cover heuristic (WSC), maximal clique heuristic (MC), and repeated maximal clique heuristic (RMC).

Input instances. As input, we use a large collection of real-world networks as well as generated networks. For the latter, we use geometric inhomogeneous random graphs (GIRGs) [BKL19], which resemble real-world networks in regards to important properties and have been shown to be well suited for the evaluation of algorithms [BF24]. GIRGs can be generated efficiently [Blä+22c] and allow to vary the power-law exponent (ple) of the degree distribution controlling its heterogeneity, as well as a parameter α controlling the locality by either strengthening the influence of the geometry (high values of α) or increasing the probability for random edges

6 https://github.com/marcwil/cptw_code



(a) Number of solved instances with 500 (bright) 5 k (medium) and 50 k (dark) vertices.

(b) Distribution of run time relative to fastest solver within time limit on a given graph.

(c) Distribution of obtained width relative to best found solution on a given graph.

Figure 3.2: Comparison of run time and solution quality of the different exact (red), greedy (green, blue) and set cover based (violet) solvers for the Weighted Clique Partition problem on GIRGs.

not based on the geometry (low values of α). We mainly use the following two datasets, where each graph has been reduced to its largest connected component.

- A collection of 2967 real-world networks derived from the dataset of [BF22] by removing 39 graphs that are isomorphic duplicates; the dataset essentially consists of all networks with at most 1 M edges from Network Repository [RA15], see also [BF24] for details.
- GIRGs with $n \in \{500, 5000, 50000\}$ vertices, expected average degree 10, dimension 1, ple $\in \{2.1, 2.3, 2.5, 2.7, 2.9\}$, and $\alpha \in \{1.25, 2.5, 5, \infty\}$. For each parameter configuration, we generate ten networks with different random seeds, to smooth out random variations.

3.4.1 Performance comparison

Here, we evaluate the performance of our Clique Partition approaches on the two datasets.

Generated instances. In Figure 3.2, we compare the run times as well as the solution quality of the different considered Clique Partition algorithms on the dataset of generated networks. In Figure 3.2 (a), we show how many of the 600 instances were solved within the time limit by each solver. While the greedy heuristics are able to finish on almost all instances, the set cover heuristic and the branch-and-bound solver get timed on some of the larger networks with 5 k and

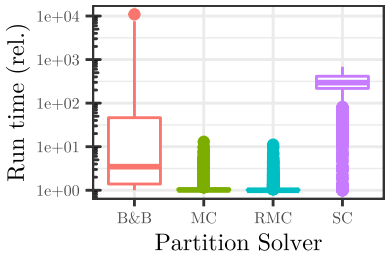


Figure 3.3: Distribution of run time relative to fastest solver within time limit on our set of real-world networks.

Table 3.1: Distribution of obtained cp-treewidth relative to optimum clique partition on our set of real-world networks.

Measure	MC	RMC	SC
Mean	1.008	1.009	1.002
Median	1	1	1
90th percentile	1.035	1.036	1.000
99th percentile	1.108	1.121	1.046
Maximum	1.254	1.192	1.113

50 k vertices. The weighted set cover heuristic performs much worse, finishing only on few networks. We therefore exclude it from the other comparisons.

In Figures 3.2 (b) and 3.2 (c), we compare the performance for all instances that were solved within the time limit by all other algorithms. Figure 3.2 (b) shows the run time of each algorithm relative to the fastest one on each instance. Figure 3.2 (c) shows the obtained upper bound on the cp-treewidth relative to the optimal solution computed by the branch-and-bound solver.

Our findings are as follows. The branch-and-bound solver solves the fewest instances of the four considered algorithms, but is quick on most of the instances it is able to solve within the time limit. Both greedy heuristics (MC and RMC) are similarly fast, significantly outcompeting the other approaches. In terms of quality, all three heuristics perform well, achieving solutions within few percent of the optimum. The set cover heuristic slightly outperforms the greedy heuristics in terms of quality, but pays for this with substantially higher running time.

Real-world networks. We complement the above evaluation of our Clique Partition algorithms, by comparing their performance on the collection of real-world networks. As above, we exclude the weighted set cover heuristic. The other four approaches were able to finish on 1243 (B&B), 2619 (MC), 2622 (RMC), and 2204 (SC) of the 2967 networks within the time limit. We compare our algorithms on the 1237 networks that were solved by all four approaches. Figure 3.3 shows the run time of each solver relative to the fastest solver on each instance. In Table 3.1 we describe the distribution of the obtained upper bounds on the cp-treewidth relative to the optimal solution found by the branch-and-bound solver.

Our results are the following. In general, our observations on generated networks are replicated on the real-world networks. Even though the branch-and-bound algorithm solved fewer instances than the set cover heuristic, it is comparatively

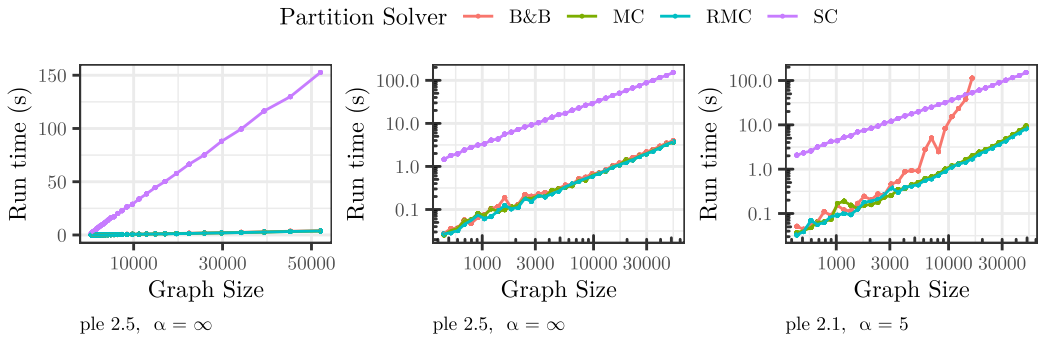


Figure 3.4: Scaling behavior of Clique Partition algorithms on GIRGs with different parameters.

faster on the networks it is able to solve. Both approaches are, however, considerably slower than the greedy heuristics and this difference is more pronounced than on the generated networks. Regarding the solution quality, all three heuristic solvers perform even better than on the generated networks, with only a tiny fraction of instances not being solved almost optimally.

Discussion. We find that the proposed algorithms show good performance both on generated and real-world instances. Although, the branch-and-bound solver was only able to solve about half of the considered networks, its run time typically beats the set cover heuristic on the networks it can solve. In addition, it is a valuable tool for evaluating the solution quality of the other approaches. We find that especially the set cover heuristic, but also the greedy heuristics (MC and RMC) often find close to optimal clique partitions. Due to their excellent trade-off between speed and solution quality, the greedy heuristics are probably the best approach in most practical settings. In general, we do not expect that there is substantial room for improvement in the engineering of Clique Partition solvers for the computation of cp-treewidth. Instead, in order to achieve better upper bounds, we suggest future research to optimize the tree decomposition and the partition into cliques at the same time.

3.4.2 Run time scaling

Next, we consider the scaling behavior of our solvers. For this, we generated GIRGs of varying sizes up to around 50 k vertices for various parameters. As in Section 3.4.1, we did not evaluate the weighted set cover heuristic. Figure 3.4 shows the run times

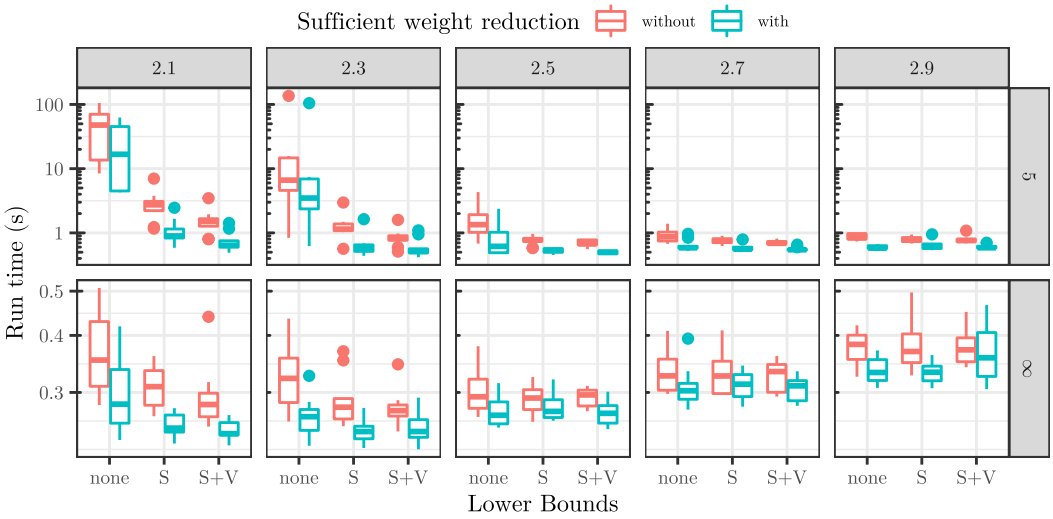


Figure 3.5: Run time of different variants of the branch-and-bound solver on GIRGs with 5 k vertices and different values for the power-law exponent (left to right) and α (top / bottom).

for GIRGs with two different parameter configurations. On the networks with high locality (α = inf), all four approaches seem to have close to linear run time, despite enumerating all maximal cliques present in each bag of the tree decomposition. However, as we decrease the locality ($\alpha = 5$) the performance of the branch-and-bound solver deteriorates while the greedy heuristics and especially the set cover heuristic are only slightly affected. In the logarithmic plot, we observe clearly super-polynomial scaling behavior only for the branch-and-bound solver. Further experiments on a larger grid of parameter settings confirm the above findings.

3.4.3 Branch-and-bound: lower bounds and reduction rule

In the following, we evaluate the effectiveness of the lower bounds and the reduction rule in speeding up our branch-and-bound solver. For this, we use the dataset of generated networks. As the performance without lower bounds does not allow for the timely evaluation on larger instances, we consider only graphs generated with 5 k vertices. Figure 3.5 shows the average run time without lower bounds (none), with only the size lower bound (S) and with the valuable sequence bound in addition to the size bound (S+V) as well as with and without the sufficient weight reduction for different network parameters. We only show $\alpha \in \{5, \infty\}$, as for lower values the variant without lower bounds did not finish within the time limit.

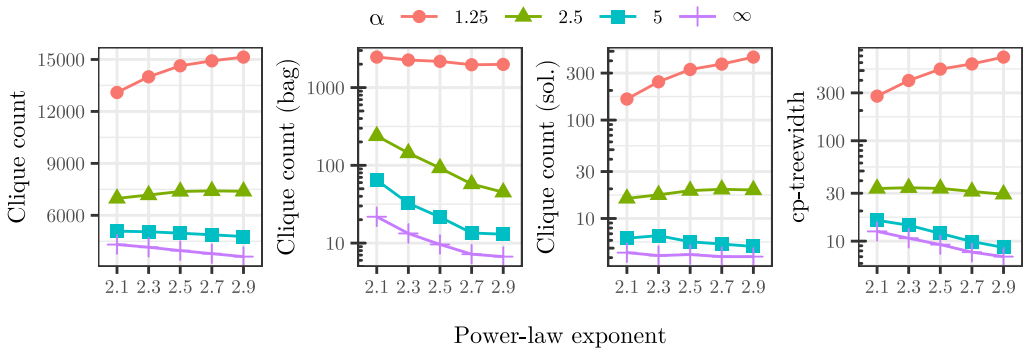


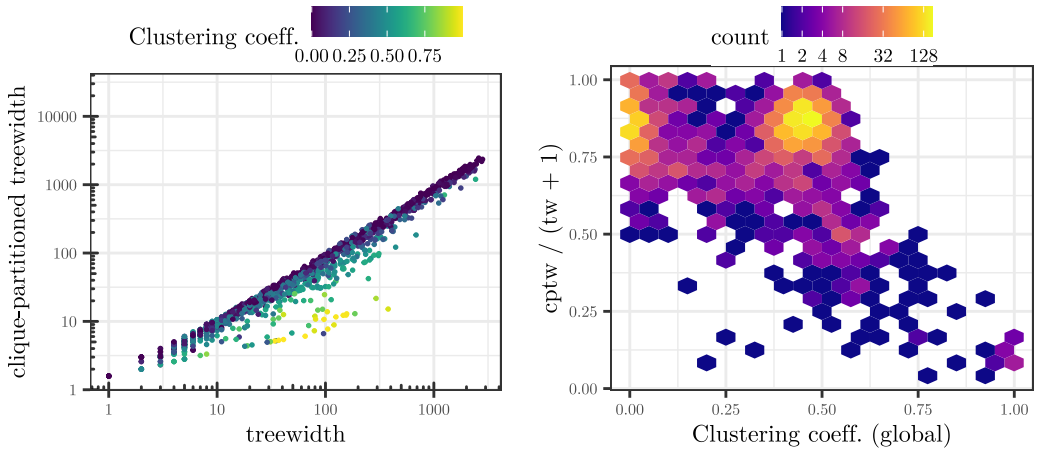
Figure 3.6: Total clique count (number of maximal cliques) per network, and highest clique count in any bag of a greedy tree decomposition as well in the lowest weight clique partition of any bag, and clique-partitioned treewidth (lowest upper bound) of the entire instance on GIRGs with 5 k vertices and varying parameters. Note the logarithmic y-axes on all except the first plot.

We find that especially for smaller power-law exponents, the lower bounds bring large speed-ups of up to multiple orders of magnitude. The additional gain of using the size lower bound is much larger than that of the much simpler valuable sequence bound. The sufficient weight reduction yields similar speed-ups for all settings. Overall, we conclude that the lower bounds are effective in speeding up the branch-and-bound solver. On a more general note, it is striking how strongly all variants of the solver are affected by lower values of α, especially also below the values shown in Figure 3.5. In additional experiments we found that the above observations also apply to the remainder of the dataset, even though for 50 k vertices the time limit is reached even more frequently.

3.4.4 Impact of network properties

At multiple points throughout the last sections, we found that, especially for the branch-and-bound algorithm, the performance strongly depended on the parameter α controlling the locality of the generated networks.

In order to better understand this, we study the structure of cliques in the generated networks depending on their parameters. Specifically, for each network we count the number of maximal cliques in the graph, we count the number of maximal cliques in each bag of the tree decomposition and take the maximum, we count the number of cliques used per bag in the clique-partitioned tree decomposition and take the maximum, and consider the width of the clique-partitioned



(a) Dependency between clustering coefficient and heuristic upper bounds on clique-partitioned treewidth and treewidth.

(b) Dependency between clustering coefficient and relative difference between clique-partitioned treewidth and treewidth.

Figure 3.7: Upper bounds for clique-partitioned treewidth on large real-world networks.

tree decomposition. The clique-partitioned tree decompositions are obtained using the MC and MCR heuristic. Figure 3.6 shows these values for GIRGs with varying power-law exponent and α.

We see that with decreasing values of α, all considered measures increase. However, while the total number of maximal cliques in the network only increases by a factor of roughly 4 to 10, the highest number of cliques intersecting some bag of the tree decomposition as well as the highest number of cliques in a lowest-weight clique partition increase by multiple orders of magnitude. Intuitively, this can be explained by cliques starting to fray if the locality is too low. This explains, why the Clique Partition problem is harder on GIRGs with lower values of α, which slows down the branch-and-bound algorithm. We also observe, that the obtained upper bounds on the cp-treewidth are not much lower than the highest number of cliques per bag of a solution, explaining the good performance of the set cover heuristic.

3.4.5 Comparing cp-treewidth to traditional treewidth

Here we consider the data set of real-world networks. As we have seen in Section 3.4.1, the maximal clique and repeated maximal clique heuristics are efficient and tend to perform well in terms of quality. Thus, we use these two heuristics to find an upper bound on the clique-partitioned treewidth.

In Figure 3.7 (a) we compare the obtained upper bounds for the weighted treewidth and the treewidth. Even though the parameter does not decrease much for the majority networks, there are some networks on which substantial reductions are achieved. This is particularly true for networks with high clustering coefficient, where for some instances our clique-partitioned tree decomposition has width 10 while the corresponding traditional tree decomposition has width above 100. This correspondence with the clustering coefficient fits well to the observations in Section 3.4.4. For the networks for which we do not yet see a big improvement, it would be interesting to see whether adjusting the computation of the initial tree decomposition can yield better bounds; see also the discussion in Section 3.4.1.

3.5 Conclusion and Open Problems

In this chapter we introduced clique-partitioned tree decompositions, investigated the weighted clique partition problem and used heuristic upper bounds to study the new parameter on real-world instances. In the following, we outline several directions for future work, in addition to the open questions regarding heuristic solvers mentioned in Section 3.4.

First, in Section 3.2 we showed that FPT membership is equivalent for treewidth and cp-treewidth, but the translation between the two parameters might incur an exponential blowup in the parameter dependency. Consequently, it would be interesting to characterize the complexity landscape for problems parameterized by cp-treewidth not only with respect to membership in FPT, but also with respect to the dependency on the parameter. For instance, we showed that Independent Set admits an FPT algorithm with a single-exponential dependency on the cp-treewidth. Can the class of problems with this property be classified? Furthermore, it would also be interesting to consider parameterization with cp-treewidth (or other treewidth variants) for problems that are already solvable in polynomial time, such as the radius and diameter problems which can be solved in $2^{O(\mathrm{tw}(G)\log \mathrm{tw}(G))}n^{1+o(1)}$ time [AVW16].

Moreover, besides the algorithm engineering side, it would be interesting to develop algorithms for computing clique-partitioned tree decompositions with good worst-case guarantees. Is it possible to compute or approximate the cp-treewidth in FPT time when parameterizing by the cp-treewidth itself, or, equivalently, by the treewidth? This is especially interesting in light of related results for tree-independence number, a smaller parameter. There, it is already NP-hard to decide whether the tree-independence number is at most k for constant $k \geq 4$, ruling out

FPT algorithms under P≠NP [Dal+26]. On the other hand, it is an open question whether the tree-independence number is FPT with respect to treewidth.

Finally, it would be interesting to compare clique-partitioned treewidth (and similar treewidth variants) with other approaches that exploit structural simplicity beyond small separators. For instance, clique width [CER93] and twin-width [Bon+22] are other parameter that can be small on graphs even if they contain large cliques. It would be interesting to compare the complexity landscapes across these parameters as well as graph classes on which they are low. For example, hyperbolic random graphs and related geometric intersection graphs are known to have polylogarithmic cp-treewidth (see Chapter 4), but it is unclear whether they also have low clique-width or twin-width and how these parameters behave on other models of real-world networks.

4 Hyperbolic Uniform Disk Graphs and Independent Set

This chapter is based on joint work with Thomas Bläsius, Jean-Pierre von der Heydt, Sándor Kisfaludi-Bak, and Geert van Wordragen [Blä+25]. The paper originated during a research visit at Aalto University and was motivated by previous studies of related graph classes (e.g., [Blä+23b; Kis20]).

While most basic ideas underlying the paper were developed jointly, the write-up was divided among the authors, with the author of this thesis contributing the description of the exact algorithm for Independent Set. This chapter focuses on that algorithm, while providing an overview of the other results.

In this chapter, we present a deterministic generalization of a probabilistic network model and demonstrate its usefulness by showing that the previously established polynomial time average-case complexity of a generally NP-hard problem can already be achieved in the more general non-random setting. More specifically, we study *hyperbolic uniform disk graphs* (HUDGs) as a parameterized graph class that generalizes hyperbolic random graphs and consider the vertex cover problem. We then leverage structural properties of independent sets to give a polynomial time algorithm for hyperbolic uniform disk graphs, placing the known polynomial time algorithm for vertex cover on hyperbolic random graphs [Blä+23a] in new context. This demonstrates that finding deterministic alternatives to probabilistic assumptions is not only possible, but can give valuable new perspectives on results from average-case analysis.

4.1 Introduction

Recall from the introduction to this thesis (see, e.g., Page 4) that Hyperbolic random graphs (HRGs) are a popular network model that combines important features of complex real-world networks via it's inherent geometry, while being mathematically simple enough to enable their rigorous analysis. In particular, HRGs have been studied from a number of structural as well as algorithmic perspectives. To give a brief overview, it has been proved that HRGs exhibit locality (clustering) [GPP12], heterogeneity (power-law degree distribution) [GPP12], and the small-world property (logarithmic diameter) [FK18; MS19]. Other studies have established bounds

on the size of balanced separators [BFK16], the size of the maximum clique [BFK18], and the chromatic number [MR26]. Algorithmically, it was proved that the balanced bidirectional search (see also Chapter 5) runs in sublinear time [Blä+22b] and that a number of generally NP-hard problems such as Vertex Cover and Maximum Clique admit polynomial time algorithms [BFK18; Blä+23a].

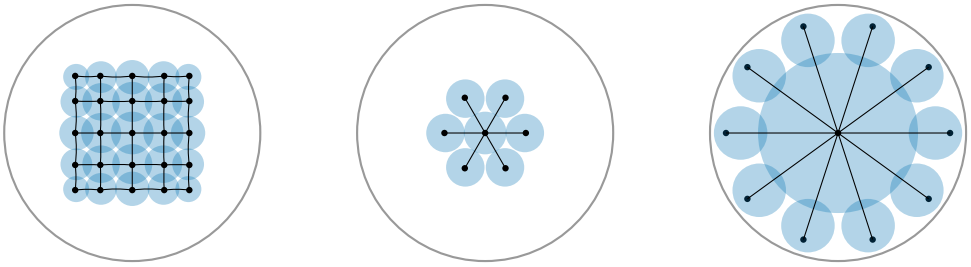
In this chapter we consider the vertex cover problem more closely. A set of vertices in a graph is called a *vertex cover* if it includes at least one endpoint of every edge. The optimization variant of Vertex Cover asks to find a minimum size vertex cover, while the decision variant asks whether there exists a vertex cover of a given size.

We call a graph a *hyperbolic uniform disk graph* with radius r, if it has a representation as the *intersection graph* of disks with radius r in the hyperbolic plane, i.e., each vertex can be associated with a disk such that these disks intersect exactly for adjacent pairs of vertices. Every hyperbolic random graph is also a hyperbolic uniform disk graph for some radius $r \in \Theta(\log n)$. Thus HUDGs can be seen as a deterministic generalization of HRGs, which means that insights about HUDGs also translate to HRGs. We use this to show that the randomization inherent to HRGs is actually not needed in order to solve Vertex Cover in polynomial time. For this, we take a detour to a different related problem. An *independent set* is a set of pairwise non-adjacent vertices and the independent set problem asks to find a maximum size independent set in a given graph, or whether there exists an independent set of a given size. Note that the complement of an independent set is a vertex cover and vice versa, so a graph with n vertices contains a vertex cover of size k if and only if it has an independent set of size $n-k$. This means that a polynomial time algorithm for one of these problem directly translates into a polynomial time algorithm for the other one. We show that on HUDGs the independent set problem can be solved as follows.

▷ **Theorem 4.1.** Let G be a hyperbolic uniform disk graph with radius r and let $k \geq 0$. Then we can decide if there is an independent set of size k in G in $n^{O\left(1+\frac{1}{r}\log k\right)}$ time. ◁

Observe that this running time is polynomial for $r \in \Omega(\log n)$ (or in fact $r \in \Omega(\log k)$), which includes the regime of hyperbolic random graphs. Due to the connection between Vertex Cover and Independent Set the above algorithm for HUDGs can be seen as a deterministic generalization of the average-case result for Vertex Cover on HRGs. This showcases the use of deterministic assumptions as an alternative to probabilistic network models.

The key insight behind the algorithm is that the disk center representing vertices of an independent set have pairwise distance more than $2r$. We then consider these



(a) Small r enable large grids. **(b)** Only small stars possible for small r. **(c)** Large r enable large stars.

Figure 4.1: The structure of hyperbolic uniform disk graphs depends on the radius: large grids are only possible for small radii, while large radii enable hierarchical structures such as large stars.

centers as sites of a Voronoi diagram and find that when viewing the Voronoi diagram as a plane graph, the minimum pairwise distance implies low outerplanarity, especially for larger radii. This allows us to implicitly compute the maximum independent set using a dynamic programming approach. Roughly speaking, we show that any independent set can be hierarchically decomposed into simple polygons and that these polygons can be enumerated and combined in order to find the maximum independent set. Note that this also means that our algorithm requires a geometric representation of the HUDG. This is different to the algorithm for the average-case setting on HRGs, which is oblivious to the geometry and only needs a combinatorial representation of the graph.

In addition to the connection to hyperbolic random graphs, HUDGs are also interesting from the perspective of computational geometry and the study of geometric intersection graphs. HUDGs can be seen as a hyperbolic equivalent of (Euclidean) *unit-disk graphs* (UDGs), i.e., intersection graphs of unit-disks in the Euclidean plane. In fact, every unit-disk graph is also a hyperbolic uniform disk graph, as shown by Bläsius, Friedrich, Katzmann, and Stephan [Blä+23b]. Additionally, every HUDG is also a (Euclidean) disk graph, an intersection graph of arbitrarily individually sized disks. One major difference to UDGs or other intersection graphs of uniformly Euclidean sized disks is that with the radius HUDGs have a parameter strongly influencing the resulting graph structure, see also Figure 4.1. For instance, every Cartesian grid can be represented as a HUDG with radius $r = n^{-3}$, while for $r = \log n$ no grid with side-length at least 5 can be represented as a HUDG [Blä+24, Theorem 7]. Roughly speaking larger – or rather more quickly growing – radii allow for more hierarchical structures, while smaller radii behave more Euclidean and grid-like.

As a graph class, HUDGs were originally introduced by Bläsius, Friedrich, Katzmann, and Stephan [Blä+23b] as *hyperbolic unit disk graphs*. Beyond proving that Euclidean unit disk graphs are also HUDGs, they further define *strongly hyperbolic unit disk graphs* as HUDGs with a maximum distance of $2r$ between vertices. In the sparse setting, this results in graphs very similar to HUDGs with radius in $\Theta(\log n)$. One interesting application demonstrated in [Blä+23b] is that these graphs enable low-stretch greedy routing. A second previously studied graph class related to HUDGs are *hyperbolic uniform ball graphs*, a d-dimensional variant of HUDGs with constant radius. They were introduced in a paper by Kisfaludi-Bak [Kis20], which also contains a separator theorem with far-reaching algorithmic applications, as well as a framework for lower bounds. Here, it should be noted that the restriction to constant radii is substantial, as the structure of HUDGs changes qualitatively when allowing r to grow with the size of the network [Blä+26]. Prior to our joint work in [Blä+25], however, the dependence of structural and algorithmic properties of HUDGs on the radius r had not been systematically studied.

The remainder of this chapter is organized as follows. In Section 4.2 we introduce the necessary background on hyperbolic geometry and Voronoi diagrams. In Section 4.3, we state and discuss the outerplanarity result for hyperbolic Delaunay complexes from [Blä+25], which underlies our algorithm. We then present the exact algorithm for Independent Set in Section 4.4. Finally, Section 4.5 discusses further results and open questions.

4.2 Preliminaries

In this section we give a basic introduction to hyperbolic geometry and introduce fundamental definitions needed for our algorithm and the underlying structural results.

Hyperbolic geometry. Coming from an axiomatic as well as a historical angle, hyperbolic geometry is one of several geometries that arise from modifying or removing axioms of Euclidean geometry, in this case Euclid's fifth postulate, the parallel postulate. Specifically, one defining difference between Euclidean geometry and hyperbolic geometry is that for a straight line ℓ and a point P not on ℓ there are infinitely many straight lines that go through P but do not intersect ℓ. We refer to the book by Ramsay and Richtmyer [RR95] for a more thorough introduction covering the axiomatic angle as well as other natural angles, such as Gaussian curvature, from which the topic can be approached.

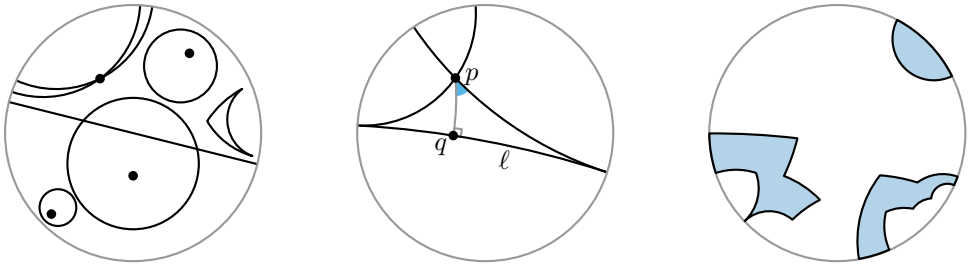
For our purposes, it suffices to present the *Poincaré disk* as a *model* for hyperbolic

geometry. Here, the entire hyperbolic plane $\mathbb{H}^2$ is mapped into the interior of a Euclidean unit disk, see also Figure 4.2. Without going into the exact details of this mapping, we want to highlight two main properties.

The first one is that the Poincaré disk model is *conformal*, i.e., angle preserving, and consequently, hyperbolic circles are represented as Euclidean circles. However, due to the natural distortion, the center of the hyperbolic circle is further from the origin than the Euclidean center of its representation; see Figure 4.2. For some intuition, think of the center of the Poincaré disk as the *origin* of the hyperbolic plane. Then, the model depicts the hyperbolic plane with a distortion that gets stronger as one goes further from the origin and approaches the boundary of the Poincaré disk. Note that the boundary of the Poincaré disk is not part of the hyperbolic plane but instead represents so called *ideal points* at infinite distance from the origin. Note also how the distortion of the Poincaré disk illustrates the claim about the radius dependent structure of HUDGs, as depicted in Figure 4.1. To aid the intuitive understanding of the distortion depicted in the Poincaré disk, it is further helpful to know that the area and circumference of a radius r circle grow as $\Theta(e^r)$ for large r, while being $\Theta(r^2)$ and $\Theta(r)$, respectively, for small r.

The second core property of the Poincaré disk is that straight lines in $\mathbb{H}^2$ are represented as either circular arcs that intersect the Poincaré disk at a right angle or as chords of the Poincaré disk that go through its center; see Figure 4.2. One consequence of this is that the sum of inner angles of triangles is strictly smaller than π. Additionally, this allows us to confirm the (negated) parallel postulate. For one straight line ℓ and a point p not on ℓ there are infinitely many other straight lines through p that do not intersect ℓ. In Figure 4.2 (b), we depict ℓ and p together with the two *limiting parallels* that do not intersect ℓ itself, but each share one ideal point with ℓ. The angle between these two straight lines and the line segment pq with $q \in \ell$ such that it intersects ℓ perpendicularly, is called the *angle of parallelism* and only depends on the distance between p and q. Importantly, it shrinks exponentially in $|pq|$.

Planarity, embeddings and polygons. A *drawing* Γ of a graph G maps its vertices to different points in $\mathbb{R}^2$ and its edges to curves between its endpoints, i.e., $\Gamma(v) \in \mathbb{R}^2$ and $\Gamma(\{u,v\})$ is a curve from $\Gamma(u)$ to $\Gamma(v)$. We sometimes also identify a vertex with its position, i.e., v is used to refer to the vertex as well as to the point $\Gamma(v)$. A drawing is *planar* if no two edges intersect except at a common endpoint and the graph G is planar if it has a planar drawing. Two planar drawings Γ_1 and Γ_2 are *combinatorially equivalent* if there is a homeomorphism of the plane onto itself that maps Γ_1 to Γ_2. The equivalence classes with respect to this equivalence



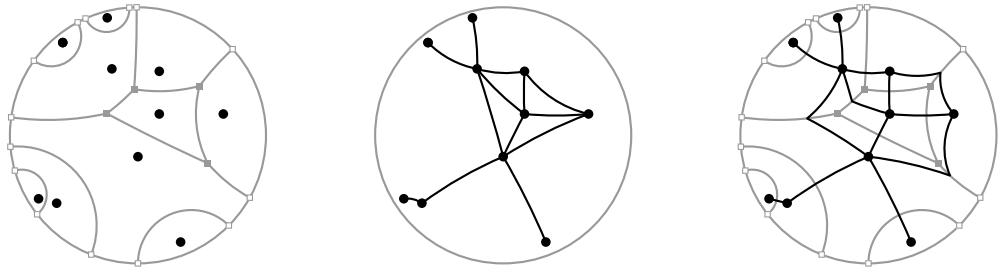
(a) Poincaré disk. **(b)** Angle of parallelism. **(c)** Generalized polygons.

Figure 4.2: Part (a) shows a visualization of the Poincaré disk including (parallel) straight lines, a triangle, and three circles of equal radius with their centers. Part (b) depicts the two limiting parallels of a straight line ℓ through a point $p \notin \ell$, showing additionally the point q where the gray line segment through p intersects ℓ perpendicularly and highlighting the angle of parallelism in blue. Part (c) three generalized polygons including one consisting of only two vertices.

relation are called *(planar) embeddings* and the drawings within such a class are said to *realize* the embedding. A planar graph together with a fixed embedding is also called a *plane* graph.

Consider a graph G with a planar drawing Γ. Removing all edges (i.e., their drawing) from $\mathbb{R}^2$ potentially disconnects $\mathbb{R}^2$ into several connected components, which are called *faces*. Exactly one of these faces is unbounded. It is called the *outer face*; all other faces are *inner faces*. The boundary of each face is a cyclic sequence of vertices and edges. For different drawings realizing the same embedding, these sequences are the same. Thus, to talk about faces and their boundaries, we do not require a drawing but only an embedding. Vertices and edges on this boundary are *incident* to the face. If a vertex appears multiple times on the boundary, it has multiple *incidences* to the face. An edge can also have two incidences to the same face, which is the case if and only if it is a *bridge*, i.e., removing it increases the number of connected components of the graph. In a plane graph, a vertex incident to the outer face is called *outer vertex*; other vertices are *inner vertices*. A plane graph is called *outerplanar* or 1-*outerplanar* if all of its vertices are on the unbounded face. A plane graph is called k-outerplanar if deleting the vertices of the unbounded face (and all edges incident to them) yields a $(k-1)$-outerplanar graph.

Given a plane graph G, the *dual* $G^\star$ has one vertex for each face of G. For every edge e in G, the dual $G^\star$ contains the dual edge $e^\star$ that connects the two faces incident to e (which results in a loop, if e is a bridge). The dual $G^\star$ is itself a plane graph and its dual is $G^{\star\star} = G$. The *weak dual* of G is $G^\star$ without the vertex representing the outer face of G.



(a) Voronoi Diagram (gray). **(b)** Delaunay Complex (black). **(c)** Combined drawing.

Figure 4.3: A set of sites $S \subseteq \mathbb{H}^2$ in the Poincaré disk model, together with their Voronoi diagram $\mathcal{V}(S)$ (gray), with Voronoi vertices drawn as filled squares and ideal Voronoi vertices as empty squares, the Delaunay complex $\mathcal{D}(S)$ (black), and a combined drawing where each edge of $\mathcal{D}(S)$ intersects only its dual edge in $\mathcal{V}(S)$.

A *polygon* is a cycle together with a planar drawing in which all edges are drawn as line segments. Note that this enables the use of graph notation, e.g., using $V(P)$ to refer to the vertices of polygon P.

Specifically on the Poincaré disk we also allow polygons containing points on the boundary of the Poincaré disk, i.e., ideal points. An *ideal arc* between two ideal points q_1 and q_2 is the set of ideal points between q_1 and q_2 when moving clockwise on the boundary of the Poincaré disk. A *generalized polygon* is a cycle together with a planar drawing that maps each vertex to a point in $\mathbb{H}^2$ or to an ideal point. Each edge is either a line segments between points, a ray from a point to an ideal point, a line between two ideal points, or an ideal arc between two ideal points. Note that the vertices do not completely determine the generalized polygon, as two ideal points can be connected via a line or an ideal arc. Moreover, a two vertex cycle can yield a generalized polygon by mapping both vertices to ideal points and one edge to the line and the other to the ideal arc between the two. Figure 4.2 (c) shows generalized polygons with their interior shaded in blue.

Delaunay complexes and Voronoi diagrams. Let S be a set of *sites*, i.e., a set of designated points in $\mathbb{H}^2$. The *Voronoi cell* of a site $s \in S$ is the set of points that are closer to s than to any other site. When considered in the Poincaré disk, the boundary of each Voronoi cell is a generalized polygon.[7] The non-ideal segments are part of the perpendicular bisector of two sites and are called *Voronoi edges*; they are illustrated in gray in Figures 4.3 (a) and 4.3 (c). The points in which

7 We note that Voronoi cells containing ideal points in their boundary are actually unbounded in the hyperbolic plane.

three or more Voronoi edges meet are called *Voronoi vertices*. The ideal points in which unbounded Voronoi edges end are called *ideal Voronoi vertices*. We define the *Voronoi diagram* $\mathcal{V}(S)$ of S as the following plane graph. Its vertex set is the set of all (ideal) Voronoi vertices. The edge set of $\mathcal{V}(S)$ is comprised of the Voronoi edges and the set of ideal arcs connecting consecutive ideal vertices (empty squares in Figures 4.3 (a) and 4.3 (c)).

The weak dual of the Voronoi diagram is called *Delaunay complex* and denoted by $\mathcal{D}(S)$; it is illustrated in black in Figures 4.3 (b) and 4.3 (c). The edges of $\mathcal{D}(S)$ are exactly the edges dual to the Voronoi edges, i.e., the edges of $\mathcal{V}(S)$ that are not ideal arcs. Thus, sites $s_1 \in S$ and $s_2 \in S$ are connected in $\mathcal{D}(S)$ if and only if their Voronoi cells share a boundary. In contrast to the Euclidean case, the outer face of $\mathcal{D}(S)$ is not the convex hull of S. In fact, its boundary is not necessarily a simple cycle; see Figures 4.3 (b) and 4.3 (c). A site is an outer vertex of $\mathcal{D}(S)$ if and only if the corresponding Voronoi cell is unbounded.

Let f_∞ denote the outer face of the Delaunay complex $\mathcal{D}(S)$. For each incidence of an edge e of $\mathcal{D}(S)$ to f_∞, the dual edge $e^\star$ has one ideal vertex as endpoint. If e is not a bridge, $e^\star$ is a ray connecting a Voronoi vertex with an ideal Voronoi vertex. If e is a bridge $e^\star$ is a line connecting two ideal Voronoi vertices; one for each incidence of e to f_∞.

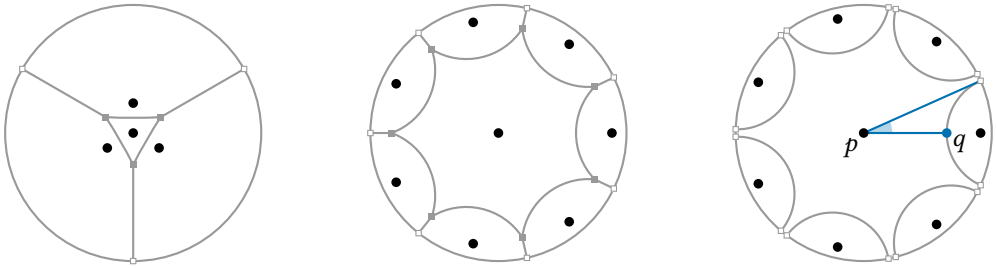
The Voronoi cells are *convex*, i.e., for any two points p, q in the cell or on its boundary, the line segment pq is fully contained in the cell. Thus, drawing each edge $\{u, v\}$ of the Delaunay complex $\mathcal{D}(S)$ by choosing an internal point p on the Voronoi edge dual to $\{u, v\}$ and then connecting u via p to v using the two line segments up and pv yields a planar drawing of $\mathcal{D}(S)$; see Figure 4.3 (c). It has the property that each edge of $\mathcal{D}(S)$ intersects its dual Voronoi edge in exactly one point and it intersects no other Voronoi edges.

4.3 Outerplanarity of Delaunay Complex

In this section we discuss the following result about the outerplanarity of the Delaunay complex of a set of well-separated sites, which is a key ingredient for our independent set algorithm.

▷ **Theorem 4.2 (Theorem 4 in [Blä+25]).** Let S be a set of n sites (points) in $\mathbb{H}^2$ with pairwise distance at least $2r$. Then the Delaunay complex $\mathcal{D}(S)$ is $1 + O\left(\frac{\log n}{r}\right)$-outerplanar. ◁

For $r \in \Theta(1)$ this gives $O(\log n)$-outerplanarity, while $r \in \Omega(\log n)$ even results in $O(1)$-outerplanarity. Additionally, there is a constant c such that any set of n



(a) Three enclosing cells. **(b)** Seven enclosing cells. **(c)** Unbounded Voronoi cells.

Figure 4.4: Voronoi diagrams of sites with increasing pairwise distance in the Poincaré disk model. Part (a) shows a bounded inner Voronoi cell surrounded by three cells. Part (b) shows that with increased pairwise distances more sites are needed to surround the inner cell. Part (c) shows that for sufficiently large pairwise distances no bounded Voronoi cells exist. Additionally, part (c) illustrates the angle of parallelism argument: the blue ray from p to the ideal point represents the limiting parallel through p with respect to the Voronoi edge through q; see also Figure 4.2 (b).

sites with pairwise distance at least $c \log n$ is outerplanar. This is very different to the Euclidean setting where a Delaunay-triangulation can have outerplanarity $\Omega(n)$ [Blä+25].

On plane graphs, the outerplanarity, its dual graph's outerplanarity, and their treewidth are within a constant multiplicative factor of each other [BMT03; Bod98]. Thus a bound of $O(1 + \frac{\log n}{r})$ also transfers to the outerplanarity (and treewidth) of the Voronoi diagram.

For the proof of Theorem 4.2, see [Blä+24; Blä+25]. For a rough intuition recall that a vertex of the Delaunay complex is on the outer face, if and only if the corresponding Voronoi cell is unbounded. If the Voronoi sites have large pairwise distance, then an inner cell in the Voronoi diagram needs to be surrounded by many neighbors, see also Figure 4.4. If r is sufficiently large, the angle of parallelism can be used to show that an unbounded Voronoi cell needs more than seven neighbors. As planar graphs have an average degree of less than 6, it follows that a constant fraction of the vertices of the Delaunay complex are outer cells. For small r this argument does however not work. The proof in [Blä+24] instead considers layers of cells in the Delaunay complex and shows that their area grows exponentially, which implies that there cannot be too many layers.

4.4 Exact Algorithm for Independent Set

In the following, we use the outerplanarity bound from Section 4.3 in order to give an algorithm for the independent set problem. We assume that a hyperbolic uniform disk graph G of radius r is given via its geometric realization. Let S be an independent set of G. As there are no edges between vertices in S, they have pairwise distance at least $2r$. Thus, Theorem 4.2 gives us an upper bound on the outerplanarity of the Delaunay complex $\mathcal{D}(S)$, which implies that $\mathcal{D}(S)$ has small treewidth and thereby small balanced separators. In a nutshell, our algorithm first lists all relevant separators of the Delaunay complexes of all possible independent sets. We then use a dynamic program to combine these candidate separators into a hierarchy of separators, maximizing the number of vertices of the corresponding Delaunay complex and thereby the size of the independent set S.

A crucial tool for dealing with these separators are sphere cut decompositions. They are closely related to tree decompositions but represent the separators as closed curves called *nooses*. We introduce sphere cut decompositions in Section 4.4.1. Afterwards, in Section 4.4.2 we define normalized geometric realizations of nooses for sphere cut decompositions of Delaunay complexes. In Section 4.4.3, we prove that there are not too many candidate nooses and that selecting a subset of candidate nooses that can be combined into a hierarchy of separators actually yields an independent set. Finally, in Section 4.4.4, we describe the dynamic program for computing the independent set, which follows almost immediately from the previous sections.

4.4.1 Sphere Cut Decompositions

A *branch decomposition* [RS91] of a graph G is an unrooted binary tree T, i.e., each non-leaf node has degree exactly 3, and there is a bijection between the leaves of T and the edges of G. Observe that removing an edge $e_T \in E(T)$ separates T into two subtrees and thereby separates its leaves and thus the edges of G in two subsets. Let $E_1 \subset E(G)$ and $E_2 \subset E(G)$ be these two edge sets and let $G[E_1]$ and $G[E_2]$ be their induced subgraphs. The vertices shared by $G[E_1]$ and $G[E_2]$, i.e., vertices that appear in edges of E_1 and of E_2, are called the *middle set* of e_T. The *width* of the branch decomposition T is the size of the largest middle set over all edges of T. The *branch-width* of G is the minimum width of all branch decompositions of G. Robertson and Seymour [RS91] showed that the branch-width of a graph is within a constant factor of its treewidth.

If G is a plane graph, results by Seymour and Thomas [ST94] imply that we can wish for some additional structure without increasing the width of the branch

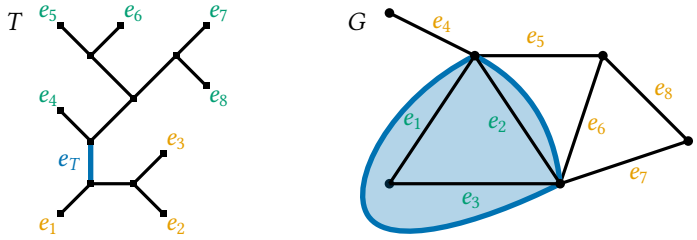


Figure 4.5: A branch decomposition T of a graph G with a highlighted edge e_T of T and the corresponding noose in G, that separates the two parts of $E(G)$.

decomposition [Dor+10; MP22]. Specifically, a *sphere cut decomposition* of a plane graph G is a branch decomposition T such that every tree edge $e_T \in E(T)$ is associated with a so-called *noose*. The noose O of e_T is a simple closed curve that intersects the vertices of G exactly in the middle set of e_T and separates $G[E_1]$ from $G[E_2]$; see also Figure 4.5. Moreover, removing the vertices of the middle set from O separates O into *noose segments* such that each segment lies entirely in the interior of a face of G and every face contains at most one noose segment, except if E_1 or E_2 contains just a bridge of G. In the latter case, the noose consists of two noose segments through the face incident to the bridge. It should be mentioned that this differs from the original definition by Dorn, Penninkx, Bodlaender and Fomin [Dor+10]. Here, a noose may intersect any face at most once and thus the graph cannot have bridges, as discussed by Marx and Pilipczuk [MP22]. With our relaxed definition of nooses however, we get the following theorem.

▷ **Theorem 4.3 ([Dor+10; MP22]).** Let G be a plane graph with branch-width b. Then G has a sphere cut decomposition, where every noose intersects at most b vertices of G. ◁

In the following, we assume sphere cut decompositions to be rooted at a leaf node. Let $e_T \in E(T)$ be a tree edge between $u_T \in V(T)$ and $v_T \in V(T)$ such that u_T is the parent of v_T. Moreover, let $G[E_1]$ and $G[E_2]$ be the two subgraphs defined by e_T and assume that E_1 corresponds to the leaves that are descendants of the child v_T in T. Then, we call the side of the noose O of e_T that contains $G[E_1]$ its *interior* and the side containing $G[E_2]$ its *exterior*. We note that, if the root of T is appropriately chosen, this corresponds to the typical definition of interior and exterior of closed curves in the plane. In the following, when specifying a noose (or a closed curve that could be a noose), its interior and exterior is implicitly given by assuming a clockwise orientation, i.e., we assume that the interior of a noose lies to its right when traversing it in the given orientation.

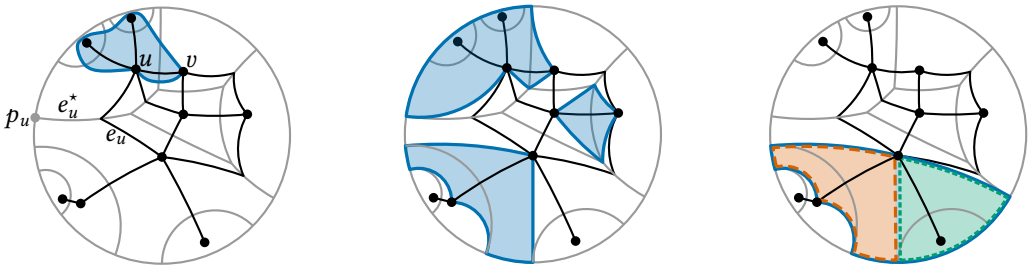


Figure 4.6: Part (a) visualizes the choice of the ideal point p_u for a Voronoi ray, as needed for the normalization of the green noose. Part (b) shows the normalized version of the noose from part (b) alongside two other normalized nooses, each separating a single edge of $\mathcal{D}(S)$. Observe that the depicted normalized nooses include zero, one, or two ideal arcs, respectively. Part (c) shows a parent noose (blue) and its child nooses, with interiors colored green and orange.

We note that rooting the sphere cut decomposition also defines a child–parent relation on the nooses. Consider a non-leaf node $v_T \in V(T)$. It has an edge e_P to the parent, and two edges e_L and e_R to the left and right child corresponding to the nooses O_P, O_L, and O_R, respectively. We say that O_P is the *parent noose* of the *child nooses* O_L and O_R. Note that the subgraph in the interior of O_P is the union of the subgraphs in the interior of O_L and O_R.

4.4.2 Geometric Nooses of Delaunay Complexes

In this section, we are interested in the sphere cut decomposition of the Delaunay complex $\mathcal{D}(S)$ of a set of sites S. When working with a sphere cut decomposition, the exact geometry of the nooses is often not relevant, i.e., it is sufficient to know that there exists a closed curve with the desired topological properties. In our case, however, the geometry is actually important. Our goal here is the following. Given a noose O of a sphere cut decomposition of $\mathcal{D}(S)$, we define a normalized geometric realization of O, i.e., a fixed closed curve satisfying the noose requirements. Afterwards, we will observe that this normalized realization has some nice properties.

Our normalized nooses will be generalized polygons (recall the definition in Section 5.2). We note that generalized polygons are technically not closed curves in the hyperbolic plane. However, they are closed curves in the Poincaré disk when perceived as a disk of the Euclidean plane. Thus, generalized polygons are suited to represent nooses.

To define the normalized nooses, let O be a noose and consider an individual noose segment of O that goes from u to v (which are vertices of $\mathcal{D}(S)$) through the face f of $\mathcal{D}(S)$. If f is an inner face, then f corresponds to a vertex of the Voronoi diagram $\mathcal{V}(S)$. Let p_f be the position of this vertex. Then, the *normalized noose segment* from u to v through f, consists of the two straight line segments from u to p_f and from p_f to v.

If $f = f_\infty$ is the outer face, u and v can have multiple incidences to f_∞ (in fact u and v could be the same vertex). To make the situation more precise, assume that we traverse the boundary of f_∞ such that f_∞ lies to the left and let e_u be the edge incident to f_∞ that precedes the incidence of u where the noose enters f_∞; see Figure 4.6 (a). Recall that the dual Voronoi edge $e_u^\star$ is unbounded and ends in some ideal point p_u. Let e_v and p_v be defined analogously for v. Then, for the normalized noose segment from u to v in f_∞, we use the ray between u and p_u, the ideal arc from p_u to p_v, and the ray between p_v and v.

Observe that combining the geometric noose segments as defined above yields a generalized polygon for each noose; see Figure 4.6 (b). Also, note that this definition also works in the special case where the noose goes only through one vertex with two incidences to the outer face, in which case the noose consists of just two rays and an ideal arc.

It is easy to observe that the above construction in fact yields geometric realizations of the nooses, i.e., the resulting curves have the desired combinatorial properties in regards to which elements of $\mathcal{D}(S)$ lie inside, outside, or on a noose. We call a sphere cut decomposition of $\mathcal{D}(S)$, with such normalized nooses a *normalized sphere cut decomposition*. In the following, we summarize properties of these nooses that we need in later arguments. The first lemma follows immediately from the construction above.

▷ **Lemma 4.4.** Let S be a set of sites with Delaunay complex $\mathcal{D}(S)$ and Voronoi diagram $\mathcal{V}(S)$. Let O be a noose of a normalized sphere cut decomposition of $\mathcal{D}(S)$. Then O is a generalized polygon and each vertex of O is either a site, a vertex of $\mathcal{V}(S)$, or an ideal vertex of $\mathcal{V}(S)$. Between any two subsequent sites visited by O, there is either exactly one vertex of $\mathcal{V}(S)$ or two ideal vertices of $\mathcal{V}(S)$ in $V(O)$. In the latter case, the ideal vertices are connected via an ideal arc. ◁

The previous lemma summarizes the core properties of individual nooses. Next, we describe additional properties of how nooses interact in their parent-child relationships. See also Figure 4.6 (c).

▷ **Lemma 4.5.** Let S be a set of sites with Delaunay complex $\mathcal{D}(S)$. Let O_P, O_L, and O_R be three nooses of a normalized sphere cut decomposition of $\mathcal{D}$ such that

O_P is the parent noose of O_L and O_R. Then O_L, O_R and O_P intersect in exactly two points p_1, p_2 and $O_P \setminus \{p_1, p_2\}$ is the symmetric difference of O_L and O_R. Further, the interiors of O_L and O_R are subsets of the interior of O_P. ◁

Proof. Recall that a noose is a simple closed curve intersecting G precisely at its associated vertex separator. It can be viewed as a sequence of segments, each traversing a face by connecting two incident vertices. For the proof of the lemma, we first take a step back from geometric perspective and define the *combinatorial representation* of a noose as the (multi-)set of its vertex–face incidences. Note that the combinatorial representation of noose O is already uniquely determined by its interior and exterior edges.

We show that the combinatorial representation of O_P is the symmetric difference of the combinatorial representations of O_L and O_R. For this, we partition the edges of G into edges E_L in the interior of O_L, E_R in the interior of O_R and E_{ex} in the exterior of O_P. Then the edges in the interior of O_P are $E_L \cup E_R$. We consider a vertex–face incidence $i = (v, f)$ of one of the child nooses, without loss of generality O_L. As O_L separates E_L from $E_R \cup E_{\text{ex}}$, next to i on the boundary of f there is an edge $e_1 \in E_L$ and an edge $e_2 \in E_R \cup E_{\text{ex}}$. If $e_2 \in E_R$, then O_R also enters f via v and also contains the vertex–face incidence i, while O_P does not contain i. If otherwise $e_2 \in E_{\text{ex}}$, then O_P also visits f through the same vertex incidence, while O_R does not. All other cases are analogous and in particular any vertex–face incidence of the parent noose occurs in exactly one of the child nooses. It follows that the vertex–face incidences of O_P are the symmetric difference of those of O_L and O_R.

The normalized geometric realization of a noose contains a line segment or ray for every vertex–face incidence. This line segment or ray is uniquely determined by the incidence's vertex and face according to the construction of normalized nooses. As O_P, O_L, and O_R are nooses of a normalized sphere cut decomposition, it follows that $E(O_P)$ is the symmetric difference of $E(O_L)$ and $E(O_R)$.[8] Thus, O_P is the symmetric difference of O_L and O_R, except for exactly two points that are visited by all three normalized nooses. These correspond either to a vertex of $\mathcal{D}(S)$ at which all three nooses meet or to an (ideal) Voronoi vertex located in a face of $\mathcal{D}(S)$ that is visited by all three nooses. As the nooses are simple closed curves, it directly follows that the interiors of O_L and O_R are subsets of the interior of O_P. ■

Finally, our last lemma states that a normalized noose does not get too close to sites not visited by that noose.

8 Recall that O_P is a generalized polygon and thus $E(O_P)$ refers to the edges of a plane graph.

▷ **Lemma 4.6.** Let S be a set of sites with pairwise distance at least $2r$, let $\mathcal{D}(S)$ be its Delaunay complex, and let O be a noose of a normalized sphere cut decomposition of $\mathcal{D}(S)$. Then O has distance at least r from any site not visited by O. ◁

Proof. Let $s_1, s_2 \in S$ be two consecutive sites visited by O and let O' be the segment of O between s_1 and s_2. We first argue that O' does not go through the interior of any Voronoi cell of a site other than s_1 or s_2. As O is a noose, O' stays inside one face f of $\mathcal{D}(S)$ with s_1 and s_2 on its boundary. If f is an inner face, then due to Lemma 4.4, O' is comprised of the two line segments from s_1 to the Voronoi vertex corresponding to f and from there to s_2. As Voronoi cells are convex, O' does not enter any other Voronoi cell. Similarly, if f is the outer face, then the line segment of s_i for $i \in \{1, 2\}$ to an ideal vertex q_i of $\mathcal{V}(S)$ on the boundary of the Voronoi cell of s_i does not leave the cell of s_i. Also, the ideal arc between q_1 and q_2 does not enter the interior of any Voronoi cell.

As this holds for any two consecutive sites visited by O, it follows that O goes only through Voronoi cells of sites it visits. Let $s \in S$ be a site not visited by O. As all other sites have pairwise distance at least $2r$ to s, the open disk of radius r around s lies inside the Voronoi cell of s. As the noose does not enter the Voronoi cell of s, it does not enter this open disk and thus has distance at least r from s. ■

4.4.3 Candidate Nooses and Polygon Hierarchies

In this section, we come back to the initial problem of computing an independent set of a hyperbolic uniform disk graph G. From the previous section, we know that any independent set $S \subseteq V(G)$ of G has a Delaunay complex $\mathcal{D}(S)$ with a sphere cut decomposition in which all nooses satisfy the properties stated in Lemmas 4.4–4.6. In the following, we show that these properties are also sufficient in the sense that a collection of nooses satisfying them corresponds to an independent set of G. Thus, instead of looking for an independent set itself, we can search for a hierarchy of nooses that satisfy these properties.

To make this precise, we call a generalized polygon O a *k-candidate noose* for G if there exists an independent set S of size $|S| \leq k$ of G and a minimum width normalized sphere cut decomposition of its Delaunay complex $\mathcal{D}(S)$ that has O as a noose. The following lemma states that there are not too many candidate nooses and that we can compute them efficiently. It in particular implies that even though there may be exponentially many independent sets in G, the normalized sphere cut decompositions of their Delaunay complexes contain substantially fewer different nooses.

▷ **Lemma 4.7.** Let G be a hyperbolic uniform disk graph with radius r, let $k \leq n$ and let C be the set of all k-candidate nooses for G. Then $|C| \in n^{O(1+\frac{\log k}{r})}$. Moreover, a set of generalized polygons that contains C can be computed in $n^{O(1+\frac{\log k}{r})}$ time. ◁

Proof. Let S be a independent set of G with $|S| \leq k$. As the vertices of S have pairwise distance at least r, the Delaunay complex $\mathcal{D}(S)$ of S is $1 + O(\frac{\log k}{r})$-outerplanar by Theorem 4.2. The outerplanarity of a graph gives a linear upper bound on its treewidth [Bod98, Theorem 83] and thus also its branchwidth [RS91, (5.1)]. Thus, $\mathcal{D}(S)$ has branchwidth $w \in O(1 + \frac{\log k}{r})$, i.e., each noose of any minimum width sphere cut decomposition of $\mathcal{D}(S)$ visits at most w sites. As candidate nooses are required to be nooses in minimum width branch decompositions, each candidate noose visits at most w vertices. Each visited vertex is one of n vertices of G and thus there are at most n^w different sequences in which a candidate noose can visit at most w vertices of G.

For a fixed sequence of visited vertices, there is the additional choice of how the candidate noose gets from one vertex to the next. Let u and v be two vertices that are consecutive in this sequence of vertices. As candidate nooses have to be nooses of normalized sphere cut decompositions, we can use that any such normalized noose satisfies Lemma 4.4, i.e., the normalized noose is a generalized polygon and between u and v there is either one Voronoi vertex of $\mathcal{V}(S)$ or two ideal Voronoi vertices of $\mathcal{V}(S)$ with an ideal arc between them. Although we do not know S, there are not too many choices for (ideal) Voronoi vertices. Each Voronoi vertex of $\mathcal{V}(S)$ is the unique point that has equal distance from three vertices in S and is thus determined by choosing three vertices of G. This mean that, although there may be many independent sets S, there are only $O(n^3)$ positions where Voronoi vertices of $\mathcal{V}(S)$ can be. Similarly, each ideal Voronoi vertex is the ideal endpoint of a perpendicular bisector between two vertices of G. Thus, there are only $O(n^4)$ ways to choose two ideal Voronoi vertices. As the noose visits at most w vertices, there are up to w pairs of consecutive vertices u and v and for each we can choose one of the $n^{O(1)}$ ways to connect them, amounting to $n^{O(w)}$ options in total.

To summarize, there are n^w sequences of vertices of G that can be visited by a candidate noose. For each of these sequences, there are $n^{O(w)}$ ways of how a candidate noose can connect these vertices. This gives the upper bound of $n^{O(w)}$ for the number of different nooses. As $w \in O(1 + \frac{\log k}{r})$, this gives the desired bound on $|C|$.

Observe that the above estimate is constructive, i.e., we can efficiently enumerate all possible options and filter out sequences of points that do not give a generalized polygon (e.g., as it would be self-intersecting). Note that not all options actually

yield valid candidate nooses. However, we clearly get a superset of C in the desired time. ■

Next, we go from considering individual candidate nooses to how candidate nooses can be combined into a sphere cut decomposition. To this end, let G be a hyperbolic uniform disk graph with radius r. We define a *polygon hierarchy* as a rooted full binary tree of generalized polygons that visit vertices of G (but can also contain other (ideal) points). Each polygon in the hierarchy is either a leaf or has two child polygons. Let O_P be a parent polygon with children O_L and O_R and let $S(O_P, O_L, O_R) \subseteq V(G)$ be the set of vertices of G visited by at least one of these generalized polygons. We say that this child–parent relation is *valid* if O_L, O_R and O_P meet in exactly two points p_1, p_2 such that $O_P \setminus \{p_1, p_2\}$ is the symmetric difference of O_L and O_R and the interiors[9] of O_L and O_R are subsets of the interior of O_P. Moreover, the child–parent relation is *well-spaced* if the vertices in $S(O_P, O_L, O_R)$ have pairwise distance at least $2r$ and, for each $i \in \{P, L, R\}$, the generalized polygon O_i has distance at least r to each vertex of $S(O_P, O_L, O_R)$ that is not visited by O_i. We call the whole polygon hierarchy valid and well-spaced if each child–parent relation is valid and well-spaced, respectively.

Now consider any independent set S of G with Delaunay complex $\mathcal{D}(S)$. Each noose of a minimum width normalized sphere cut decomposition of $\mathcal{D}(S)$ is a generalized polygon and the nooses of such a sphere cut decomposition form a polygon hierarchy. Observe that the above definition of being valid is directly derived from the properties stated in Lemma 4.5 and thus this hierarchy is valid. Moreover, due to Lemma 4.6, it is also well-spaced. Note that the statement of Lemma 4.6 is actually stronger, as it guarantees the distance requirements globally and not only locally for every child–parent relation. Thus, the sphere cut decomposition of $\mathcal{D}(S)$ yields a valid and well-spaced polygon hierarchy, leading directly to the following corollary.

▷ **Corollary 4.8.** Let G be a hyperbolic uniform disk graph with radius r and let S be an independent set of G. Then the nooses of a normalized sphere cut decomposition of $\mathcal{D}(S)$ form a valid and well-spaced polygon hierarchy of G whose generalized polygons visit all vertices of S. ◁

Next, we show the converse, i.e., that the vertices visited by generalized polygons of a valid and well-spaced hierarchy form an independent set. Together with the corollary, this shows that finding a size k independent set of G is equivalent to finding a valid and well-spaced polygon hierarchy whose generalized polygons visit k vertices.

9 As before, we assume that the region to the right of a closed curve is its interior.

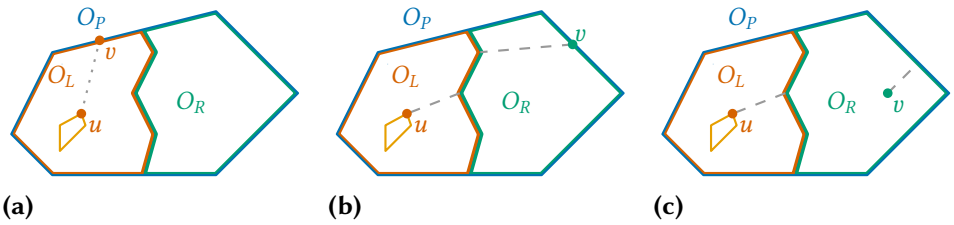


Figure 4.7: Visualization for the case distinction in the proof of Lemma 4.9: either (a) both points are in $S(O_L)$ or v is visited by either (b) O_R or (c) a descendant of O_R. The dotted gray line represents a distance of at least $2r$, the dashed lines distances of at least r.

▷ **Lemma 4.9.** Let G be a hyperbolic uniform disk graph with radius r. The vertices visited by generalized polygons of a valid and well-spaced polygon hierarchy form an independent set of G. ◁

Proof. Consider a valid and well-spaced polygon hierarchy. Let O be a generalized polygon in this hierarchy and let $S(O)$ be the set of vertices of G that are visited by O or by descendants of O. We prove the following claim by induction.

▷ **Claim 4.10.** The vertices in $S(O)$ have pairwise distance $2r$ and each vertex in S_O not visited by O lies in the interior of O and has distance at least r from O. ◁

Note that the first part of the claim that the vertices in $S(O)$ have pairwise distance $2r$ already implies the lemma statement when choosing O to be the root of the hierarchy. The second part of the claim is only there to enable the induction over the tree structure. For the base case, O is a leaf and $S(O)$ is exactly the set of vertices visited by O. Moreover, as the hierarchy is well-spaced, all vertices visited by O have pairwise distance $2r$. Hence, the claim holds for leaves.

For the general case, let O_P be a parent polygon with two children O_L and O_R. To first get the simpler second part of the claim out of the way, let $v \in S(O_P)$ be a vertex not visited by O_P. As the child–parent relation is valid, v is either visited by O_L and O_R or by a descendant of one of the two. If v is visited by O_L and O_R it lies in the interior of O_P and has distance at least r to O_P as the child–parent relation is well-spaced. Otherwise, if v is visited by neither O_L nor O_R but, without loss of generality, by a descendant of O_L. Then by induction, it lies in the interior of O_L and has distance at least r to O_L. As the interior of O_L is a subset of the interior of O_P, the same holds for O_P.

It remains to show the first part of the claim, i.e., that any two vertices $u, v \in S(O_P)$ have distance at least $2r$. If u and v are both visited by one of the polygons O_i for $i \in \{P, L, R\}$, then this follows directly from the fact that the child–parent relation

is well-spaced. Otherwise, assume without loss of generality that u is not visited by one of these three polygons, but is instead visited by a descendant of O_L. We distinguish the following three cases of where v lies; see also Figure 4.7. The first case is that v is visited by O_L or a descendant of O_L. The second case is that it is not visited by O_L but by O_R and the third case is that it is visited by neither O_L nor O_R but by a descendant of O_R. Clearly, this covers all possibilities.

In the first case, v is also visited by O_L or one if its descendants. Thus $u, v \in S(O_L)$ and they have pairwise distance at least $2r$ by induction. In the second case, v lies on O_R but not on O_L and thus in the exterior of O_L. As the hierarchy is well-spaced, v has distance at least r from O_L. Moreover, by induction, u lies in the interior of O_L and has distance at least r from O_L. Thus, u and v have distance at least $2r$. For the third and final case, v lies in the interior of O_R and has distance at least r from O_R by induction. As the interiors of O_L and O_R are disjoint, the line segment between u and v has to cover the distance of at least r from u to O_L and the distance of at least r from O_R to v. Thus, also in this final case, u and v have distance at least $2r$. ■

4.4.4 Solving Independent Set

Lemma 4.9 tells us that for any hyperbolic uniform disk graph G, any valid and well-spaced polygon hierarchy yields an independent set. Moreover, by Corollary 4.8, we can obtain any independent set of G from such a hierarchy. Thus, finding an independent set of size k is equivalent to finding a valid and well-spaced polygon hierarchy that visit k vertices of G. This can be done using a straightforward dynamic program on the set of all k-candidate nooses or, to be exact, on the not too large superset due to Lemma 4.7.

The dynamic program processes all k-candidate nooses in an order such that a noose O is processed after a noose O' if the interior of O' is a subset of the interior of O. When processing a noose O, we compute a valid and well-spaced polygon hierarchy with root O such that the number of vertices visited by nooses in the hierarchy is maximized. We call this the *partial solution for O*. Note that the maximum over the partial solutions of all nooses clearly yields a independent set with at least k vertices if such an independent set exists. Also observe that when processing O, we have already computed the partial solutions of all possible child nooses of O as we are only interested in valid hierarchies. Thus, to compute the partial solution for O, it suffices to consider all pairs of previously processed candidate nooses as potential children of O. For two such child candidates O_L and O_R, we only need to check whether a child–parent relation with O would be valid and well-spaced, which can be easily checked by only considering O_L and O_R. Moreover, if this combination is valid, the number of vertices visited by the

resulting hierarchy with O as root is the sum of the partial solutions for O_L and O_R minus the vertices visited by O_L and O_R. Finally, note that the start of the dynamic program is also easy, as each individual candidate noose by itself is a valid and well-spaced polygon hierarchy (if all its visited vertices are sufficiently far apart). This concludes the description of the dynamic program. Note that the number of partial solutions and thus the running time increase excessively if r is chosen sufficiently small depending on n. In this case, the algorithm is dominated by an algorithm for Euclidean intersection graphs. We obtain the following theorem.

▷ **Theorem 4.1.** Let G be a hyperbolic uniform disk graph with radius r and let $k \geq 0$. Then we can decide if there is an independent set of size k in G in $n^{O\left(1+\frac{1}{r}\log k\right)}$ time. ◁

Proof. The running time follows via the dynamic program described above. Here, we first enumerate all k-candidate nooses in $n^{O(1+\frac{\log k}{r})}$ time (Lemma 4.7). Then, we determine partial solutions in the form of a polygon hierarchy for each k-candidate noose, by considering the partial solutions of all pairs of smaller polygons. In total, this means that the dynamic program can be evaluated in time cubic in the number of k-candidate nooses to find a valid and well-spaced polygon hierarchy maximizing the number of visited vertices in $n^{O(1+\frac{\log k}{r})}$ time. As the vertices visited by such a hierarchy form an independent set by Lemma 4.9 and the enumerated nooses admit a hierarchy corresponding to a size k independent set in G by Corollary 4.8 if such an independent set exists, this concludes the proof. ■

Thus we in particular get a polynomial time algorithm for $r \in \Omega(\log n)$, confirming that the previously achieved polynomial average-case running time on hyperbolic random graphs can even be achieved in the worst-case setting without randomization. For $r \in \Theta(1)$ the running time evaluates to $n^{O(\log k)}$, which is tight under the exponential time hypothesis (ETH) as shown by a conditional lower bound of $n^{\Omega(\log n)}$ [Kis20]. Further, for the Euclidean-like setting of $r \in \Theta(1/\sqrt{n})$ we get a running time of $n^{O(\sqrt{n}\log k)}$, which is somewhat close to the ETH-tight running time of $2^{O(\sqrt{n})}$ on Euclidean unit disk graphs [Ber+20]. Only for even smaller radii, the running time deteriorates increasingly. However, as every HUDG is also a Euclidean disk graph, one can instead switch to Euclidean algorithms, which improves the running time as follows.

▷ **Corollary 4.11.** Let G be a hyperbolic uniform disk graph with radius r and let $k \geq 0$. Then we can decide if there is an independent set of size k in G in $\min\left\{n^{O(1+\frac{1}{r}\log k)}, 2^{O(\sqrt{n})}, n^{O(\sqrt{k})}\right\}$ time. ◁

Proof. Recall that the uniformly sized hyperbolic disks representing the vertices of G are represented as (differently sized) Euclidean disks in the Poincaré disk. This means that the problem can also be solved with algorithms for (Euclidean) disk graphs, such as the ones described by De Berg, Bodlaender, Kisfaludi-Bak, Marx, and Van der Zanden [Ber+20, Corollary 2.4] and Marx and Pilipczuk [MP22]. These run in $2^{O(\sqrt{n})}$ time or $n^{O(\sqrt{k})}$ time, respectively, which concludes the proof. ■

We note that the idea of considering the Voronoi diagram of a solution has already been used by Har-Peled [Har14] and by Marx and Pilipczuk [MP22]. The latter in particular also consider the sphere cut decomposition of such a Voronoi diagram and exploit the fact that as a planar graph with k vertices the Voronoi diagram has balanced separators of size $\sqrt{k}$. Then, even though these separators are unknown, a superset can be enumerated efficiently, which allows for a divide-and-conquer approach with running time $n^{O(\sqrt{k})}$. In our setting, such a divide-and-conquer approach does not give a polynomial running time, as recurring on n^c sub-problems with $c \in \Theta(1)$ gives a recurrence of the form $T(n) = n^c \cdot T(n/2)$ which resolves to $T(n) \in n^{\Theta(\log n)}$. This is the reason why we have to introduce the definitions for candidate nooses and well-spaced and valid polygon hierarchies, as these allow us to implicitly do dynamic programming on the unknown sphere cut decomposition of low width.

4.5 Discussion and Outlook

In this chapter we introduced hyperbolic uniform disk graphs and leveraged properties of Voronoi diagrams in hyperbolic geometry in the development of an algorithm for the independent set problem. This way, we used deterministic properties to show that on hyperbolic random graphs the vertex cover can be solved in polynomial time in the worst-case and not only in the average case, as was previously known.

In the following we summarize further important results about HUDGs that were presented in [Blä+25] and discuss open problems.

Separator theorem and algorithmic applications. The second main structural result from [Blä+25], besides the outerplanarity bound discussed in Section 4.3, concerns separators. Specifically, it shows that HUDGs contain balanced separators that can be covered with a small number of cliques.

▷ **Theorem 4.12 (Theorem 1 in [Blä+25]).** Let G be a hyperbolic uniform disk graph with radius r. Then G has a separator S that can be covered with $O(\log n \cdot$

$\left(1 + \frac{1}{r}\right)$) cliques, such that all connected components of $G - S$ have at most $\frac{2}{3}n$ vertices. The separator can be computed in $O(n \log n)$ time. ◁

For $r \in \Omega(1)$, the separator can be covered by $O(\log n)$ cliques, while requiring a number of cliques with linear dependence on $1/r$ for smaller radii. This generalizes a previous separator theorem by Kisfaludi-Bak [Kis20], which cannot be applied to HUDGs with non-constant radius. In the Euclidean-like setting of $r = 1/\sqrt{n}$ the separator has $O(\sqrt{n} \log n)$ cliques, thus almost matching the $O(\sqrt{n})$ clique separator for unit disk graphs proposed by De Berg, Bodlaender, Kisfaludi-Bak, Marx, and Van der Zanden [Ber+20].

For the proof of Theorem 4.12, see [Blä+24]. The rough idea is that the separator consists of all vertices that are intersecting a carefully chosen line. We then divide the region around this line that contains separator vertices into boxes that each form a clique and bound the number of such boxes.

Note that each clique in the separator has at most n vertices, so a separator consisting of k cliques has weight $O(k \log n)$ with the weight function from Chapter 3. As the separator is $\frac{2}{3}$-balanced the theorem can be applied recursively to obtain a tree decomposition with bags consisting of the union of $O(\log n)$ such separators. This directly implies a clique-partitioned tree decomposition with weight $O\left(\log^3 n(1 + 1/r)\right)$, which allows us to solve the independent set problem in quasi-polynomial time for $r \in \Omega(1)$.

▷ **Corollary 4.13.** Let G be a hyperbolic uniform disk graph with radius r. Then a maximum independent set of G can be computed in $n^{O((1+1/r)\log^2 n)}$ time. ◁

Proof. Recall from Chapter 3 that by Lemma 3.3 a clique-partitioned tree decomposition with weight τ allows to find a maximum independent set of G in $O(2^\tau \cdot \text{poly}(n))$ time. The claimed running time then follows as $2^{O\left(\log^3 n(1+1/r)\right)} = n^{O(\log^2(1+1/r))}$. ■

For $r \in O(1)$ one can additionally consider regular hyperbolic tilings to show that the clique-partitioned treewidth, or indeed also the $\mathcal{P}$-flattened treewidth, is in $O((1 + 1/r) \log^2 n)$ [Blä+25; Kis20]. We refer to [Blä+25, Corollary 3] and [Kis20] for further algorithmic implications of such separators and clique-partitions for a number of different problems.

Approximation algorithm. In [Blä+25] we also present an approximation algorithm for independent set. The algorithm is *$(1 - \varepsilon)$-approximate*, i.e., it finds an independent set with size at least $(1 - \varepsilon)$ times the size of a maximum independent set. The algorithm is further parameterized by the *ply* ℓ of the intersection graph, i.e., every point of $\mathbb{H}^2$ is contained in at most ℓ disks.

▷ **Theorem 4.14 (See Theorem 6 in [Blä+25]).** Let $\varepsilon \in (0,1)$ and let G be a hyperbolic uniform disk graph of radius r and ply ℓ. Then a $(1-\varepsilon)$-approximate maximum independent set of G can be found in $O(n^4 \log n) + n \cdot \left(\frac{\ell}{\varepsilon}\right)^{O(1+\frac{1}{r}\log\frac{\ell}{\varepsilon})}$ time. ◁

Very roughly, the idea for this is similar to Lipton and Tarjan's approach for planar graphs [LT80]: partition the graph into small subgraphs, solve each subgraph exactly (using Theorem 4.1), and ignore the vertices in the separators. For the partition the algorithm uses a known separator [Ber+20]. The approximation guarantee follows because on a HUDG of ply ℓ, the maximum independent set has size $\Omega(n/\ell)$ and it can be ensured that the separator contains only $O(\varepsilon n/\ell)$ cliques, each containing at most one vertex of a maximum independent set. Interestingly, for ply $\ell = n$ and $\varepsilon = 1/n$ the approximation algorithm recovers the asymptotic running time of the exact algorithm.

Open problems. Several natural questions remain open. First, is there a polynomial time algorithm for the independent set problem on HUDGs with radius in $\Omega(\log n)$ that does not require a geometric representation of the graph? Such an algorithm would fully generalize the known algorithm for hyperbolic random graphs [Blä+23a]. This seems difficult, as our approach fundamentally relies on enumerating candidate nooses, which are in terms of Voronoi vertices and perpendicular bisectors and thus require the geometric representation.

Moreover, can the bound for the $O((1+1/r)\log^2 n)$ clique-partitioned treewidth of HUDGs be generalized from the regime with (sub-)constant radius to larger radii? The proof of Corollary 3 in [Blä+24] relies on properties of regular hyperbolic tilings that break down for asymptotically growing radius. At least for hyperbolic random graphs, such a bound should follow rather directly if recursive separators divide their parent separator in a balanced way with respect to the number of cliques. Alternatively, is there an infinite family of HUDGs with radius $\Theta(\log n)$ and clique-partitioned treewidth in $\omega(\log^2 n)$?

Finally, it would be interesting to consider other problems besides independent set on HUDGs, especially for larger radii. Kisfaludi-Bak studied for instance Dominating Set, Feedback Vertex Set, Vertex Cover, connected variants of these problems and more problems on HUDGs with constant radius. Can these problems be solved faster if $r \in \Theta(\log n)$? Apart from these NP-hard problems, another direction could be to search for improvements to polynomial time solvable problems such as Maximum Matching, Diameter, or others.

5 On the Practical Efficiency of Bidirectional BFS

The following chapter is based on joint work with Thomas Bläsius [BW23; BW26]. The project was motivated by previous studies analyzing the bidirectional BFS on random network models [Blä+22b; BN19] and the intuition that exponentially expanding search spaces should already imply sublinear running time. For this thesis, the evaluation on real-world networks has been restricted to the set of networks also used in Chapter 3.

Having seen the use of deterministic properties in the context of two more general parameterized and geometric frameworks, we now turn to the analysis of an individual algorithm. Specifically, we consider the bidirectional breadth-first search algorithm and formulate deterministic conditions that imply sublinear running time for shortest path queries. This way, we give a deterministic explanation for practically observed sublinear running times that were previously only studied in probabilistic settings. The core challenge here is to find conditions that are restrictive enough to allow the derivation of sublinear running time bounds, while also being sufficiently permissive to fit to real-world networks. We achieve this with deterministic properties describing the overlap of exponentially growing neighborhood balls and analyze three increasingly refined settings, one of which results in a necessary and sufficient condition for sublinear running time. We validate these theoretical findings experimentally, confirming that our properties capture the observed performance on a large set of real-world networks.

5.1 Introduction

A common way to speed up the search for a shortest path between two vertices is to use a bidirectional search strategy instead of a unidirectional one. The idea is to explore the graph from both, the start and the destination vertex, until a common vertex somewhere in between is discovered. Even though this does not improve upon the linear worst-case running time of the unidirectional search, it leads to significant practical speedups on some classes of networks. Specifically, Borassi and Natale [BN19] found that bidirectional search seems to run asymptotically faster

than unidirectional search on scale-free real-world networks. This does, however, not transfer to other types of networks like for example transportation networks, where the speedup seems to be a constant factor [Bas+16].

There are several results aiming to explain the practical run times of the bidirectional search, specifically of the balanced bidirectional breadth-first search (short: bidirectional BFS). These results have in common that they analyze the bidirectional BFS on probabilistic network models with different properties. Borassi and Natale [BN19] show that it takes $O(\sqrt{n})$ time on Erdős-Rényi-graphs [ER59] with high probability. The same holds for slightly non-uniform random graphs as long as the edge choices are independent and the second moment of the degree distribution is finite. For more heterogeneous power-law degree distributions with power-law exponent in $\tau \in (2, 3)$, the running time is $O(n^c)$ for $c \in [1/2, 1)$. Note that this covers a wide range of networks with varying properties in the sense that it predicts sublinear running times for homogeneous as well as heterogeneous degree distributions. However, the proof for these results heavily relies on the independence of edges, which is not necessarily given in real-world networks. The authors in [Blä+22b] further consider the bidirectional BFS on network models that introduce dependence of edges via an underlying geometry. Specifically, they show sublinear running time if the underlying geometry is the hyperbolic plane, yielding networks with a heterogeneous power-law degree distribution. Moreover, if the underlying geometry is the Euclidean plane, they show that the speedup is merely a constant factor.

Summarizing these theoretical results, one can roughly say that the bidirectional BFS has sublinear running time unless the network has dependent edges and a homogeneous degree distribution. Note that this fits to the above observation that bidirectional search works well on many real-world networks, while it only achieves a constant speedup on transportation networks. However, these theoretical results only give actual performance guarantees for networks following the assumed probability distributions of the analyzed network models. Thus, the goal of this chapter is to understand the efficiency of the bidirectional BFS in terms of deterministic properties of the considered network.

Intuition. To present our technical contribution, we first give high-level arguments and then discuss where these simple arguments fail. As noted above, the bidirectional BFS is highly efficient unless the networks are homogeneous and have edge dependencies. In the field of network science, it is common knowledge that these are the networks with high diameter, while other networks typically have the small-world property. This difference in diameter coincides with differences in

the expansion of search spaces. To make this more specific, let v be a vertex in a graph and let $f_v(d)$ be the number of vertices of distance at most d from v. In the following, we consider two settings, namely the setting of *polynomial expansion* with $f_v(d) \approx d^2$ and that of *exponential expansion* with $f_v(d) \approx 2^d$ for all vertices $v \in V$. Now assume we use a BFS to compute the shortest path between vertices s and t with distance d.

To compare the unidirectional with the bidirectional BFS, note that the former explores the $f_s(d)$ vertices at distance d from s, while the latter explores the $f_s(d/2) + f_t(d/2)$ vertices at distance $d/2$ from s and t. In the polynomial expansion setting, $f_s(d/2)+f_t(d/2)$ evaluates to $2(d/2)^2 = d^2/2 = f_s(d)/2$, yielding a constant speedup of 2. In the exponential expansion setting, $f_s(d/2) + f_t(d/2)$ evaluates to $2 \cdot 2^{d/2} = 2\sqrt{f_s(d)}$, resulting in a polynomial speedup of the running time.

With these preliminary considerations, it seems like exponential expansion is already the deterministic property explaining the asymptotic performance improvement of the bidirectional BFS on many real-world networks. However, though this property is strong enough to yield the desired theoretic result, it is too strong to actually capture real-world networks. There are two main reasons for that. First, the expansion in real-world networks is not that clean, i.e., the actual increase of vertices varies from step to step. Second, and more importantly, the considered graphs are finite and with exponential expansion, one quickly reaches the graph's boundary where the expansion slows down. Thus, even though search spaces in real-world networks are typically expanding quickly, it is crucial to consider the number of steps during which the expansion persists. To actually capture real-world networks, weaker conditions are needed.

Contribution. The main contribution of this chapter is to solve this tension between wanting conditions strong enough to imply sublinear running time and wanting them to be sufficiently weak to still cover real-world networks. We solve this by defining multiple parameters describing expansion properties of vertex pairs. These parameters address the above issues by covering a varying amount of expansion and stating requirements on how long the expansion lasts. We refer to Section 5.2 and Section 5.3.1 for the exact technical definitions, but intuitively we define the *expansion overlap* as the number of steps for which the exploration cost is growing exponentially in both directions. Based on this, we give different parameter settings in which the bidirectional search is sublinear. In particular, we show sublinear running time for logarithmically sized expansion overlap (Theorem 5.4) and for an expansion overlap linear in the distance between the queried vertices (Theorem 5.6, the actual statement is stronger). For a slightly more general setting

we also prove a tight criterion for sublinear running time in the sense that the parameters either guarantee sublinear running time or that there exists a family of graphs that require linear running time (Theorem 5.12). Note that the latter two results also require the relative difference between the minimum and maximum expansion to be constant. Finally, we demonstrate that our parameters do not only allow for the derivation of sublinear worst-case guarantees for the running time, but additionally capture the behavior actually observed in practice. For this, we perform experiments on a data-set of roughly 3 k real-world networks. The results of this evaluation are presented in Section 5.4.

5.2 Preliminaries

For $i, j \in \mathbb{N}_0$, we write $[i, j]$ for the set $\{i, \dots, j\}$. In a (unidirectional) *breadth-first search* (*BFS*) from a vertex s, the graph is explored layer by layer until the target vertex $t \in V$ is discovered. More formally, for a vertex $v \in V$, the i-th *BFS layer around* v (short: *layer*), $\ell_G(v, i) = N_G(v, i)$, is the set of vertices that have distance exactly i from v. Thus, the BFS starts with $\ell_G(s, 0) = \{s\}$ and then iteratively computes $\ell_G(s, i)$ from $\ell_G(s, i-1)$ by iterating through the neighborhood of $\ell_G(s, i-1)$ and ignoring vertices contained in earlier layers. We call this the i-th *exploration step* from s. We omit the subscript G from the above notation in cases when it is clear from context.

In the *bidirectional* BFS, layers are explored both from s and t until a common vertex is discovered. This means that the algorithm maintains layers $\ell(s, i)$ of a *forward search* from s and layers $\ell(t, j)$ of a *backward search* from t and iteratively performs further exploration steps in one of the directions. The decision about which search direction to progress in each step is determined according to an *alternation strategy*. Note that we only allow the algorithm to switch between the search directions after fully completed exploration steps. If the forward search performs k exploration steps and the backward search the remaining $d(s, t) - k$, then we say that the search *meets* at layer k.

The forward and backward search need to perform a total of $d(s, t)$ exploration steps. To ease the notation, we say that exploration step i (of the bidirectional search between s and t) is either the step of finding $\ell(s, i)$ from $\ell(s, i-1)$ in the forward search or the step of finding $\ell(t, d(s, t) + 1 - i)$ from $\ell(t, d(s, t) - i)$ in the backward search. For example, exploration step 1 is the step in which either the forward search finds the neighbors of s or in which s is discovered by the backwards search. Also, we identify the i-th exploration step with its index i, i.e., $[d(s, t)]$ is the set of all exploration steps. We often consider multiple consecutive exploration

steps together. For this, we define the interval $[i, j] \subseteq [d(s,t)]$ to be a *sequence* for $i, j \in [d(s,t)]$. The *exploration cost* of exploration step i from s equals the number of visited edges with endpoints in $\ell(s, i-1))$, i.e., $c_s(i) = \sum_{v \in \ell(s,i-1)} \deg(v)$. The exploration cost for exploration step i from t is $c_t(i) = \sum_{v \in \ell(t,d(s,t)-i)} \deg(v)$. For a sequence $[i, j]$ and $v \in \{s, t\}$, we define the cost $c_v([i, j]) = \sum_{k \in [i,j]} c_v(k)$. Note that the notion of exploration cost is an independent graph theoretic property and also valid outside the context of a particular run of the bidirectional BFS in which the considered layers are actually explored. Also on any particular graph the total cost of the bidirectional BFS depends only on the number of layers explored by s (or t, respectively), which is determined by the alternation strategy. In particular the cost does not depend on the order in which the layers are explored.

We analyze a particular alternation strategy called the *balanced* alternation strategy [BN19]. This strategy greedily chooses to continue with an exploration step in either the forward or backward direction, depending on which is cheaper. Comparing the anticipated cost of the next exploration step requires no asymptotic overhead, as it only requires summing the degrees of vertices in the preceding layer. The following lemma gives a running time guarantee for balanced BFS relative to any other alternation strategy. It is a generalization of Theorem 3.2 from [Blä+22b].

▷ **Lemma 5.1.** Let G be a graph and (s, t) be a pair of vertices. Assume there exists an alternation strategy such that the bidirectional BFS between s and t meets at layer k and has cost $f(n)$. If the balanced bidirectional search meets at layer k', then the balanced bidirectional search has cost at most $(1 + |k - k'|)f(n)$. ◁

Proof. Let A be the algorithm that meets at layer k and explores $f(n)$ edges. If $k = k'$ both algorithms explore the same edges, so we assume $k < k'$ without loss of generality. Then, the first k exploration steps from s and also the first $d(s,t) - k'$ layers from t are explored by both algorithms. Any difference in the number of explored edges must thus come from the fact that A explores the sequence of exploration steps $[k + 1, k']$ from t while the algorithm with the balanced strategy explores these steps from s. Let $i \in [k + 1, k']$ be one of these exploration steps. As the balanced alternation strategy explores i from s instead of exploring some step $j \in [k', d(s,t)]$ from t, we have $c_s(i) \leq c_t(j)$. Moreover, we have $c_t(j) \leq f(n)$ as A explores j from t. Consequently, the total cost for the balanced bidirectional search is composed of cost at most $f(n)$ for $d(s,t) - |k - k'|$ layers also explored by A, and additionally $|k - k'|$ layers with cost at most $f(n)$ each. ■

This lets us consider arbitrary alternation strategies in our mathematical analysis, while only costing a factor of $|k-k'| \leq d(s,t)$, which is typically at most logarithmic.

For a vertex pair s, t we write $c_{\text{bi}}(s,t)$ for the cost of the bidirectional search with start s and destination t, using the balanced alternation strategy.

5.3 Performance Guarantees for Expanding Search Spaces

We now analyze the bidirectional BFS based on expansion properties. In Section 5.3.1, we introduce expansion, including the concept of expansion overlap, state some basic technical lemmas and give an overview of our results. In the subsequent sections, we then prove our results for different cases of the expansion overlap.

5.3.1 Expanding Search Spaces and Basic Properties

We define *expansion* as the relative growth of the search space between adjacent layers. Let $[i, j]$ be a sequence of exploration steps. We say that $[i, j]$ is *b-expanding from s* if for every step $k \in [i, j)$ we have $c_s(k+1) \geq b \cdot c_s(k)$. Analogously, we define $[i, j]$ to be *b-expanding from t* if for every step $k \in (i, j]$ we have $c_t(k-1) \geq b \cdot c_t(k)$. Note that the different definitions for s and t are completely symmetrical. With this definition layed out, we investigate its relationship with logarithmic distances.

▷ **Lemma 5.2.** Let $G = (V, E)$ be a graph and let $s, t \in V$ be vertices such that the sequence $[1, c \cdot d(s,t)]$ is b-expanding from s for constants $b > 1$ and $c > 0$. Then $d(s,t) \leq \log_b(2m)/c$. ◁

Proof. The cost of discovering the layer with distance $c \cdot d(s,t)$ from s is at least $b^{c \cdot d(s,t)}$. Thus we have $b^{c \cdot d(s,t)} \leq 2m$, which can be rearranged to $c \cdot d(s,t) \leq \log_b(2m)$. ■

Note that this lemma uses s and t symmetrically and also applies to expanding sequences from t. Together with Lemma 5.1, this allows us to consider arbitrary alternation strategies that are convenient for our proofs. Next, we show that the total cost of a b-expanding sequence of exploration steps is constant in the cost of the last step, which often simplifies calculations.

▷ **Lemma 5.3.** For $b > 1$ let $f : \mathbb{N} \to \mathbb{R}$ be a function with $f(i) \geq b \cdot f(i-1)$ and $f(1) = c$ for some constant c. Then $f(n)/\sum_{i=1}^{n} f(i) \geq \frac{b-1}{b}$. ◁

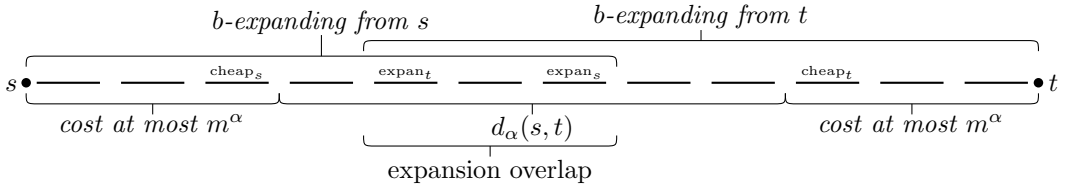


Figure 5.1: Visualization of cheap_v, expan_v and related concepts. Each line stands for an exploration step between s and t. Additionally, certain steps and sequences relevant for Theorem 5.4 and Theorem 5.6 are marked.

Proof. We have $f(i-1) \leq f(i)/b$ and so we get

$$\frac{f(n)}{\sum_{i=1}^{n} f(i)} \geq \frac{f(n)}{\sum_{i=0}^{n-1} \frac{f(n)}{b^i}} = \frac{1}{\sum_{i=0}^{n-1} \frac{1}{b^i}} = \frac{1-\frac{1}{b}}{1-\frac{1}{b}^n} = \frac{b-1}{b-\frac{1}{b}^{n-1}} > \frac{b-1}{b}. \qquad \blacksquare$$

We define four specific exploration steps depending on two constant parameters $0 < \alpha < 1$ and $b > 1$. First, $\text{cheap}_s(\alpha)$ is the latest step such that $c_s([1, \text{cheap}_s(\alpha)]) \leq m^\alpha$. Next, $\text{expan}_s(b)$ is the latest step such that the sequence $[1, \text{expan}_s(b)]$ is b-expanding from s. Analogously, we define $\text{cheap}_t(\alpha)$ and $\text{expan}_t(b)$ to be the smallest exploration steps such that $c_t([\text{cheap}_t(\alpha), d(s,t)]) \leq m^\alpha$ and $[\text{expan}_t(b), d(s,t)]$ is b-expanding from t, respectively. If $\text{expan}_t(b) \leq \text{expan}_s(b)$, we say that the sequence $[\text{expan}_t(b), \text{expan}_s(b)]$ is a *b-expansion overlap*. We define the *size of the expansion overlap* $d_{s,t}^{\text{overlap}}(b)$ as $\text{expan}_s(b) - \text{expan}_t(b) + 1$. Note that $d_{s,t}^{\text{overlap}}(b)$ can also be negative. See Figure 5.1 for a visualization of these concepts. Note that the definition of expan_s (reps. expan_t) cannot be relaxed to only require expansion behind cheap_s (resp. cheap_t). This is because in that case an existing expansion overlap no longer implies logarithmic distance between s and t, which allows for the construction of instances with linear running time (see Lemma 5.5). To simplify notation, we often omit the parameters α and b as well as the subscript s and t if they are clear from the context. Note that cheap_s or cheap_t is undefined if $c_s(1) > m^\alpha$ or $c_t(d(s,t)) > m^\alpha$. Moreover, in some cases expan_v may be undefined for $v \in \{s,t\}$, if the first exploration step of the corresponding sequence is not b-expanding. Such cases are not relevant in the remainder of this chapter.

Overview of our Results. Now we are ready to state our results. Our first result (Theorem 5.4) shows that for $b > 1$ we obtain sublinear running time if the expansion overlap has size at least $\Omega(\log m)$. Note that this already motivates why the two steps expan_s and expan_t and the resulting expansion overlap are of interest.

The logarithmic expansion overlap required for the above result is of course

a rather strong requirement that does not apply in all cases where we expect expanding search spaces to speed up bidirectional BFS. For instance, the expansion overlap is at most the distance between s and t, which might already be too small. This motivates our second result (Theorem 5.6), where we only require an expansion overlap of sufficient relative size, as long as the maximum expansion is at most a constant factor of the minimum expansion b. Additionally, we make use of the fact that cheap_s and cheap_t can give us initial steps of the search that are cheap. Formally, we define the *(α-)relevant distance* as $d_\alpha(s,t) = \text{cheap}_t - \text{cheap}_s - 1$ and require expansion overlap linear in $d_\alpha(s,t)$, i.e., we obtain sublinear running time if the expansion overlap is at least $c \cdot d_\alpha(s,t)$ (see also Figure 5.1) for some constant c.

Finally, in our third main result (Theorem 5.12), we relax the condition of Theorem 5.6 further by allowing expansion overlap that is sublinear in $d_\alpha(s,t)$ or even non-existent. The latter corresponds to an expansion overlap of negative size. Specifically, we define $S_1 = \text{expan}_s$, $S_2 = \text{cheap}_t - \text{expan}_s - 1$, $T_1 = d(s,t) - \text{expan}_t + 1$, and $T_2 = \text{expan}_t - \text{cheap}_s - 1$ (see Figure 5.2). We then give a bound on the values of

$$\rho = \frac{\max\{S_2, T_2\}}{\min\{S_1, T_1\}}, \tag{5.1}$$

for which sublinear running time is guaranteed. We denote this as $\rho_{s,t}(\alpha, b)$ when the parameters require clarification. This bound is tight (see Lemma 5.11), i.e., for all larger values of ρ we give instances with linear running time.

5.3.2 Large Absolute Expansion Overlap

We start by proving sublinear running time for a logarithmic expansion overlap.

▷ **Theorem 5.4.** For parameter $b > 1$ let $s, t \in V$ be a start–destination pair with a b-expansion overlap of size at least $\gamma \log_b(m)$ for a constant $\gamma > 0$. Then $c_{\text{bi}}(s,t) \leq 8 \log_b(2m) \cdot \frac{b^2}{b-1} \cdot m^{1-\gamma/2}$. ◁

Proof. We analyze bidirectional search when meeting in the middle k_{mid} (rounded either up or down) of the expansion overlap. Using Lemma 5.1 for an upper bound on the cost of the balanced bidirectional search under the assumed meeting point, we get

$$c_{\text{bi}}(s,t) \leq d(s,t) \cdot (c_s([1, k_{\text{mid}}]) + c_t([k_{\text{mid}} + 1, d(s,t)])).$$

For an upper bound on $d(s,t)$, note that as there is an expansion overlap, $\text{expan}_s \geq d(s,t)/2$ or $\text{expan}_t \leq d(s,t)/2$. This means that $d(s,t) \leq 2 \log_b(2m)$ by Lemma 5.2.

Applying Lemma 5.3 we get

$$c_{\text{bi}}(s,t) \leq 2\log_b(2m) \cdot \frac{b}{b-1}(c_s(k_{\text{mid}}) + c_t(k_{\text{mid}}+1)),$$

which, assuming without loss of generality $c_s(k_{\text{mid}}) \geq c_t(k_{\text{mid}}+1)$, gives us

$$\leq 4\log_b(2m) \cdot \frac{b}{b-1}c_s(k_{\text{mid}}).$$

At least $\lfloor \frac{1}{2}\gamma \log_b(m) \rfloor$ more b-expanding layers follow after layer $\ell(s, k_{\text{mid}})$. Counting the edges in these layers, we get

$$c_s(k_{\text{mid}}) \cdot b^{\lfloor \frac{1}{2}\gamma \log_b(m) \rfloor} \leq 2m,$$

which can be transformed to

$$c_s(k_{\text{mid}}) \leq 2m \cdot b^{-\lfloor \frac{1}{2}\gamma \log_b(m) \rfloor}.$$

Inserting this into the upper bound for $c_{\text{bi}}(s,t)$, we get

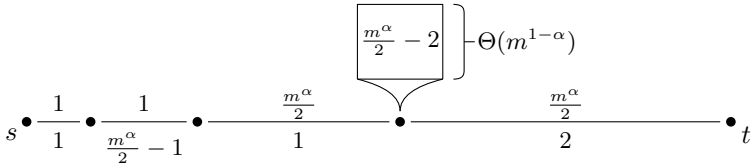
$$\begin{aligned}
c_{\text{bi}}(s,t) &\leq 8\log_b(2m) \cdot \frac{b}{b-1}m \cdot b^{-\lfloor \frac{1}{2}\gamma \log_b(m) \rfloor} \\
&\leq 8\log_b(2m) \cdot \frac{b}{b-1}m \cdot b^{-\frac{1}{2}\gamma \log_b(m)+1} \\
&\leq 8\log_b(2m) \cdot \frac{b^2}{b-1}m \cdot b^{-\frac{1}{2}\gamma \log_b(m)} \\
&\leq 8\log_b(2m) \cdot \frac{b^2}{b-1} \cdot m^{1-\frac{1}{2}\gamma}.
\end{aligned}$$

■

We briefly discuss why the definition of expan_s (respectively expan_t) cannot be relaxed to ignore the expansion in the first $\text{cheap}_s(\alpha)$ (respectively $d(s,t) - \text{cheap}_t(\alpha) + 1$) layers. To that end let expan'_s and expan'_t be defined as the latest exploration step such that the sequences $[\text{cheap}_s(\alpha), \text{expan}_s]$ (respectively $[\text{expan}_t, \text{cheap}_t(\alpha)]$) are b-expanding from s or t, respectively. Let $[\text{expan}'_t, \text{expan}'_s]$ be the *relaxed* expansion overlap of size $\text{expan}'_s - \text{expan}'_t + 1$.

▷ **Lemma 5.5.** For parameters $b > 1$ and $0 < \alpha < 1$, there is an infinite family of graphs with relaxed expansion overlap in $\Theta(\log m)$ on which the cost of the bidirectional search is linear. ◁

Proof. We construct an instance as sketched below.



We connect s to the rest of the graph via one layer with cost 1, a second layer with cost $\frac{m^\alpha}{2} - 1$ and another $\frac{m^\alpha}{2}$ layers of cost 1. Similarly, we connect t via $\frac{m^\alpha}{2}$ layers of cost 2. Thus, we formed cheap regions of cost m^α around s and t. Behind this, we append the rest of the graph consisting of $\Theta(\log_b m)$ b-expanding layers with cost up to $\frac{m^\alpha}{2} - 2$ that are followed by $\Theta(m^{1-\alpha})$ layers of cost $\frac{m^\alpha}{2} - 2$.

As the b-expanding layers follow directly after the cheap regions with cost m^α, they form a relaxed expansion overlap of logarithmic size. However, the balanced alternation strategy will only perform one step in the forward direction and instead explore the (individually) cheaper layers of cost 2 and $\frac{m^\alpha}{2} - 2$. As there are $\Theta(m^{1-\alpha})$ layers of cost $\frac{m^\alpha}{2} - 2$, this results in linear cost. ■

5.3.3 Large Relative Expansion Overlap

Note that Theorem 5.4 cannot be applied if the size of the expansion overlap is too small. We resolve this in the next theorem, in which the required size of the expansion overlap is only relative to α-relevant distance between s and t, i.e., the distance without the first few cheap steps around s and t. Additionally, we say that b^+ is the *highest expansion between s and t* if it is the smallest number, such that there is no sequence of exploration steps that is more than b^+-expanding from s or t.

▷ **Theorem 5.6.** For parameters $0 \leq \alpha < 1$ and $b > 1$, let $s, t \in V$ be a start–destination pair with b-expansion overlap of size at least $\gamma \cdot d_\alpha(s,t)$ for some constant $\gamma > 0$ and assume that $b^+ \geq b$ is the highest expansion between s and t. Then $c_{\text{bi}}(s,t) \in \tilde{O}\big(m^{1-\varepsilon}\big)$ for $\varepsilon = \frac{\gamma(1-\alpha)}{\log_b(b^+)+\gamma} > 0$. ◁

Proof. We perform a case distinction on the size of the expansion overlap. In case of a large expansion overlap we apply Theorem 5.4. Otherwise, the number of layers is small enough that even if their cost grows with a factor b^+ per step, the overall cost will be sublinear.

We begin with the first case. As there is an expansion overlap of size $\gamma \cdot d_\alpha(s,t)$, we can apply Theorem 5.4 if $d_\alpha(s,t)$ is large enough. Assume that $d_\alpha(s,t) \geq a \log_b(m)$ for some constant a to be determined later. Then, by Theorem 5.4 we immediately

get a sublinear upper bound

$$c_{\text{bi}}(s,t) \leq 8 \log_b(2m) \cdot \frac{b^2}{b-1} \cdot m^{1-\gamma a/2},$$

if we can choose a suitably.

In the second case, we can assume $d_\alpha(s,t) < a \log_b m$. In this case, we can find an upper bound for $c_{\text{bi}}(s,t)$ by considering the cost for an assumed meeting point in the middle between cheap_s and cheap_t, i.e., after $\text{cheap}_s + \lceil d_\alpha(s,t)/2 \rceil$ steps of the forward search and $(d(s,t) - \text{cheap}_t + 1) + \lfloor d_\alpha(s,t)/2 \rfloor$ steps of the backward search. Via Lemma 5.1 we thus get

$$\begin{aligned} c_{\text{bi}}(s,t) \leq d(s,t) \cdot \big(&c_s([1, \text{cheap}_s + \lceil d_\alpha(s,t)/2 \rceil]) + \\ &c_t([\text{cheap}_t - \lfloor d_\alpha(s,t)/2 \rfloor, d(s,t)])\big). \end{aligned}$$

Pessimistically assuming $c_s(i+1) = c_s(i) \cdot b^+$ for $i \geq \text{cheap}_s$, we can use Lemma 5.3 to obtain

$$\begin{aligned} &\leq d(s,t) \cdot \left(\frac{b^+}{b^+ - 1} c_s(\text{cheap}_s + d_\alpha(s,t)/2 + 1) + \right. \\ &\qquad \left. \frac{b^+}{b^+ - 1} c_t(\text{cheap}_t - d_\alpha(s,t)/2) \right) \\ &\leq d(s,t) \cdot \left(\frac{b^{+2}}{b^+ - 1} c_s(\text{cheap}_s) \cdot b^{+d_\alpha(s,t)/2} + \right. \\ &\qquad \left. \frac{b^+}{b^+ - 1} c_t(\text{cheap}_t) \cdot b^{+d_\alpha(s,t)/2} \right), \end{aligned}$$

where we apply $c_s(\text{cheap}_s) \leq c_s([1, \text{cheap}_s]) \leq m^\alpha$ and, symmetrically, $c_t(\text{cheap}_t) \leq c_t([\text{cheap}_t, d(s,t)]) \leq m^\alpha$ and $d_\alpha(s,t) < a \log_b m$ to get

$$\begin{aligned} &\leq d(s,t) \cdot \left(2 \cdot \frac{b^{+2}}{b^+ - 1} m^\alpha \cdot b^{+a \log_b m/2} \right) \\ &\in O\left(d(s,t) \cdot m^{\alpha + a \log_b(b^+)/2} \right). \end{aligned}$$

In order to find the optimal choice of a, we set the two exponents from the case distinction to be equal

$$\begin{aligned} 1 - \gamma a/2 &= \alpha + a \log_b(b^+)/2 \\ 1 - \alpha &= a \log_b(b^+)/2 + \gamma a/2 \end{aligned}$$

$$a = \frac{2(1-\alpha)}{\log_b(b^+) + \gamma}.$$

As there is an expansion overlap, we have $\text{expan}_s \geq d(s,t)/2$ or $\text{expan}_t \leq d(s,t)/2$ and thus $d(s,t) \leq 2\log_b(2m)$ by Lemma 5.2. With this we finally obtain

$$c_{\text{bi}}(s,t) \in O\left(d(s,t) \cdot m^{1-\frac{\gamma(1-\alpha)}{\log_b(b^+)+\gamma}}\right) \subseteq \tilde{O}\left(m^{1-\frac{\gamma(1-\alpha)}{\log_b(b^+)+\gamma}}\right).$$

■

Note that Theorem 5.6 does not require $\text{expan}_t > \text{cheap}_s$ or $\text{expan}_s < \text{cheap}_t$, i.e., the expansion overlap may intersect the cheap prefix and suffix. Also, note that the condition is more general than the one in Theorem 5.4. Before extending this result to an even wider regime, we want to briefly mention a simple corollary of the theorem, in which we consider vertices with an expansion overlap region and sublinear degree m^δ for constant $0 < \delta < 1$.

▷ **Corollary 5.7.** For parameter $b > 1$, let $s, t \in V$ be a start–destination pair with a b-expansion overlap of size at least $\gamma \cdot d(s,t)$ for a constant $0 < \gamma \leq 1$. Further, assume that $\deg(t) \leq \deg(s) \leq m^\delta$ for a constant $\delta \in (0,1)$ and that b^+ is the highest expansion between s and t. Then $c_{\text{bi}}(s,t) \in \tilde{O}\left(m^{1-\frac{\gamma(1-\delta)}{\log_b(b^+)+\gamma}}\right)$. ◁

This follows directly from Theorem 5.6, using $\text{cheap}_s(\delta)$ and $\text{cheap}_t(\delta)$.

5.3.4 Small or Non-Existent Expansion Overlap

Theorem 5.6 is already quite flexible, as it only requires an expansion overlap with constant size relative to the distance between s and t, minus the lengths of a cheap prefix and suffix. In this section, we weaken these conditions further, obtaining a tight criterion for polynomially sublinear running time. In particular, we relax the size requirement for the expansion overlap as far as possible. Intuitively, we consider the case in which the cheap prefix and suffix cover almost all the distance between start and destination. Then, the cost occurring between prefix and suffix can be small enough to stay sublinear, regardless of whether there still is an expansion overlap or not.

In the following we first examine the sublinear case. Afterwards, we construct a family of graphs with linear running time for the other case and putting together the complete dichotomy in Theorem 5.12. Note that while this extends the condition given by Theorem 5.6 by relaxing the size requirement for the expansion overlap, it

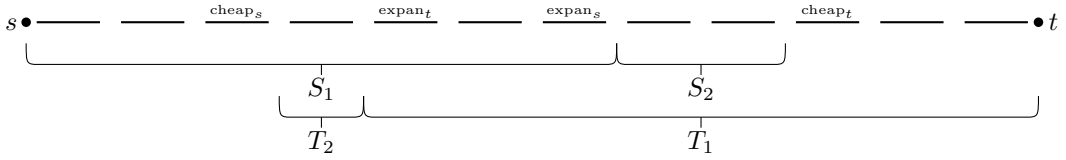


Figure 5.2: Visualization of exploration steps and (lengths of) sequences relevant for Lemmas 5.9 and 5.10

is not a strict generalization, as Theorem 5.6 is more flexible regarding the placement of the expansion overlap.

We begin by giving an intuitive overview of our proof for the sublinear case. The exact criterion depends on $\rho_{s,t}(\alpha, b) = \frac{\max\{S_2,T_2\}}{\min\{S_1,T_1\}}$ as defined in Section 5.3.1; see also Figure 5.2. The intuition is that if ρ is small, there cannot be many exploration steps after expan_s until cheap_t from s, or after expan_t until cheap_s from t, respectively. Using this, we show that if ρ is small, we can give a sublinear upper bound for the cost of exploring up to expan_t from s or, up to expan_s from t, respectively. Our upper bound then follows by combining this with the previous theorems.

We begin by proving an upper bound for the length of low-cost sequences, such as $[1, \text{cheap}_s]$ and $[\text{cheap}_t, d(s,t)]$.

▷ **Lemma 5.8.** Let v be a vertex with a b-expanding sequence S starting at v with cost $c_v(S) \leq C$. Then $|S| \leq \log_b(C) + 1$. ◁

Proof. We have

$$C \geq c_v(S) = \sum_{i=1}^{|S|} c_v(i) \geq \sum_{i=0}^{|S|-1} b^i \geq b^{|S|-1}$$

and thus get $|S| \leq \log_b(C) + 1$. ■

This means that $c_s([1, \text{cheap}_s]) \leq m^\alpha$ implies $\text{cheap}_s \leq \alpha \log_b(m) + 1$. This and the symmetrical statement from the direction of t are used in the following technical lemma. We show that if $\rho_{s,t}(\alpha, b)$ is sufficiently small, we find a sublinear upper bound for the cost of exploring from s up to expan_t and from t up to expan_s.

▷ **Lemma 5.9.** For parameters $0 \leq \alpha < 1$ and $b > 1$, let $s, t \in V$ be a start–destination pair and assume that b^+ is the highest expansion between s and t and $\rho_{s,t}(\alpha, b) < \frac{1-\alpha}{1-\alpha+\alpha\log_b(b^+)}$. There are constants $c > 0$ and k such that if the size of the b-expansion overlap is less than $c \cdot \log_b(m) + k$, then there is a constant $x < 1$ such that $c_s([1, \text{cheap}_s + T_2]) \leq 2^{1-\alpha} \cdot m^x$ and $c_t([\text{cheap}_t - S_2, d(s,t)]) \leq 2^{1-\alpha} \cdot m^x$. ◁

Proof. We use d^{overlap} for the size of the expansion overlap. With Figure 5.2 as reference, it is easy to verify that $d^{\text{overlap}} = S_1 - T_2 - \text{cheap}_s$. Note that this also holds in the case of $d^{\text{overlap}} \leq 0$. Without loss of generality assume $S_1 \geq T_1$. Together with the definition of ρ this implies $S_1\rho \geq T_1\rho \geq \max\{S_2, T_2\}$. We use $T_2 \leq S_1\rho$ and $S_1 \geq \frac{\max\{S_2,T_2\}}{\rho}$ to obtain

$$\begin{aligned} d^{\text{overlap}} &= S_1 - T_2 - \text{cheap}_s \\ &\geq S_1(1-\rho) - \text{cheap}_s \\ &\geq \frac{1-\rho}{\rho}\max\{S_2, T_2\} - \text{cheap}_s, \end{aligned}$$

which we rephrase as

$$\max\{S_2, T_2\} \leq \frac{\rho}{1-\rho}(d^{\text{overlap}} + \text{cheap}_s). \tag{5.2}$$

This means that a small expansion overlap also implies small S_2 and T_2.

We use this to derive a suitable upper bound on the expansion overlap that gives an upper bound on T_2 and S_2 for which the desired sublinear cost follows. As no exploration step is more than b^+-expanding, we have

$$c_s(\text{cheap}_s + T_2) \leq m^\alpha \cdot b^{+T_2} \tag{5.3}$$

if we pessimistically assume maximum expansion in every step. Note that under the pessimistic assumption of b^+-expansion, this is at least a constant fraction of the cost in $c_s([1, \text{cheap}_s + T_2])$ and, by symmetry, also in $c_t([\text{cheap}_t - S_2, d(s,t)])$. We use Equation (5.3) and derive

$$\begin{aligned} c_s(\text{cheap}_s + T_2) &\leq m^\alpha \cdot m^{T_2 \cdot \log_m(b^+)} \\ &= m^\alpha \cdot m^{T_2 \cdot \log_m(b^+) - (1-\alpha)\cdot\log_m 2} \cdot m^{(1-\alpha)\log_m 2} \\ &= 2^{1-\alpha} \cdot m^{\alpha + T_2 \cdot \log_m(b^+) - (1-\alpha)\cdot\log_m 2}. \end{aligned}$$

Clearly, this is sublinear if the exponent of m is smaller than 1. Investigating this, we get

$$\begin{aligned} \alpha + T_2 \cdot \log_m(b^+) - (1-\alpha)\log_m 2 &< 1 \\ T_2 \log_m(b^+) &< 1 - \alpha + (1-\alpha)\log_m 2 \\ T_2 &< \frac{(1-\alpha)(1+\log_m 2)}{\log_m(b^+)} \end{aligned}$$

$$T_2 < (1-\alpha)\log_{b^+}(2m).$$

Using $T_2 \leq \frac{\rho}{1-\rho}(d^{\text{overlap}} + \text{cheap}_s)$ from above, we continue with a stricter inequality

$$\begin{aligned}\frac{\rho}{1-\rho}(d^{\text{overlap}} + \text{cheap}_s) &< (1-\alpha)\log_{b^+}(2m)\\ d^{\text{overlap}} + \text{cheap}_s &< \frac{(1-\alpha)(1-\rho)}{\rho}\log_{b^+}(2m)\end{aligned}$$

and use $\alpha\log_b(2m)+1$ as an upper bound from Lemma 5.8 for cheap_s to derive a sufficient upper bound on d^{overlap} as

$$\begin{aligned}d^{\text{overlap}} &< \frac{(1-\alpha)(1-\rho)}{\rho}\log_{b^+}(2m) - \alpha\log_b(2m) - 1\\ d^{\text{overlap}} &< \left(\frac{(1-\alpha)(1-\rho)}{\rho\log_b b^+} - \alpha\right)\log_b(2m) - 1.\end{aligned}$$

Relying on the initial assumption on ρ, we verify that the factor before the logarithm is a positive constant

$$\begin{aligned}\frac{(1-\alpha)(1-\rho)}{\rho\log_b b^+} - \alpha &> 0\\ \frac{1-\alpha}{\rho\log_b(b^+)} - \frac{\rho(1-\alpha)}{\rho\log_b(b^+)} - \frac{\alpha\rho\log_b(b^+)}{\rho\log_b(b^+)} &> 0\\ \frac{1-\alpha-\rho(1-\alpha)-\alpha\rho\log_b(b^+)}{\rho\log_b(b^+)} &> 0\\ 1-\alpha-\rho(1-\alpha)-\alpha\rho\log_b(b^+) &> 0\\ 1-\alpha &> \rho\big((1-\alpha)+\alpha\log_b(b^+)\big)\\ \rho &< \frac{1-\alpha}{1-\alpha+\alpha\log_b(b^+)}.\end{aligned}$$

Thus the condition under which the considered sequences of exploration steps have sublinear cost can be expressed as $d^{\text{overlap}} < c\cdot\log_b(m)+k$ for constants $c>0$ and k. ■

This lets us prove the sublinear upper bound.

▷ **Lemma 5.10.** For parameters $0 \leq \alpha < 1$ and $b > 1$, let $s,t \in V$ be a start–destination pair and assume that b^+ is the highest expansion between s and t. If $\rho_{s,t}(\alpha,b) < \frac{1-\alpha}{1-\alpha+\alpha\log_b(b^+)}$, then $c_{\text{bi}}(s,t) \in \tilde{O}\big(m^{1-\varepsilon}\big)$ for a constant $\varepsilon > 0$. ◁

Proof. We make a case distinction on the size of the expansion overlap, similarly to the proof of Theorem 5.6. First, we note that by Theorem 5.4, the bidirectional search has sublinear running time if the b-expansion overlap has size $\Omega(\log m)$.

For the other case, we consider an expansion overlap of size less than $c \cdot \log_b(m) + k$, for constants $c > 0$ and k as in Lemma 5.9. We analyze the cost of doing $\text{cheap}_s + T_2$ exploration steps in the forward search and, symmetrically, $(d(s,t) - \text{cheap}_t + 1) + S_2$ in the backward search. By Lemma 5.9, these sequences have sublinear cost, as there is an $x < 1$ such that $c_s([1, \text{cheap}_s + T_2]) \leq 2^{1-\alpha} \cdot m^x$ and $c_t([\text{cheap}_t - S_2, d(s,t)]) \leq 2^{1-\alpha} \cdot m^x$. Without loss of generality, assume $S_1 \geq T_1$. We again consider two cases.

If $\text{expan}_s > \text{cheap}_s + T_2$, the expansion-overlap is reached after the considered sequences of exploration steps. As the cost of these sequences is in $O(m^x)$, we have $\text{cheap}_s(\alpha) + T_2 \leq \text{cheap}_s(x)$, $\text{cheap}_t(\alpha) - S_2 \geq \text{cheap}_t(x)$ and the size of the expansion-overlap is at least the x-relevant distance between s and t. In this situation we have sublinear running time according to Theorem 5.6.

In the other case we have $\text{expan}_s \leq \text{cheap}_s + T_2$. Thus, the considered sequences of exploration steps overlap, as $\text{cheap}_s + T_2 + 1 \geq \text{cheap}_t - S_2$. This means that if the search meets at any layer between $\text{cheap}_s + T_2$ and $\text{cheap}_t - S_2$ we get running time in $O(m^x)$. Consequently, it remains to consider meeting points after $\text{cheap}_s + T_2$ or before $\text{cheap}_t - S_2$. By the definition of S_2 and T_2 we have $\text{cheap}_s + T_2 = \text{expan}_t - 1$ and $\text{cheap}_t - S_2 = \text{expan}_s + 1$. Moreover, by Lemma 5.8, the sequences $[1, \text{expan}_s]$ and $[\text{expan}_t, d(s,t)]$ have length in $O(\log m)$. Thus, if the search meets after $\text{cheap}_s + T_2$ or before $\text{cheap}_t - S_2$, this meeting point differs from a meeting point with running time in $O(m^x)$ only by a logarithmic number of steps. This results in a running time in $\tilde{O}(m^x)$ by Lemma 5.1. ■

The following lemma covers the other side of the dichotomy, by proving a linear lower bound on the running time for the case where the conditions on ρ in Lemma 5.10 are not met. The rough idea is the following. We construct symmetric trees of depth d around s and t. The trees are b-expanding for $(1-\rho)d$ steps and b^+-expanding for subsequent ρd steps and are connected at their deepest layers.

▷ **Lemma 5.11.** For any choice of the parameters $0 < \alpha < 1$, $b^+ > b > 1$, $\rho_{s,}(\alpha, b) \geq \frac{1-\alpha}{1-\alpha+\alpha\log_b(b^+)}$ there is an infinite family of graphs with two designated vertices s and t, such that in the limit $\text{cheap}_s(\alpha)$, $\text{cheap}_t(\alpha)$, $\text{expan}_s(b)$, and $\text{expan}_t(b)$ fit these parameters, b^+ is the highest expansion between s and t and $c_{\text{bi}}(s,t) \in \Theta(m)$. ◁

Proof. We construct such instances by taking two isomorphic trees T_s and T_t with roots s and t and connecting their deepest leaves with a matching. Let $d_1 + d_2 = d$ be the depth of these trees, with $d \in \Theta(\log m)$. The number of branches at each

layer is chosen so that s and t are b-expanding for d_1 steps and then b^+-expanding for the remaining d_2 steps.

In the following, we verify that this construction satisfies our requirements asymptotically. First, note that ignoring constant factors for $i \leq d_1$ we have $c_s(i) \in \Theta\left(b^i\right)$ and for $d_1 < i \leq d_1 + d_2$ we have $c_s(i) \in \Theta\left(b^{d_1} \cdot b^{+i}\right)$. This means that $c_s(d_1 + d_2)$ is linear in the total cost of $\sum_{i=1}^{d_1+d_2} c_s(i)$ and therefore also linear in the total number of edges. This means that the most expensive layers are in the middle, just before the two trees meet. As there are no shortcuts, both the bidirectional search and the unidirectional search have to explore at least one of the two most expensive layers, resulting in linear running time overall.

We now determine a suitable choice of d_1 and d_2. The idea is that from s, we want d (at least) b-expanding layers, followed by d_2 no longer expanding layers. Also, we want the first d_1 layers to have cost at most m^α. In other words, the goal is to have $\text{expan}_s(\alpha, b) = d_1+d_2$ and $\text{cheap}_s(\alpha) = d_1$, and analogously $\text{expan}_t(\alpha, b) = d_1+d_2+1$ and $\text{cheap}_t = d_1 + 2d_2 + 1$. In order to make the construction fit to the definitions, we need to ensure that $c_s(\text{cheap}_s) \leq m^\alpha$ and the length $S_2 = T_2$ is sufficiently small compared to $S_1 = T_1$, that is, $d_2 \leq \rho d$. As the construction is symmetrical, it suffices to check this for s.

For the cost of the first d_1 steps we have $c_s(\text{cheap}_s) \in \Theta(c_s(d_1)) = \Theta\left(b^{d_1}\right)$. Also we have $m \in \Theta(c_s(d_1 + d_2)) = \Theta\left(b^{d_1} \cdot b^{+d_2}\right) = \Theta\left(b^{d_1+d_2 \log_b(b^+)}\right)$. This means that if the exponent of b^{d_1} is at most the exponent of $b^{\alpha\left(d_1+d_2 \log_b(b^+)\right)}$, we get $c_s(\text{cheap}_s) \in \Theta(m^\alpha)$. In order to also get $c_s(\text{cheap}_s) \leq m^\alpha$, we additionally need $b^{d_1+d_2 \log_b(b^+)}$ to be a sufficiently small fraction of m. We ensure this by appending a sufficiently long (linear in the size of the construction) path to some vertex in layer d. This does not asymptotically change the cost of any layer, but makes the number of edges in layer d an arbitrarily small fraction of the total number of edges. It remains to compare the exponents of b^{d_1} and $b^{\alpha\left(d_1+d_2 \log_b(b^+)\right)}$, which allows us to derive the following requirement on α, b, and b^+ subject to d_1 and d_2.

$$\begin{aligned} d_1 &\leq \alpha d_1 + \alpha \log_b(b^+) d_2 \\ d_1(1-\alpha) &\leq \alpha \log_b(b^+) d_2 \\ \frac{d_2}{d_1} &\geq \frac{1-\alpha}{\alpha \log_b(b^+)}. \end{aligned}$$

We now set $d_2 = \rho d$ and $d_1 = (1-\rho)d$. Then, this translates to

$$\frac{d_2}{d_1} = \frac{\rho}{1-\rho} \geq \frac{1-\alpha}{\alpha \log_b(b^+)}.$$

It is easy to verify that $\frac{x}{1-x} \geq y$ and $x \geq \frac{y}{1+y}$ are equivalent. This means we get

$$\rho \geq \frac{(1-\alpha)/(\alpha \log_b(b^+))}{1 + (1-\alpha)/(\alpha \log_b(b^+))} = \frac{1-\alpha}{1-\alpha+\alpha \log_b(b^+)},$$

which matches exactly the claimed requirement for ρ. This means that for any set of parameters with $\rho \geq \frac{1-\alpha}{1-\alpha+\alpha \log_b(b^+)}$, we can construct an instance in which s and t fulfil these parameters and in which the bidirectional search between s and t has linear cost. ■

This lets us state a complete characterization of the worst case running time of bidirectional BFS depending on $\rho_{s,t}(\alpha, b)$. It follows directly from Lemma 5.10 and Lemma 5.11.

▷ **Theorem 5.12.** Let an instance (G, s, t) be a graph with two designated vertices, let b^+ be the highest expansion between s and t and let $0 < \alpha < 1$ and $b > 1$ be parameters. For a family of instances we have $c_{\text{bi}}(s,t) \in O(m^{1-\varepsilon})$ for some constant $\varepsilon > 0$ if $\rho_{s,t}(\alpha, b) < \frac{1-\alpha}{1-\alpha+\alpha \log_b(b^+)}$ and $c_{\text{bi}}(s,t) \in \Theta(m)$ otherwise. ◁

5.4 Empirical Evaluation

A convincing explanation for why the bidirectional BFS is so efficient in practice requires two ingredients. First, we have to make assumptions on the input that are strong enough to allow for a sublinear upper bound, which our theoretical analysis in the previous section provides. Secondly, the assumptions must be justified, i.e., they should hold for many real-world inputs, in particular for those, where the bidirectional BFS is fast. Note that the latter can only be verified empirically. In the following, we evaluate whether the conditions for our theorems are fulfilled on practical instances and if so, to what extend the observed running times align with the theoretical predictions.

This empirical evaluation involves multiple challenges. Very roughly speaking, the theorems given in Section 5.3 are of the form "if condition X holds, then the running time is bounded by Y". Unfortunately, neither X nor Y are trivial to check for any given instance. The difficulty with condition X is that it depends on certain

constants like b (the lower bound for the basis of the exponential expansion). This makes it important to not only select reasonable values for these constants, but to also investigate the robustness with respect to this choice. Concerning the running time Y, the theorems provide asymptotic bounds. However, observing asymptotic behavior on any fixed real-world network is not feasible. To address this, we define an *estimated exponent* for the running time and later validate this definition on a family of generated instances, which allows us to scale the input size.

The remainder of this section is organized as follows. We begin by introducing our dataset and methodology in Section 5.4.1. This includes our choice for the constants as well as the definition of estimated exponent. Section 5.4.2 contains our main results. There we demonstrate that the assumptions underlying our theoretical results hold on many practical instances. Moreover, we show that the observed running times align well with our theoretical predictions, underscoring the practical utility of our analysis. Next, in Section 5.4.3, we consider synthetic input data, which allows us to validate our methodology for distinguishing linear from sub-linear running times. Finally, we conclude with a discussion in Section 5.4.4.

Our experiments are implemented in C++ and R. The code is available on a public GitHub repository.[10]

5.4.1 Dataset and methodology

In this section we introduce the set of real-world networks used for the evaluation, present the *estimated exponent*, as a way to evaluate the running times on these instances, and explain how we evaluate the assumptions made in the theoretical analysis.

Dataset. We use a collection of networks curated by Bläsius and Fischbeck [BF24]. This collection contains 2967 networks with up to 1 M edges from Network Repository [RA15] and is available online [BF22], see also Section 3.4. The networks come from a wide range of domains such as social-, biological-, and infrastructure-networks and have a mean size of 12 166 vertices (median 539) with a mean average degree of 21.8 (median: 5.7) and median clustering coefficient of 0.45. We refer to Bläsius and Fischbeck [BF24] for more details and an analysis of bidirectional BFS performance depending on the locality and heterogeneity of the networks.

Estimated exponent. For each graph we sample 250 start–destination pairs uniformly at random. For each pair we measure the cost of a bidirectional BFS as

10 https://github.com/marcwil/det_bfs_exp

the sum of the degrees of explored vertices. This measure correlates directly with the actual running time measured in seconds, but is much more robust and easier to obtain. Then, assuming that the cost c behaves asymptotically as $c = m^x$, we compute $x = \log_m c$ as a normalization of the cost. We call x the *estimated exponent*. While this measure is not exact, it serves as a practical proxy for asymptotic behavior on individual networks of fixed size, where direct observation is infeasible. See Section 5.4.3 for detailed experiments supporting the validity of this approach.

Properties of expanding search spaces. For each sampled vertex pair we further compute all properties relevant for the evaluation of our theoretical results. Most importantly, this includes the size of the b-expansion overlap, i.e., the number of exploration steps with b-expanding cost in both the direction from s and t. In the condition for sublinear running time given by Theorem 5.4, this is compared with $\log_b(m)$. In the condition given by Theorem 5.6 the size of the expansion overlap is compared to the α-relevant distance, the distance between the vertex pair without prefixes of cost up to m^α. Finally, Theorem 5.12 considers $\rho_{s,t}(\alpha, b)$, the ratio between the number of non-expanding steps before the opposite cheap region and the number of expanding steps. We compute these values for $b \in \{1.1, 1.25, 2, 4\}$ and $\alpha \in \{0, 0.1, 0.2, 0.3, 0.4, 0.5\}$.

5.4.2 Results

Here we present and discuss our main results, i.e., how well our theoretical insights reflect the practical behavior of the bidirectional BFS. We do this in three parts. We start by reporting some numbers on the expansion overlap, which is a fundamental concept in all our theorems. We then give a detailed evaluation of the conditions for sublinarity given by Theorems 5.6 and 5.12. In the last part, we evaluate the specific running time bounds in Theorems 5.4 and 5.6, which go beyond just stating sublinearity.

Expansion overlap. All our theoretical results rely, directly or indirectly, on the b-expansion overlap. In fact a positive overlap is sufficient such that a constant can be chosen such that Theorem 5.4 applies and an expansion overlap of at least a constant fraction of the distance between the considered vertex pair means that Theorems 5.6 and 5.12 can be applied. Table 5.1 shows the share of vertex pairs for which this holds, split by their estimated exponents. Recall that we sample 250 start–destination pairs for each of the 2967 graphs, i.e., Table 5.1 is based on almost 750 k shortest-path computations.

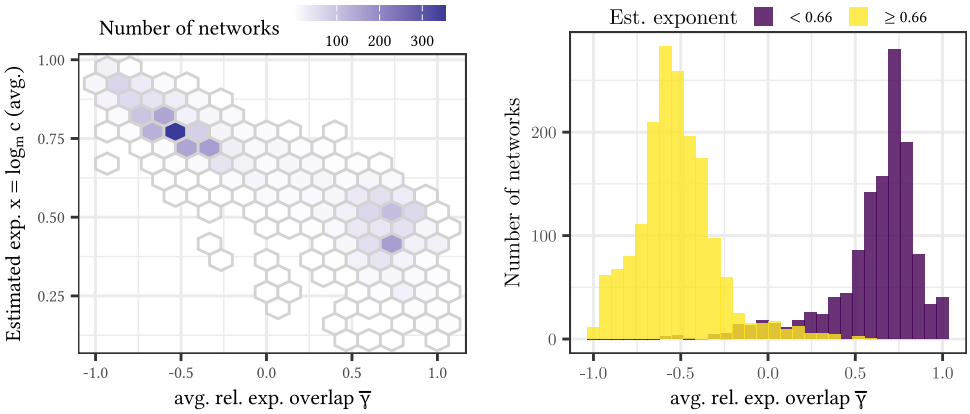
Table 5.1: Statistics for b-expansion overlap. We report the overall share of sampled vertex pairs (s, t) with positive expansion overlap (i.e., $d^{\text{overlap}} > 0$) and with overlap at least half the distance (i.e., $d^{\text{overlap}} > \frac{d(s,t)}{2}$) among vertex pairs with estimated exponent below and above, respectively, the median estimated exponent of 0.64.

estimated exponent	b				
x	1.1	1.25	1.5	2	4
	Share of vertex pairs with $d^{\text{overlap}} > 0$				
$x > 0.64$	34.6%	22.6%	12.6%	6.9%	1.1%
$x \leq 0.64$	96.9%	95.2%	91.7%	84.8%	65.1%
	Share of vertex pairs (s, t) with $d^{\text{overlap}} > \frac{d(s,t)}{2}$				
$x > 0.64$	18.2%	10.5%	5.4%	2.2%	0.2%
$x \leq 0.64$	89.4%	85.8%	79.7%	69.3%	41.5%

We see that the assumptions underlying our theoretical analysis are fulfilled on many of the considered instances. Moreover, instances where a small estimated exponent indicates sublinear running time exhibit positive or large expansion overlap much more often. This fits well with the expectations arising from the theoretical analysis.

Conditions for sublinearity of Theorems 5.6 and 5.12. Now we examine the relationship between observed running times and the conditions for sublinear running time as given by Theorems 5.6 and 5.12. For both theorems, we evaluate a binary classifier and show the joint distribution between estimated exponent and the theoretical condition.

For Theorem 5.6 we do the following. For every sampled vertex pair, we compute the *relative expansion overlap* $\gamma = d^{\text{overlap}}/d_\alpha(s, t)$. We then take the average over all pairs in the same graph, yielding one number for each graph, which we call *average relative expansion overlap* and denote it by $\bar{\gamma}$. Figure 5.3 (a) shows the relationship between the relative expansion overlap and estimated exponent for $b = 2$ and $\alpha = 0$. We find that most networks fall into two groups. The first group has $\bar{\gamma}$ around -0.5 and higher estimated exponent, while the second group has $\bar{\gamma}$ around 0.7 and lower estimated exponent. This indicates that the running time of the bidirectional search is more evidently sublinear on networks where the average vertex pair fulfills the condition of Theorem 5.6. To quantify this relationship, we create a simple binary classifier that classifies networks as sublinear for positive $\bar{\gamma}$. To evaluate whether it



(a) Hexagonal binning plot showing the number of networks with different combinations of γ and estimated exponent.

(b) Histogram showing the number of networks for different values of γ, split by estimated exponent (< 0.66 in purple, ≥ 0.66 in yellow).

Figure 5.3: Distribution of relative expansion overlap γ from Theorem 5.6 for graphs with different estimated exponents under $b = 2$ and $\alpha = 0$.

classifies a give graph correctly, we have to define a cutoff exponent x^* below which a graph counts as sublinear, i.e., a graph with estimated exponent x is counted as sublinear if and only if $x < x^*$. We pick this cutoff such that the F1-score is maximized. For some values of b and α, this yields an F1-score as high as 0.954. Figure 5.3 (b) shows the distribution of $\bar{\gamma}$ for graphs with estimated exponent below and above the cutoff.

Table 5.2 shows the F1-scores and optimal cutoff for a range of values for α and b. We find that the F1-score varies only slightly across different parameter values, but we observe that the optimal cutoff systematically decreases for larger values of b. This indicates that only networks with more strongly sublinear running times exhibit b-expansion overlaps with large basis.

Next, we consider Theorem 5.12. Here, the condition for sublinear running time is that $\rho_{s,t}(\alpha, b) = \frac{\max(S_2,T_2)}{\min(S_1,T_1)}$ needs to be smaller than $\rho_{\text{thresh}} = \frac{1-\alpha}{1-\alpha+\alpha\log_b(b^+)}$, see also Equation (5.1) and Figure 5.2. To measure how much slack this inequality has (or how strongly it is not satisfied), we compute $\delta_\rho = \log(\rho_{\text{thresh}}) - \log(\rho_{s,t}(\alpha, b))$. Note that positive δ_ρ values indicate sublinearity. In $\rho_{s,t}(\alpha, b)$, both numerator and denominator may be 0, which we handle by assuming $\log(a/0) = \infty$ (for $a > 0$), $\log(0) = \log(0/0) = -\infty$. This is consistent with the property that δ_ρ is positive if and only if the conditions of Theorem 5.12 are satisfied. To aggregate over the different vertex pairs for each graph, we take the median of the δ_ρ values. We call

Table 5.2: Summary of F1-score for binary classifier predicting small (≤ cutoff) estimated exponent based on positive expansion overlap for different values of α and b.

α	F1-score, varying b					Cutoff for est. exp., varying b				
	1.1	1.25	1.5	2	4	1.1	1.25	1.5	2	4
0	0.920	0.931	0.951	0.954	0.895	0.781	0.745	0.694	0.660	0.570
0.1	0.919	0.930	0.951	0.954	0.896	0.781	0.745	0.694	0.660	0.570
0.2	0.920	0.930	0.951	0.953	0.898	0.781	0.745	0.694	0.660	0.570
0.3	0.920	0.929	0.950	0.954	0.902	0.781	0.745	0.694	0.660	0.570
0.4	0.918	0.928	0.951	0.953	0.903	0.781	0.745	0.694	0.660	0.570
0.5	0.917	0.930	0.951	0.956	0.913	0.781	0.743	0.694	0.662	0.570

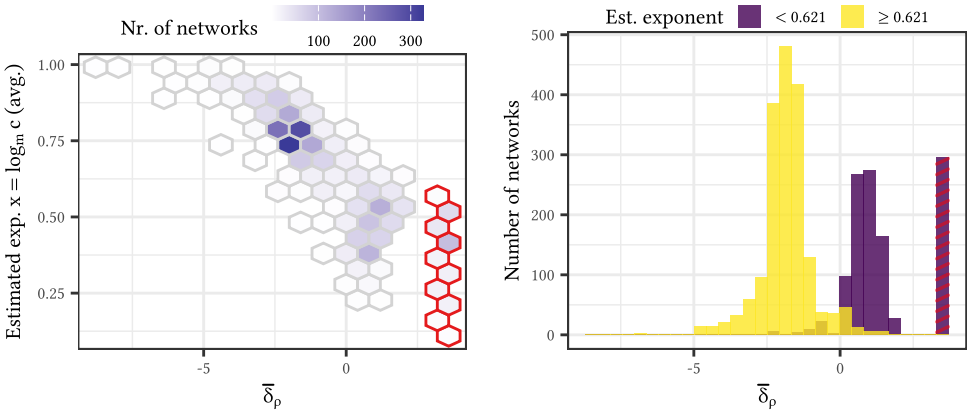
Table 5.3: Summary of F1-score for binary classifier predicting small (≤ cutoff) estimated exponent based on positive median slack $\bar{\delta}_\rho$.

α	F1-score, varying b					Cutoff for est. exp., varying b				
	1.1	1.25	1.5	2	4	1.1	1.25	1.5	2	4
0	0.913	0.940	0.941	0.927	0.798	0.704	0.671	0.641	0.621	0.529
0.1	0.875	0.898	0.921	0.921	0.804	0.619	0.632	0.636	0.615	0.529
0.2	0.941	0.930	0.922	0.905	0.813	0.622	0.632	0.636	0.614	0.529
0.3	0.949	0.949	0.935	0.914	0.828	0.646	0.632	0.631	0.614	0.529
0.4	0.951	0.952	0.955	0.947	0.863	0.663	0.632	0.636	0.614	0.529
0.5	0.955	0.961	0.966	0.961	0.902	0.663	0.666	0.643	0.614	0.529

the resulting value the *median slack* and denote it with $\bar{\delta}_\rho$. Figure 5.4 (a) shows the relationship between the median slack $\bar{\delta}_\rho$ and the estimated exponent for $b = 2$ and $\alpha = 0$.

We evaluate a binary classifier predicting sublinearity if and only if $\bar{\delta}_\rho > 0$. As before, we need a cutoff x^* for the estimated exponent, below which a graph counts as sublinear. We choose x^* such that the F1-score is maximal. For $b = 2$ and $\alpha = 0$, this results in an F1-score of 0.927 with $x^* = 0.621$. We also include Table 5.3 with results for a range of values for b. The performance of the classifier is similar to the one for Theorem 5.6, with slightly larger variation depending on α.

Running time bounds of Theorems 5.4 and 5.6. In addition to giving a criterion for sublinear running time, Theorems 5.4 and 5.6 also give closed formulas for the exponent of the running time. We evaluate how these theoretical worst-case



(a) Hexagonal binning plot showing the number of networks for different combinations of median slack $\bar{\delta}_\rho$ and estimated exponent.

(b) Histogram showing the number of networks for different values of median slack $\bar{\delta}_\rho$, split by estimated exponent.

Figure 5.4: Distribution of the median slack $\bar{\delta}_\rho$ for graphs with different estimated exponents under $b = 2$ and $\alpha = 0$. Networks where the median slack is infinite are shown next to the distribution of finite values and marked in red.

guarantees for the exponent align with the estimated exponent. The exponents are $1 - \frac{\gamma}{2}$ for Theorem 5.4, where γ is the size of the b-expansion overlap divided by $\log_b(m)$, and $1 - \frac{\gamma(1-\alpha)}{2(\log_b(b^+)+\gamma)}$ for Theorem 5.6, where c is the size of the b-expansion overlap divided by the $d_\alpha(s,t)$. We compute both terms for all sampled (s,t)-pairs using $b = 2$ and $\alpha = 0$ and set any value larger than 1 to 1. For each network, we then compare the mean of these bounds for the exponent with the mean estimated exponent.

For the bounds of Theorem 5.4 we observe a Pearson correlation of 0.860 and a Spearman correlation of 0.903. For the bounds of Theorem 5.6 these values are 0.751 and 0.869. The high Spearman correlations strongly indicate a monotonic relationship, while the Pearson correlations suggest a moderately linear relationship. Table 5.4 shows correlation values for different values of α and b. Figure 5.5 shows the joint distribution of the mean estimated exponent and mean predicted exponents across the dataset. Additionally, both plots also show a fitted linear model. The models achieve a residual standard error of 0.090 (0.117) when applied to the data of Theorem 5.4 (respectively Theorem 5.6). Further, 77.0% (respectively 66.4%) of networks fall within 0.10 of the predicted values. This further supports a linear relationship between the two variables. Interestingly, the bounds of Theorem 5.4 are much more pessimistic than those of Theorem 5.6. At the same time, the observed

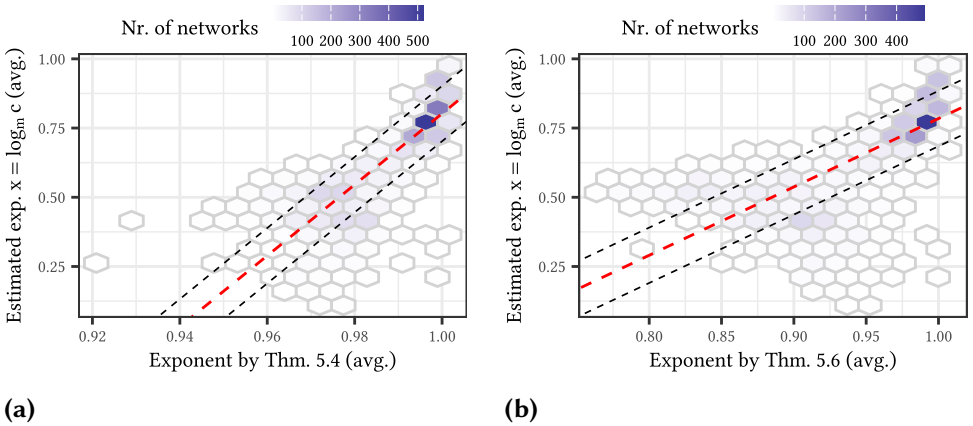


Figure 5.5: Relationship between estimated exponents and theoretically upper bounds of Theorem 5.4 (a) and Theorem 5.6 (b). The plots include the fitted linear model (red) and versions of the model shifted up and down by 0.1 (black, thinner).

linear relationship between the bound and the estimated exponents is stronger for Theorem 5.4.

As a final thought, we want to note that the bounds of the theorems are worst-case bounds. Thus, even though they take the structure of the graph into account, they are still pessimistic and one can see in Figure 5.5 that the practical running times are indeed much better than our upper bounds. We think that this makes it even more interesting that the pessimistic upper bounds are so strongly correlated with the observed running times.

5.4.3 Scaling Experiments

In the previous experiments, we used the estimated exponent to identify networks with supposedly sublinear running time for the bidirectional BFS. In the following we do experiments on synthetic instances of varying size, that justify the use of the estimated exponent for this purpose.

For this, we use geometric inhomogeneous random graphs (GIRGs) [BKL19; Blä+22c]. GIRGs combine an underlying geometry facilitating community structure with a power-law degree distribution. The heterogeneity of the degree distribution can be adjusted by changing the power-law exponent τ. For $\tau \in (2, 3]$, the variance of the degree distribution is unbounded. For $\tau \to \infty$, the degree distribution becomes uniform. For hyperbolic random graphs, a model closely related to GIRGs, the bidirectional BFS is known to be sublinear for $\tau \in (2, 3)$ with high probability [Blä+22b]. Thus, varying τ yields a diverse set of graphs in regards to the

Table 5.4: Correlations between average estimated exponents and upper bounds of Theorems 5.4 and 5.6 for different values of b (for Theorem 5.4), respectively α and b (for Theorem 5.6).

α	Pearson cor., varying b					Spearman cor., varying b				
	1.1	1.25	1.5	2	4	1.1	1.25	1.5	2	4
	Correlations for Theorem 5.4									
0.00	0.722	0.792	0.838	0.860	0.808	0.826	0.884	0.905	0.903	0.847
	Correlations for Theorem 5.6									
0.00	0.453	0.606	0.698	0.751	0.340	0.583	0.775	0.854	0.869	0.692
0.10	0.470	0.618	0.704	0.754	0.316	0.602	0.786	0.857	0.870	0.693
0.20	0.480	0.624	0.706	0.752	0.314	0.611	0.790	0.858	0.869	0.703
0.30	0.446	0.604	0.698	0.755	0.305	0.565	0.762	0.849	0.873	0.711
0.40	0.436	0.597	0.695	0.756	0.287	0.559	0.752	0.842	0.869	0.726
0.50	0.422	0.590	0.693	0.757	0.256	0.539	0.744	0.841	0.869	0.731

performance of the bidirectional BFS. We generate GIRGs a varying number of vertices n and with 8 different power-law exponents $\tau \in \{2.1, 2.3, 2.5, 2.7, 3, 3.5, 6, 10\}$. All other parameters of the model are kept constant (average degree 10, dimension 2, temperature 0). For each parameter configuration we sample five networks, resulting in 880 synthetic graphs in total. As in Section 5.4.2, we 250 vertex pairs for each graph.

Figure 5.6 shows the average estimated exponent for networks of growing size across the different power-law exponents. To interpret this, recall that the estimated exponent is based on the assumption that the cost c behaves like $c = m^x$. Rearranging for x is asymptotically robust with respect to constant factors (and thus also lower order terms) in the following sense. If the actual running time was not m^x but $c = a \cdot m^x$ for some constant a, then $x = \log_m c - \log_m a$. As $\log_m a$ converges to 0 for $m \to \infty$, $x = \log_m c$ yields the correct exponent, assuming that m is sufficiently large. Thus, when scaling the graph size as in Figure 5.6, we expect the estimated exponent to converge to the correct exponent.

For small power-law exponents $\tau \in (2, 3]$, we can see in Figure 5.6 that, although there is some random noise, the estimated exponent does not change much for varying graph sizes. This indicates that the estimated exponent is already a good approximation for small graphs. For $\tau > 3$, there appears to be less noise but the convergence is slower, i.e., we need sufficiently large graphs to get an estimated exponent $x \approx 1$ for graph families with linear running time. Nonetheless, there is a clear gap between the families with sublinear running time (small τ) and linear

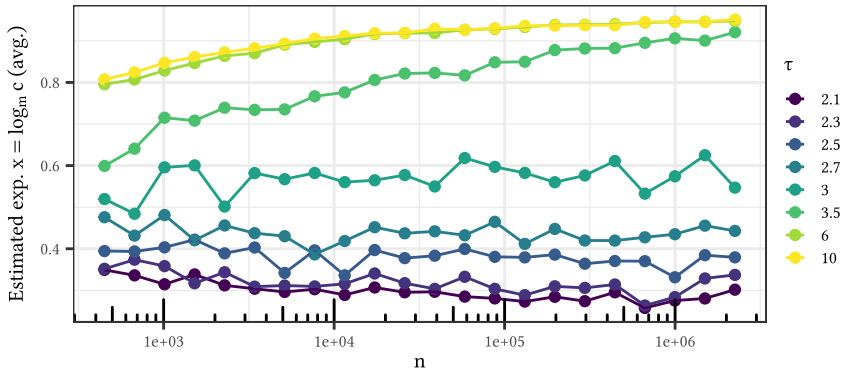


Figure 5.6: Estimated exponent (mean) of GIRGs with different size and power-law exponent.

running time ($\tau > 3$), even for small graphs. This justifies that the estimated exponent is a reasonable measure to distinguish linear form sub-linear running time. Moreover, it explains why our classifier yields cutoffs around $x = 0.6$, while something like $m^{0.7}$ would, from a theoretical standpoint, clearly justify as being sublinear.

5.4.4 Discussion

With the experiments in this section, we have seen that our criteria for sublinear running time do not only yield theoretic upper bounds, but are also satisfied by many real-world networks. In fact, we can use our criteria for a classification into linear and sublinear, which very much agrees with the observed running times. This indicates that our criteria are not only sufficient for a sublinear running time (as asserted by the theorems) but additionally provide a good characterization of the circumstances, under which the bidirectional BFS is efficient in practice.

The one downside of Theorems 5.6 and 5.12 is that they are somewhat technical. We leave it as an open question, whether there are simpler criteria that capture the behavior of the bidirectional BFS similarly well. In this regard, we want to briefly discuss the diameter as well as locality and heterogeneity as considered by Bläsius and Fischbeck [BF24]. Roughly speaking, the bidirectional BFS performs well except for graphs with high locality and low heterogeneity [BF24]. Similarly, these tend to be the graphs with low (i.e., poly-logarithmic) diameter. However, we want to stress that these observations are purely empirical. There are in particular no performance guarantees and one can easily construct graph families with small

diameter or high heterogeneity, for which the bidirectional BFS has linear running time.

Nonetheless, we did experiments to evaluate whether locality and heterogeneity or the diameter can be used to classify practical inputs. For this, we have to choose a cutoff x^* for the estimated exponent x to separate the graphs into linear ($x \geq x^*$) and sublinear ($x < x^*$). Assume for now, that we have fixed this cutoff x^*. For the classification with locality and heterogeneity, we use linear regression to predict whether $x < x^*$ based on the locality and heterogeneity values. For the diameter d, we do not use d directly but normalize it, i.e., we do the classification based on the parameter $\log_n d$, which is a constant if the diameter is large (e.g., $\log_n d = 0.5$ if $d = \sqrt{n}$) and tends to 0 if the diameter is sub-polynomial (e.g., if $d = \log n$). For both cases, we choose the exponent cutoff $x^* \in [0.4, 0.75]$ such that it yields the best F1 score.[11]

This yields an F1-score of 0.869 for locality and heterogeneity, and 0.901 for the diameter). Thus, not only are these parameter lacking provable performance guarantees, but they also give worse predictions than our criteria. Nonetheless, they are simple and closely related to the running time.

5.5 Conclusion and Open Problems

In this chapter, we defined the expansion overlap and other parameters related to the exponential expansion of neighborhood balls and used them to give sufficient and partly necessary conditions for sublinear running time of the balanced bidirectional search. Our subsequent empirical evaluation shows that our theoretical conditions also capture practically observed running times. We conclude this chapter with a brief discussion of open questions.

First, can the perspective presented in this chapter be extended to settings with super-exponential expansion of BFS layer costs? On a number of network models such as Chung-Lu random graphs, HRGs, and GIRGs the average distance between vertices is doubly-logarithmic [AFB17; BKL25; CL02b]. Thus on such networks one can expect a constant growth rate to underestimate the expansion of neighborhood balls. Note that only Theorem 5.4 applies to non-constant expansion while Theorem 5.6 and Theorem 5.12 rely on an upper bound for the maximum growth between two subsequent BFS steps.

Moreover, our conditions for sublinear running time are formulated for instances consisting of a graph and an (s, t) pair. Is it possible to give natural and empirically

11 This restriction to an interval is here to prevent choosing $x^* = 0$ or $x^* = 1$ and then just classify all graphs as linear or all graphs as sublinear, respectively.

grounded conditions that guarantee sublinear running time for all or at least most pairs of vertices in a graph? Clearly, one can consider the conjunction of our conditions over all pairs of vertices, but maybe more natural properties of a graph can be specified? One relevant direction in this context are expander graphs, as for example recently used by Alon, Grønlund, Jørgensen, and Larsen [Alo+24].

6 Diameter Computation in (Random) Geometric Graphs

This chapter is based on joint work with Thomas Bläsius and Annemarie Schaub [BSW26]. The project was motivated by empirical results about the performance of a practical algorithm for the diameter problem on real-world networks with torus-like geometry [BF24]. The presented algorithm extends upon a basic idea for a separator-based approach that Annemarie Schaub analyzed on toroidal random geometric graphs in her bachelor's thesis [Sch24].

We finally apply the deterministic properties methodology to the problem of computing the graph diameter in a geometric setting. This way, we complement the previous chapter's analysis of a known algorithm with the design of a new algorithmic approach. Concretely, we define a set of deterministic properties formalizing combinatorial features a graph plausibly inherits from an underlying geometry and show that they can be exploited algorithmically. Roughly speaking, we specify the size of (recursive) balanced separators that divide the graph into approximately ball-shaped subgraphs in addition to some properties of highly distant vertex pairs. We argue that our deterministic properties are well motivated on an intuitive level, but additionally verify them probabilistically on random geometric graphs. This way, our properties can be seen as forming an interface between an average-case analysis and the algorithm itself, explicitly making it transparent which properties of the underlying probability distribution are exploited.

Our analysis is motivated from two perspectives: complementing previous work that focused on heterogeneous networks with a geometric viewpoint, and addressing the empirically observed difficulty of diameter computation on graphs with torus-like geometry. We formulate our properties to capture both periodic and aperiodic geometry and verify them on random geometric graphs in both settings.

6.1 Introduction

The *diameter*, i.e., the maximum distance between any pair of vertices, is one of the most fundamental graph parameters. It is relevant for numerous applications for example in network design [Chu86; PY00; XKY04], distributed systems [AM05;

Cha+07; LCW05] and graph clustering [Sch07]. A simple algorithm to compute the diameter of a graph is to perform a breadth-first search (BFS) from every vertex, taking $O(nm)$ time on a graph with n vertices and m edges. The iFUB algorithm (short for iterative fringe upper bound) [Cre+13] constitutes a notable improvement over this approach in practice. Despite a $\Theta(nm)$ worst-case running time, it is often much faster on real-world inputs, especially on complex scale-free networks [BCT17] and graphs with underlying geometry [BF24].

The core intuition behind iFUB is that on many real-world networks there is a meaningful notion of center and periphery (or fringe). More precisely, vertices in the center have smaller distances to most other vertices than vertices in the periphery. Moreover, distant pairs of vertices always lie in the periphery and their shortest paths are (roughly) bisected by the center. The iFUB algorithm exploits this structure by heuristically choosing a vertex in the center and then restricting the search for diametric vertices to vertices that are sufficiently far from this center. As a result, the algorithm only executes a constant number of breadth-first searches in order to select a central vertex and afterwards only performs a BFS for vertices that have distance at least half the diameter from this central vertex. On graphs with a strong center–periphery structure, where diametric paths are indeed approximately halved, this results in a low number of BFS runs.

An extensive empirical study confirms that many real-world networks exhibit a sufficiently pronounced center–periphery structure for iFUB to achieve sublinear running times in practice [BF24]. In particular, the study identifies two regimes of such networks. The first consists of graphs with a strongly heterogeneous degree distribution, i.e., scale-free networks. For this setting Borassi, Crescenzi, and Trevisan [BCT17] prove running time bounds for iFUB under the assumption of a power-law degree distribution together with independently sampled edges. The second regime are graphs with a homogeneous degree distribution and high locality, i.e., with an underlying geometric structure. While it might seem intuitive that such graphs are benign for iFUB, the authors of [BF24] also identify some notable exceptions. These are graphs with a clearly apparent geometric structure, but a periodic geometric ground space, where distances "wrap around" like on a flat torus or a spherical surface. Intuitively, such graphs have neither center nor periphery, thus any chosen central vertex splits some diametric paths very unevenly, and iFUB needs to explore a large portion of the graph in order to find the diameter. To the best of our knowledge, the performance of iFUB on geometric graphs has not yet been formally analyzed. In particular, the conjecture that it benefits from aperiodic geometry but deteriorates on periodic geometry has not been studied from a rigorous theoretical perspective.

In this work, we provide the first theoretical explanation of this behavior by

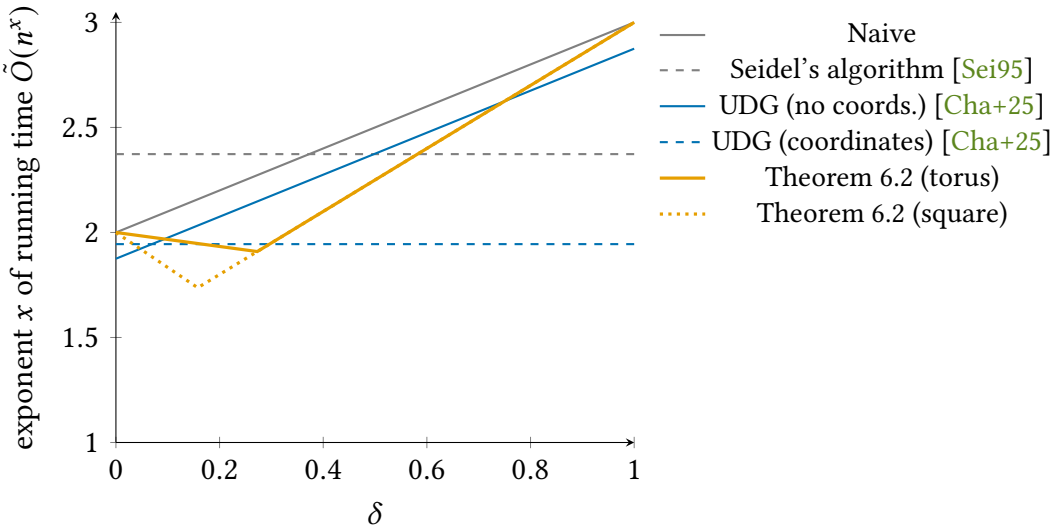


Figure 6.1: Running times on torus/square RGGs with expected average degree in $\Theta(n^\delta)$ for constant $\delta \in (0,1)$. Our algorithm is compared with the naive $O(nm)$ running time, Seidel's matrix-multiplication based $\tilde{O}(n^\omega)$ approach, and the unit-disk graph algorithms from [Cha+25] in both variants with a geometric representation and without.

analyzing iFUB's performance on random geometric graphs (RGGs). We show that on RGGs with a square ground space, iFUB achieves an asymptotic speed-up when choosing a central vertex using the so-called 2-sweep heuristic and that this is not the case on RGGs with a flat torus as ground space.

▷ **Theorem 6.1.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a square random geometric graph with expected average degree $d \in \Omega(\log^{\frac{3}{2}} n)$. Then, a.a.s., 2-sweep iFUB has running time in $O((nd^{-\frac{2}{3}} + \log n)m)$. If $G \sim \mathcal{G}(\mathcal{T}, n, r)$ is a torus RGG with expected average degree in $\Omega(\log^{\frac{3}{2}} n)$, then a.a.s. for every choice of the central vertex, the running time of iFUB is in $\Omega(nm)$. ◁

This raises the question of whether the resulting running time in $\Omega(nm)$ is inherent to graphs with a torus-like geometry or whether better algorithmic approaches are possible. In the following theorem, we give a positive answer by showing that a polynomial improvement to this is possible. Still, for square RGGs we achieve an even better running time exponent.[12]

▷ **Theorem 6.2.** On RGGs with expected average degree $\Theta(n^\delta)$ for constant $\delta \in (0,1)$, asymptotically almost surely the diameter can be computed in $\tilde{O}(n^{\frac{3}{2}(1+\delta)} +$

12 We use $\tilde{O}$-notation to hide poly-logarithmic factors in the running time and O^* for $n^{o(1)}$ factors.

$n^{2-\frac{1}{3}\delta})$ time for torus RGGs, respectively $\tilde{O}(n^{\frac{3}{2}(1+\delta)} + n^{2-\frac{5}{3}\delta})$ time for square RGGs. ◁

We note that in the above theorem the probabilistic statement only concerns drawing the random geometric graph; the algorithm itself is fully deterministic and always computes the diameter correctly. To the best of our knowledge these are the first running time bounds for the diameter problem specifically on random geometric graphs. For an overview of how they compare to known running times for diameter computation on related graph classes, see Figure 6.1. Note that specifically the class of unit disk graphs makes for a suitable comparison as it can be seen as a deterministic worst-case variant of (square) random geometric graphs. Here, the fastest known running time is in $O^*(n^{2-1/18})$ as shown recently by Chan, Chang, Gao, Kisfaludi-Bak, Le, and Zheng [Cha+25], with O^*-notation hiding $n^{o(1)}$ factors. Our running time on square RGGs gives a polynomial improvement upon this for average degrees $\Theta(n^\delta)$ with constant δ strictly between $1/30 \approx 0.033$ and $8/27 \approx 0.296$. Even our running time on torus RGGs gives a polynomial improvement for d strictly between $1/6 \approx 0.167$ and $8/27 \approx 0.296$. Additionally, we note that the $O^*(n^{2-1/18})$ algorithm depends on a coordinate representation of the graphs, which is $\exists\mathbb{R}$-hard to obtain [KM12]. A variant of their algorithm that only requires the graph as input runs in $\tilde{O}(mn^{1-1/8})$ time [Cha+25]. Compared to this, our algorithm on degree $\Theta(n^\delta)$ random geometric graphs is faster for $\delta \geq \frac{3}{64} \approx 0.047$ (square RGGs), respectively $\delta \geq \frac{3}{32} \approx 0.094$ (torus RGGs). Concerning the requirements on the input, we note that our algorithm lies between these two variants. We only use the coordinates implicitly in the sense that we require a hierarchy of separators that is straightforward to compute given coordinates. But in principle, the separator hierarchy could be computed in a different way without having the coordinates as an intermediate step.

While the algorithms stated in Theorem 6.2 are interesting contributions on their own, they are only an application of a more general framework. Our main contribution is to distill a set of deterministic graph properties and to prove that they enable efficient diameter computation. The running times on random geometric graphs then follow, by proving that these graphs a.a.s. fit into our framework.

To introduce our framework and motivate the properties it is based on, we start by making a few obvious observations about geometric ground spaces (specifically square and flat torus) and distances therein; also see Figure 6.2. Observations 1 and 2 are related to *diametric pairs*, where a pair of points is diametric if the distance between them is the diameter of the ground space. Moreover, a pair is *x-diametric* if their distance is at most x smaller than the diameter. If two points p and q form a (x)-diametric pair, we say that p is a *(x)-diametric partner* of q and vice versa.

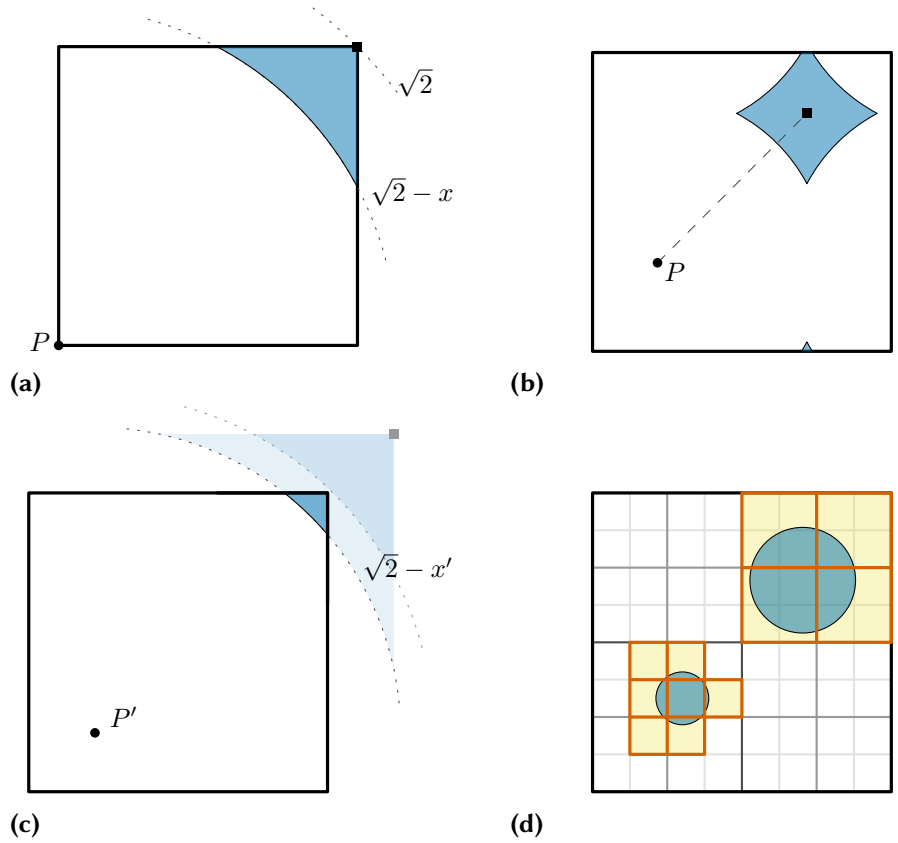


Figure 6.2: Visualization of the observations. Regarding Observation 1, Part (a) shows a point P on the unit square, its unique diametric partner at distance $\sqrt{2}$ (black square); the blue region of all x-diametric partners of P for $x = \sqrt{2}/5$ is clearly not much larger than x. Part (b) shows an analogous situation on the torus; note that here, any point has a diametric partner. Part (c) visualizes the second part of Observation 2: a point P' sufficiently far from any corner of the unit-square has no x-diametric partners for $x = \sqrt{2}/5$; still, for a larger $x' > x$ the region of x'-diametric partners is non-empty. Regarding Observations 3–5, Part (d) shows a hierarchical subdivision of the square/torus ground space and two differently sized blue disks that intersect only few cells with similar diameter (marked in yellow).

Observations 3–5 are based on partitioning the ground space like with a quad-tree into a hierarchy of cells. Observe the following.

1. For every point on a torus or square and $x > 0$, the x-diametric partners lie within a disk of radius $O(x)$. **(local diametric partners)**
2. In a square, only the four corners have a diametric partner. Moreover, for every $x > 0$ all points with x-diametric partners lie inside 4 disks of radius $O(x)$. **(few corners)**
3. The boundary of each cell is small compared to its area. **(small separators)**
4. Cells with smaller area have smaller diameter and vice versa. **(size-dependent diameters)**
5. Every disk in the ground space intersects only few cells that have diameter similar to the disk. **(low fragmentation)**

We now translate these geometric observations into graph properties. For this, the challenge is to strike a balance between introducing enough flexibility in order to include a meaningful class of graphs and being strong enough to allow algorithmic improvements. In the following, notions related to distances refer to graph distances, e.g., a ball of radius r is the set of vertices with graph distance at most r from some central vertex. To translate the hierarchical partitioning of the geometric ground space, we introduce *recursive partitions* of a graph into *blocks*. We assume that recursive partitions have constant *branching factor*, i.e., each block has only a constant number of children. We first state the five properties and discuss them below.

1. **d_{local}-local diametric partners:** For every vertex v and $x > 0$, the x-diametric partners of v can be covered with $O(1)$ balls of radius $O(x + d_{\text{local}})$.
2. **d_{corner}-few corners:** For every $x > 0$, all vertices with an x-diametric partner can be covered with $O(1)$ balls of radius $O(x + d_{\text{corner}})$.
3. **(α, β)-small separators:** The separator of each block with k vertices has size $O(k^\alpha n^\beta)$.
4. **size-dependent diameters:** For all blocks A and B with diameters D_A and D_B, $|A| \in O(|B|)$ implies $D_A \in O(D_B)$, and vice versa.[13]

13 We note that the intuitive interpretation of asymptotics is correct here: O-notation hides universal constants that do not depend on individual instances. For Properties 3 and 4 it is also important that the blocks are only compared per instance and not across instances. However, as this makes the formal definitions slightly tricky, we provide a full explanation in Section 6.2.1.

5. **low fragmentation:** Every ball of radius r intersects a constant number of blocks of diameter $\Theta(r)$.

Each property directly corresponds to the observation with the same number, but differs in a few key ways. In Property 1 and Observation 1 we allow $O(1)$ balls and keep the linear dependency of their radius on x. Additionally we introduce two parameters d_{local} and d_{corner} that allow slack for small x. Property 3 uses two parameters to specify the size of separators depending not only on the block size, but also on the size of the whole graph. This is important to also capture intersection graphs of objects with size growing in n. Finally, Properties 4 and 5 are the same as Observations 4 and 5, except that we require the bounds to only hold up to constant factors. We call a recursive partition *(α, β)-well-spaced*, if Properties 3–5 hold and it is *balanced*, i.e., if for each block the size of any two children differs only by a constant factor. Intuitively, a well-spaced recursive partition uses balanced sublinear separators that divide the graph into roughly ball-shaped subgraphs.

We are almost ready to state our main theorem. We say that a recursive partition has leaf-block size at most k_{leaf} if every leaf-block has at most k_{leaf} vertices. Further, the degeneracy of a graph is the smallest d such that every subgraph has a vertex with degree at most d.

▷ **Theorem 6.3.** Let G be a n-vertex graph with degeneracy d satisfying Property 1. Let $\mathcal{P}$ be a (α, β)-well-spaced recursive partition with leaf-block size k_{leaf}. For every $k \geq k_{\text{leaf}}$ such that $k \in o(n)$ and blocks of size $\Theta(k)$ have diameter $\Omega(d_{\text{local}})$, our algorithm computes the diameter of G in time

$$\tilde{O}\Big(n^{1+\alpha+\beta} \cdot d + \min\big\{nkd + k^{2\alpha}n^{1+2\beta} + k^{2\alpha-1}n^{1+\alpha+3\beta},\ kn^{1+\alpha+\beta}\big\}\Big).$$

If also Property 2 holds and blocks of size $\Theta(k)$ have diameter $\Omega(d_{\text{corner}})$, it runs in time

$$\tilde{O}\Big(n^{1+\alpha+\beta} \cdot d + \min\big\{k^2d + k^{1+2\alpha}n^{2\beta} + k^{2\alpha}n^{\alpha+3\beta},\ k^2n^{\alpha+\beta}\big\}\Big).$$

◁

We note that the above bounds on the running time of our algorithm hold for every k satisfying the requirements, i.e., our algorithm implicitly chooses k such that the running time is minimized. Note that, unless $\alpha < 1/2$, the running time is monotone in k. It thus makes sense to think of k as small as possible such that the diameter of size $\Theta(k)$ block is still in $\Omega(d_{\text{corner}})$.

Outline. The remainder of this chapter is structured as follows. We introduce important definitions and notation in Section 6.2 and provide alternative verbose

definitions of Properties 1–5 with explicitly spelled out asymptotics. Section 6.3 then presents our algorithm, while Section 6.4 contains our analysis on random geometric graphs. We finally discuss generalizations of our parameters and directions for future work in Section 6.5.

6.2 Preliminaries

The *eccentricity* of a vertex v, written $\mathrm{ecc}_G(v)$, is the maximum distance between v and any other vertex of G. The *diameter* of G, written diam_G is the largest eccentricity of any vertex. We write $\mathrm{maxdist}_G(A, B) = \max\{d_G(a, b) \mid a \in A, b \in B\}$ for the maximum distance between two vertex sets $A, B \subseteq V(G)$. We call two vertices (respectively, a path) *diametric* if their distance (respectively, its length) is diam_G. We also extend the notions of eccentricity and diameter to *metric spaces* in general. For a metric space M consisting of a set of elements X and a distance function d_M, we define the *M-ball* of radius r around an element $e \in X$ as the set of elements $e' \in X$ with $d_M(e, e') \leq r$. Note that on the metric space induced by a graph this is equal to the closed neighborhood. We say that a set of elements $Y \subseteq X$ can be *covered* by a ball of radius r, if there is an element $e \in X$ such that Y is contained in the ball of radius r around e.

Let G be connected. We define a *recursive partition* $\mathcal{P} = (T, \{B_v\}_{v \in V(T)})$ of G as a rooted tree T where each *node* $v \in V(T)$ is associated with a connected set of vertices $B_v \subseteq V(G)$. Note that we call the vertices of T *nodes* to distinguish them from vertices of G. The vertex sets $\{B_v\}_{v \in V(T)}$ are called *blocks*, i.e., B_v is the *block* of v. We require the blocks of the leaves of T to form a partition of $V(G)$. Moreover, for each non-leaf node u of T the block B_u is equal to the union of the blocks of leaf nodes in the subtree below u. Note that this implies $B_r = V(G)$ for the root node r of T. We further require each non-leaf node has at least two children and at most a constant number of children (constant branching factor).

Note that the tree T is uniquely defined by the set of blocks and we thus often use $\mathcal{P}$ as a set of blocks and write $B \in \mathcal{P}$ for a block B. We call a block $B_v \in \mathcal{P}$ a parent of a block B_w, if v is a parent of w in T. This lets us also use the relations of parent, descendant, ancestor, and leaf for blocks of $\mathcal{P}$. In particular, we write $\mathrm{parent}(B_w) = B_v$ and $\mathrm{parent}(w) = v$. We define the *boundary* S_u of a block B_u as the subset of B_u that has neighbors in $V(G) \setminus B_u$ in the graph G. The *(inner) separator* of a block is the union of the boundaries of its children. Note that leaf blocks have empty separators and the root block has an empty boundary. We call $\mathcal{P}$ *ε-balanced*, for $\varepsilon \geq 0$ if, for every node $u \in V(T)$ and children v and w of u, we have $|B_v| \leq (1 + \varepsilon)|B_w|$. We call $\mathcal{P}$ *balanced* if it is ε-balanced for $\varepsilon \in O(1)$.

6.2.1 Asymptotics of Properties 1–5

In order to talk about asymptotic running times of algorithms, one needs to consider infinite families of inputs. As the definitions of Properties 1–5 in Section 6.1 do not make their asymptotic interpretations explicit, we provide formal definitions of the properties defined in Section 6.1 in this section. Let $\mathcal{G} = \{G_1, G_2, \dots\}$ be an infinite family of graphs.

For the first property, local diametric partners, the asymptotic interpretation is straightforward. The only important detail is that the constants hidden by the big O-notation may not depend on individual graphs.

▷ **Property 6.4 (local diametric partners).** We say that $\mathcal{G}$ has *d_{local}-local diametric partners*, if there exist positive integer constants a, b, c such that for every graph $G \in \mathcal{G}$, every vertex $v \in V(G)$, and every positive integer x, there exists at most a vertices $w_1, \dots, w_a \in V(G)$ such that the union of their closed $b \cdot (x + d_{\text{local}}) + c$ neighborhoods contains every x-diametric partner of v in G. ◁

The asymptotic interpretation of the second property, few corners, is analogous.

▷ **Property 6.5 (few corners).** We say that $\mathcal{G}$ has *d_{corner}-few corners*, if there exist positive integer constants a, b, c such that for every graph $G \in \mathcal{G}$ and every positive integer x there exist up to a vertices $w_1, \dots, w_a \in V(G)$ such that the union of their closed $b \cdot (x + d_{\text{corner}}) + c$ neighborhoods contains every vertex with at least one x-diametric partner in G. ◁

The remaining properties also depend on recursive partitions. For each graph $G_i \in \mathcal{G}$, let $\mathcal{P}_i$ be a recursive partition. Then $\mathcal{I} = \{(G_1, \mathcal{P}_1), (G_2, \mathcal{P}_2), \dots\}$ forms an infinite family of graphs together with recursive partitions. With this the asymptotic interpretation of the third property is again straightforward, again with the only important detail being that the constants hidden in the big O-notation must be universal for the family of instances.

▷ **Property 6.6 (small separators).** We say that $\mathcal{I}$ has *(α, β)-small separators*, if there exist positive integer constants b, c such that for every $(G, \mathcal{P}) \in \mathcal{I}$ and every block B induced by $\mathcal{P}$ on G, the separator of B has size at most $b \cdot \left(n^{\alpha} \cdot |B|^{\beta}\right) + c$. ◁

For the fourth property it is important to only compare the sizes and diameters of blocks of the same graph, as across graphs similarly sized blocks are allowed to have different diameter.

▷ **Property 6.7 (size-dependent diameters).** We say that $\mathcal{I}$ has *size-dependent diameters*, if there exist constants $b, c, d, e > 1$ such that for every $(G, \mathcal{P}) \in \mathcal{I}$ and

every two blocks A, B induced by $\mathcal{P}$ on G, we have $|A| \leq b \cdot |B| + c$ if and only if $\text{diam}_{G[A]} \leq d \cdot \text{diam}_{G[B]} + e$. ◁

For the fifth property, the interpretation is again straightforward.

▷ **Property 6.8 (low fragmentation).** We say that $\mathcal{I}$ has *low fragmentation* if there exist positive integer constants a, b such that for every $(G, \mathcal{P}) \in \mathcal{I}$, every vertex $v \in V(G)$ and every integer x, the closed x neighborhood of v intersects at most a blocks B of $\mathcal{P}$ with diameter $\text{diam}_{G[B]}$ between $\frac{1}{b}x$ and bx. ◁

6.3 Diameter Algorithm

Let $G = (V, E)$ be a graph and let $\mathcal{P}$ be a recursive partition of G. Our algorithm roughly works as follows. We choose a *flat partition*, a set of similarly sized blocks $\mathcal{B} \subset \mathcal{P}$ that together form a partition of V. This can be easily achieved by choosing some parameter $k \geq k_{\text{leaf}}$ and including a block $B \in \mathcal{P}$ in $\mathcal{B}$ if it has size at most k while its parent has size larger than k: $\mathcal{B} = \{B \in \mathcal{P} \mid |B| \leq k \text{ and } |\text{parent}(B)| > k\}$.

As $\mathcal{B}$ is a partition of V, computing $\text{maxdist}_G(A, B)$ for every pair of blocks $A, B \in \mathcal{B}$ (including $A = B$) yields the diameter of the graph. To save time, we ignore a pair of blocks $A, B \in \mathcal{B}$ if $\text{maxdist}_G(A, B)$ is obviously too small to be relevant for the diameter. For this, we use an upper bound $u(A, B)$ on $\text{maxdist}_G(A, B)$ and a global lower bound ℓ on the diameter. If $u(A, B) < \ell$, we can safely skip the pair (A, B). Otherwise, we call (A, B) a *candidate pair* and A a *candidate partner* for B (and vice versa). With this, it remains to solve the following problems.

1. Efficiently compute the maxdist for each candidate pair.
2. Bound the number of candidate pairs and compute them efficiently.
3. Obtain a lower bound ℓ.

Regarding Point 1, we discuss in Section 6.3.1 how to compute $\text{maxdist}(A, B)$ efficiently for a candidate pair (A, B). For this, we introduce a pre-processing step in which we construct a data structure that acts as an exact distance oracle and lets us quickly compute the distance between arbitrary vertices.

For Point 2, the number of candidate pairs depends on the upper bound $u(A, B)$. We define $u(A, B)$ in Section 6.3.2 and show how it can be efficiently evaluated using the distance oracle from Section 6.3.1. Assuming that G has d_{local}-local diametric partners (Property 1) (and optionally also d_{corner}-few corners, Property 2) and that $\mathcal{P}$ is (α, β)-well-spaced (Properties 3–5) we then show that our definition of

$u(A, B)$ leads to a small number of candidate pairs. In fact, there can be much fewer candidate pairs than pairs of blocks. In Section 6.3.4 we provide a way of computing all candidate pairs that does not need to consider all pairs of blocks.

Finally, we address Point 3, by essentially performing a binary search on the solution. For this, we treat ℓ not as a lower bound on the diameter, but simply as a *guess* for the diameter. Then, using the approach outlined above, we can decide whether ℓ is smaller, equal or larger than the diameter. Assuming $\ell \geq \text{diam}_G$, there are only few candidate pairs and the algorithm terminates quickly (see also the discussion of Point 2 above). Equivalently, if the algorithm does not terminate quickly, this means that $\ell < \text{diam}_G$. This allows us to determine diam_G in a binary search. In Section 6.3.5 we discuss the details for this, including an additional exponential search in order to choose k optimally.

6.3.1 Distance Oracle and Maxdist Computation

In the pre-processing step, we conduct breadth-first searches from separator vertices in the blocks of the recursive partition. This allows us to construct an exact distance oracle and compute vertex eccentricities in each block.

The rough idea for the distance oracle is very simple. If a vertex set S separates two vertices v and w, then shortest paths between v and w cross S. Thus there exists a vertex $s \in S$ such that the distance between v and w is $\text{dist}_G(v, s) + \text{dist}_G(s, w)$ and the distance between v and w can be found by checking the distances to v and w from each $s \in S$. We note that this is a standard approach and has already been used for distance oracles and (directed) reachability oracles in other settings [Ari+96; Ber23; FK14].

▷ **Lemma 6.9.** Let G be a graph with degeneracy d and a balanced recursive partition $\mathcal{P}$ with (α, β)-small separators (Property 3). Then, in $O(n^{1+\alpha+\beta} \cdot d)$ time, we can construct a data-structure $\mathcal{D}$ requiring $O(n^{1+\alpha+\beta})$ space that can

1. for any two vertices v, w compute $\text{dist}_G(v, w)$ in $O(n^{\alpha+\beta})$ time, unless v and w are both non-boundary vertices of the same leaf-block of $\mathcal{P}$,
2. for each block $B \in \mathcal{P}$ return a vertex $b \in B$ and $\text{ecc}_{G[B]}(b)$ in $O(1)$ time.

◁

Proof. For each block B of $\mathcal{P}$ we perform a BFS restricted to $G[B]$ from each vertex s in the separator of B. We store the distances from s to each vertex $b \in G[B]$ in an array $D_{B,s}$. For the running time, note that a block B with k vertices has $k \cdot d$ edges and an separator of size $s_k \in O(k^\alpha \cdot n^\beta)$. Thus, the cost for the s_k BFS runs on

$G[B]$ is in $O(s_k \cdot kd)$. We write $T(k)$ for the running time of these BFS runs plus the running time in all descendant blocks of B. Each block B has $c \in O(1)$ children that form a partition of B, so this results in the recurrence

$$T(k) = k^\alpha \cdot n^\beta \cdot k \cdot d + \sum_{i=1}^{c} T(b_i k),$$

where the $b_i \in (0,1)$ are constants summing up to 1, $\sum_{i=1}^{c} b_i = 1$. Then, $T(n) \in O(n^{1+\alpha+\beta} \cdot d)$ follows via induction over n or by applying the theorem of Akra and Bazzi [AB98]. Similarly, for each size k block we need to store $O(k)$ distances, so total space needed is in $O(n^{1+\alpha+\beta})$. Note that these BFS runs allow us to store, for each block B, the eccentricity of a vertex of $G[B]$ without incurring any additional asymptotic overhead. This implies statement (2).

It remains to discuss the distance queries. Let v and w be two vertices and let P be a shortest path between them in G. Consider the smallest block B that contains P. If B is a leaf-block, then both v and w are contained in B. Then, if without loss of generality v is a boundary vertex of B, we have $\mathrm{dist}_{G[B]}(v,w) = \mathrm{dist}_G(v,w)$. Furthermore, v is a separator vertex of the parent B' of B and the distance between v and w can be looked up in the pre-computed distance array $D_{B',v}$. If v and w are both non-boundary vertices of B, it P may not contain any separator vertices, so we ignore this case.

If otherwise B is not a leaf-block, we claim that P contains a separator of B. To see this consider two cases. If v and w lie in different child blocks of B, then a path from v to w clearly needs to cross the separator of B. Otherwise, if v and w lie in the same child block B' of B, then by the choice of B the path P is not contained in B'. This means that P crosses a boundary vertex b of B', which is a separator vertex of B. Then, as P is contained in B we have $\mathrm{dist}_G(v,w) = \mathrm{dist}_{G[B]}(v,b) + \mathrm{dist}_{G[B]}(b,w)$. Again, these distances can be looked up in a pre-computed distance array $D_{B,b}$.

To summarize both cases, there exists a block B that is a common ancestor of the leaf blocks containing v and w and a separator vertex s of B such that $\mathrm{dist}_G(v,w) = \mathrm{dist}_{G[B]}(v,s) + \mathrm{dist}_{G[B]}(s,w)$. Thus, in order to answer distance queries, we can proceed as follows. For a given pair of vertices $v, w \in V(G)$ we first identify the leaf blocks B_v and B_w with $v \in B_v$ and $w \in B_w$. This can be done in $O(1)$ time, assuming that as an additional preprocessing step we iterate over all leaves of $\mathcal{P}$ in $O(n)$ time. Next, we find the lowest common ancestor B^* of B_v and B_w in $\mathcal{P}$ in $O(1)$ time assuming some additional $O(n)$ time preprocessing [BF00]. Let $B_1 = B^*, \ldots, B_\ell$ be the sequence of ancestor blocks from B^* to the root. For each block B_i with separator S_i (for $i \in [\ell]$) we iterate over each separator vertex $s \in S_i$ and return the minimum value of $\mathrm{dist}_{G[B_i]}(s,v) + \mathrm{dist}_{G[B_i]}(s,w)$. By the considerations above, this

correctly gives the distance of v and w in G unless both v and w are non-boundary vertices of the same leaf-block.

To analyze the running time recall that distances $\text{dist}_{G[B_i]}(s,v)$ and $\text{dist}_{G[B_i]}(s,v)$ can be looked up in $D_{B,s}$ in $O(1)$ time. Consequently, we need to bound the number of the separator vertices $S_1 \cup \ldots \cup S_\ell$ of B^* and its ancestors. Each block with k vertices has a separator of size $O(k^\alpha n^\beta)$ and, for a constant $c < 1$, each child block has size at most c times the size of its parent. Using $\ell \in O(\log n)$, we have

$$\sum_{i=1}^{\ell} |S_i| \in \sum_{i=0}^{\log n} O\Big((n \cdot c^i)^\alpha n^\beta\Big) = O(n^\alpha n^\beta) \sum_{i=0}^{\log n} (c^\alpha)^i = O(n^\alpha n^\beta),$$

so the oracle query can be answered in $O(n^\alpha n^\beta)$ time. ■

In addition to accelerating the upper bound evaluation (see Section 6.3.2), this distance oracle allows us to efficiently compute the maximum distance between any two blocks A and B of $\mathcal{P}$. We present two methods for doing so. The first one simply computes $\text{maxdist}(A,B)$ using $|A| \cdot |B|$ oracle calls.

▷ **Lemma 6.10.** Let G be a graph with degeneracy d and let $\mathcal{P}$ be a balanced recursive partition with (α,β)-small separators (Property 3). After a pre-processing step taking $O(n^{1+\alpha+\beta}d)$ time, for any two distinct blocks $A, B \in \mathcal{P}$ we can compute $\text{maxdist}_G(A,B)$ in $O(|A| \cdot |B| \cdot n^{\alpha+\beta})$ time. ◁

Proof. The running time follows directly from Lemma 6.9, by calling the distance oracle for each pair of vertices in $A \times B$. ■

For the second method, we construct small auxiliary graphs on which distance computations are faster than on the whole graph. This improves upon the simple approach in some settings, especially on graphs with small separators.

Let $A, B \subseteq V(G)$ be two sets of vertices. We define the *overlay graph* $H_{(A,B)}$ of A and B as follows. Let S_A and S_B be the boundary of A and B. Then, we construct $H_{(A,B)}$ by taking $G[A \cup B]$ and inserting weighted edges between all vertices $S_A \cup S_B$. For $s, s' \in S_A \cup S_B$ the edge $\{s,s'\} \in E(H_{(A,B)})$ has weight equal to the distance of s and s' in G, i.e., $d_G(s,s')$. The following lemma shows that distances in the overlay graph are equal to distances in G.

▷ **Lemma 6.11.** Let $A, B \subseteq V(G)$ be vertex subsets and let $H = H_{(A,B)}$ be the overlay graph of A and B. Then for any two vertices $u, v \in A \cup B$ their distance in G is equal to their distance in H, i.e., $d_G(u,v) = d_H(u,v)$. ◁

Proof. We first show $d_{H_{(A,B)}}(u,v) \leq dG(u,v)$. For this, let P be a shortest path from u to v in G. If P contains no vertices of $V(G) \setminus V(H_{(A,B)})$, then P is also a path in $H_{(A,B)}$. For the other case, we first note that only boundary vertices of A or B can have neighbors in $V(G) \setminus V(H_{(A,B)})$. Now consider an inclusion-maximal subpath P_s of P that contains only vertices of $V(G) \setminus V(H_{(A,B)})$. Then, in $H_{(A,B)}$ the two vertices that come before and after P_s on P are connected by a weighted edge of length $|P_s|$. Thus, by removing all maximal subpaths containing only vertices of $V(G) \setminus V(H_{(A,B)})$ from P we obtain an equally long path in $H_{(A,B)}$.

Next, we show $d_G(u,v) \leq d_{H_{(A,B)}}(u,v)$. Suppose P is a shortest path between u and v in $H_{(A,B)}$. Then, an equally long path in G can be obtained by replacing any weighted shortcut edge of P with the shortest path between the endpoints of that edge in G. ■

To compute the maximum distance of two blocks A and B of $\mathcal{P}$, we can thus compute it on $H_{(A,B)}$ instead. The running time then consists of the time for the construction of $H_{(A,B)}$ plus the time for running Dijkstra's algorithm $|A|$ times. We summarize this in the following lemma.

▷ **Lemma 6.12.** Let G be a graph with degeneracy d and let $\mathcal{P}$ be a balanced recursive partition with (α, β)-small separators (Property 3). After a pre-processing step taking $O(n^{1+\alpha+\beta}d)$ time, for any blocks $A, B \subseteq \mathcal{P}$ (including $A = B$) with $k = |A \cup B|$ we can compute $\text{maxdist}_G(A, B)$ in time

$$O(k^2 \log k + k^2 d + k^{1+2\alpha} n^{2\beta} + k^{2\alpha} n^{\alpha+3\beta}).$$

◁

Proof. We rely on the pre-processing and distance oracle given in Lemma 6.9 and construct the overlay graph $H_{(A,B)}$. For this, we construct the subgraph $G[A \cup B]$ in $O(kd)$ time and then look up all distances between boundary vertices using the distance oracle. Each distance lookup takes $O(n^{\alpha+\beta})$ time. As the blocks have $O(k^\alpha n^\beta)$ boundary vertices, this results in a total running time in

$$O(kd + (k^\alpha n^\beta)^2 n^{\alpha+\beta}) = O(kd + k^{2\alpha} n^{\alpha+3\beta})$$

for the construction of $H_{(A,B)}$. By Lemma 6.11, we can determine $\text{maxdist}_G(A, B)$ by running Dijkstra's algorithm from every vertex a of A to find the vertex $b \in B$ most distant from a. The overlay graph has $O(k)$ vertices and $O(kd + k^{2\alpha} n^{2\alpha})$ edges, so Dijkstra's algorithm runs in

$$O(k \log k + kd + k^{2\alpha} n^{2\beta})$$

time and $O(k)$ such queries take

$$O(k^2 \log k + k^2 d + k^{1+2\alpha} n^{2\beta})$$

time. Together with the construction of $H_{(B,C)}$, this yields the running time claimed in the lemma statement. ■

6.3.2 Upper Bound

Recall that we want to avoid computing the maxdist of pairs of blocks that are too close to possibly contain diametric vertex pairs. In this section we give the upper bound used for this purpose and show that it can be evaluated efficiently.

▷ **Lemma 6.13.** Let G be a graph with degeneracy d and a balanced recursive partition $\mathcal{P}$ with (α, β)-small separators. After an $O(n^{1+\alpha+\beta} d)$ time pre-processing, we can for any given blocks $A, B \in P$ (including $A = B$) compute a value $u(A, B)$ in time $O(n^{\alpha+\beta})$, such that

$$\operatorname{maxdist}_G(A, B) \leq u(A, B) \leq \operatorname{maxdist}_G(A, B) + 2 \operatorname{diam}_{G[A]} + 2 \operatorname{diam}_{G[B]} .$$

◁

Proof. As a pre-processing, we rely on the distance oracle from Lemma 6.9, which is constructed in $O(n^{1+\alpha+\beta} d)$ time. We consider blocks $A, B \in \mathcal{P}$.

If $A = B$ we compute $u(A, B)$ by selecting an arbitrary vertex $a \in A$ and looking up the eccentricity $\operatorname{ecc}_{G[A]}(a)$ in $O(1)$ time. Setting $u(A, B) = 2 \cdot \operatorname{ecc}_{G[A]}(a)$, we have $\operatorname{maxdist}_G(A, B) = \operatorname{diam}_{G[A]} \leq u(A, B) \leq 2 \operatorname{diam}_{G[A]}$. It remains to consider the case $A \neq B$. We set

$$u(A, B) = 2 \operatorname{ecc}_{G[A]}(a) + 2 \operatorname{ecc}_{G[B]}(b) + \operatorname{dist}_G(a, b),$$

where $a \in A$ and $b \in B$ are chosen such that, using the distance oracle, we can look up their eccentricities in $O(1)$ time and compute $d_G(a, b)$ in $O(n^{\alpha+\beta})$ time (see also Lemma 6.9).

Regarding the claimed inequalities, we first show $\operatorname{maxdist}_G(A, B) \leq u(A, B)$. Consider maximally distant vertices $a^* \in A$ and $b^* \in B$, i.e., $\operatorname{dist}_G(a^*, b^*) = \operatorname{maxdist}_G(A)$, and let P be a shortest path between a^* and b^*. Then, any other path from a^* to b^* is at least as long as P. We thus consider the path P' obtained by concatenating a shortest a^*a path in $G[A]$, a shortest ab path and a shortest bb^* path in $G[B]$. Then

we have

$$\begin{aligned}|P| &= \operatorname{maxdist}_G(A,B) = \operatorname{dist}_G(a^*,b^*)\\ &\leq \operatorname{dist}_{G[A]}(a^*,a) + \operatorname{dist}_G(a,b) + \operatorname{dist}_{G[B]}(b,b^*)\\ &= |P'|.\end{aligned}$$

We have $\operatorname{dist}_G(a^*,a) \leq \operatorname{diam}_{G[A]} \leq 2\operatorname{ecc}_{G[A]}(a)$. With an analogous estimate on $\operatorname{dist}_G(b^*,b)$, we thus obtain

$$\operatorname{maxdist}_G(A,B) \leq 2\operatorname{ecc}_{G[A]}(a) + \operatorname{dist}_G(a',b') + 2\operatorname{ecc}_{G[B]}(b) = u(A,B).$$

For the other claimed inequality,

$$u(A,B) \leq \operatorname{maxdist}_G(A,B) + 2\operatorname{diam}_{G[A]} + 2\operatorname{diam}_{G[B]},$$

note that $\operatorname{dist}_G(a,b) \leq \operatorname{maxdist}_G(A,B)$ and additionally that any eccentricity in a graph is at most the diameter. ■

This means that the $u(A,B)$ overshoots $\operatorname{maxdist}_G(A,B)$ by at most a constant multiple of the diameters of $G[A]$ and $G[B]$. This helps us to analyze the effectiveness of the pruning based on this upper bound in the next section.

6.3.3 Number of Candidate Pairs

Recall from the beginning of Section 6.3, that we consider a flat partition $\mathcal{B} \subset \mathcal{P}$, i.e., a set of similarly sized blocks of $\mathcal{P}$ that partitions V. Moreover, we denote $B \in \mathcal{B}$ as a candidate partner for A under ℓ, if $u(A,B) \geq \ell$. In the following, we give upper bounds for the number of candidate pairs in $\mathcal{B}$ on graphs with local diametric partners and, optionally, few corners. We begin by formalizing the intuition that candidate pairs are located far from each other in the graph.

▷ **Lemma 6.14.** For $\ell \geq \operatorname{diam}_G$, let A and B be two candidate pairs under ℓ, i.e., $u(A,B) \geq \ell$. Then for every pair of vertices $a \in A$ and $b \in B$ we have

$$\operatorname{dist}_G(a,b) \geq \operatorname{diam}_G - 3\operatorname{diam}_{G[A]} - 3\operatorname{diam}_{G[B]}.$$

◁

Proof. Consider A and B with $u(A,B) \geq \ell$. Using $\ell \geq \operatorname{diam}_G$ and the maximum value of $u(A,B)$ from Lemma 6.13, this implies

$$\operatorname{maxdist}(A,B) + 2\operatorname{diam}_{G[A]} + 2\operatorname{diam}_{G[B]} \geq u(A,B) \geq \ell \geq \operatorname{diam}_G.$$

This means there are vertices $a^* \in A$ and $b^* \in B$ with

$$\operatorname{dist}_G(a^*, b^*) \geq \operatorname{diam}_G - 2\operatorname{diam}_{G[A]} - 2\operatorname{diam}_{G[B]} .$$

For a pair of vertices $a \in A$ and $b \in B$, we have

$$\begin{aligned}\operatorname{dist}_G(a^*, b^*) &\leq \operatorname{dist}_G(a^*, a) + \operatorname{dist}_G(a, b) + \operatorname{dist}_G(b, b^*) \\ &\leq \operatorname{diam}_{G[A]} + \operatorname{dist}_G(a, b) + \operatorname{diam}_{G[B]} .\end{aligned}$$

Together, this implies $\operatorname{dist}_G(a, b) \geq \operatorname{diam}_G - 3\operatorname{diam}_{G[A]} - 3\operatorname{diam}_{G[B]}$. ■

Assume that G has d_{local}-local diametric partners (Property 1) and $\mathcal{P}$ is (α, β)-well-spaced. We show that among the similarly sized blocks $\mathcal{B}$ each block has only few candidates. The idea for this is roughly as follows. By Lemma 6.14, vertices of a candidate pair are almost diametrical. However, by Property 1 the almost diametrical partners of any vertex are covered by few balls of bounded radius and by Property 5 only few blocks with relevant diameters intersect any such ball. This gives a bound on the number of candidates.

▷ **Lemma 6.15.** Let G be a graph with d_{local}-local diametric partners and a (α, β)-well-spaced recursive partition $\mathcal{P}$. Let further $\ell \geq \operatorname{diam}_G$, and let $\mathcal{B} \subset \mathcal{P}$ be a flat partition with similarly sized blocks of diameter in $\Omega(d_{\text{local}})$. Then each block $A \in \mathcal{B}$ has only $O(1)$ candidates in $\mathcal{B}$. ◁

Proof. Let $A \in \mathcal{B}$ be a block that has a candidate $B \in \mathcal{B}$ under ℓ, i.e., $u(A, B) \geq \ell$. Then, for any pair of vertices $a \in A$ and $b \in B$ we have

$$\operatorname{dist}_G(a, b) \geq \operatorname{diam}_G - 3\operatorname{diam}_{G[A]} - 3\operatorname{diam}_{G[B]}$$

by Lemma 6.14. All blocks of $\mathcal{B}$ have roughly the same size and by Property 4 also roughly the same diameters, so this means $\operatorname{dist}_G(a, b) \geq \operatorname{diam}_G - x$ for $x \in \Theta(\operatorname{diam}_{G[A]})$. Recall that we call vertices satisfying the above inequality x-diametric. We have thus shown that the vertices of any candidate block B of A are among the x-diametric partners of $a \in A$.

Property 1 guarantees that all x-diametric partners of a lie in $O(1)$ balls of radius $O(x + d_{\text{local}}) = O(x)$. Moreover, due to Property 5, only a constant number of blocks with diameter $\Theta(\operatorname{diam}_{G[A]}) = \Theta(\operatorname{diam}_{G[B]})$ intersect any ball of radius $O(x) = O(\operatorname{diam}_{G[A]})$, i.e., the x-diametric partners of a lie in a constant number of blocks with diameter $\Theta(\operatorname{diam}_{G[A]})$. Thus, A has only a constant number of candidates in $\mathcal{B}$. ■

Now assume that G also has d_{corner}-few corners (Property 2). We show that in a flat partition with blocks of sufficiently large diameter only few blocks have candidates.

▷ **Lemma 6.16.** Let G be a graph with d_{corner}-few corners, let $\mathcal{P}$ be a (α, β)-well-spaced recursive partition, let $\ell \geq \operatorname{diam}_G$, and let $\mathcal{B} \subset \mathcal{P}$ be a flat partition with blocks of diameter in $\Omega(d_{\text{corner}})$. Then, only $O(1)$ blocks of $\mathcal{B}$ have candidates in $\mathcal{B}$. ◁

Proof. Let $A \in \mathcal{B}$ be a block that has a candidate B in $\mathcal{B}$, i.e., $u(A, B) \geq \ell$. By Lemma 6.14, for any vertices $a \in A$ and $b \in B$ we have

$$\operatorname{dist}_G(a, b) \geq \operatorname{diam}_G - 3\operatorname{diam}_{G[A]} - 3\operatorname{diam}_{G[B]}.$$

This means that b is in the $x = (3\operatorname{diam}_{G[A]} + 3\operatorname{diam}_{G[B]})$-diametric set of a. Note that $\operatorname{diam}_{G[A]} \in \Omega(d_{\text{corner}})$ and thus $x \in \Omega(d_{\text{corner}})$. We have thus shown that for some $x \in \Omega(d_{\text{corner}})$ every block A with a candidate in $\mathcal{B}$ has at least one x-diametric partner. By Property 2, the set of vertices that have x-diametric partners can be covered by $O(1)$ balls of radius $O(x + d_{\text{corner}}) = O(x)$ in G. By Property 5, any such ball is intersected by only $O(1)$ blocks with diameter in $\Theta(x) = \Theta(\operatorname{diam}_{G[A]})$. This implies that only $O(1)$ blocks of $\mathcal{B}$ contain vertices with x-diametric partners. ■

Assuming that the diameter of blocks in $\mathcal{B}_i$ is in $\Omega(\min\{d_{\text{local}}, d_{\text{corner}}\})$ this improves the bound on the number of candidate pairs given by Lemma 6.15. We get that only $O(1)$ blocks of $\mathcal{B}_i$ have candidates and each only has $O(1)$ candidates. Thus, the total number of candidate pairs is also in $O(1)$.

6.3.4 Efficient Candidate Enumeration

We have shown that within a flat partition $\mathcal{B}$ with blocks of sufficiently large diameter, every block has only $O(1)$ candidates (see Section 6.3.3). Additionally, for each pair of blocks we can quickly test whether they form a candidate pair (see Lemma 6.13). However, if the blocks of $\mathcal{B}$ consist of $\Theta(k)$ vertices, there are $\Theta(n/k)$ many blocks and thus testing each of the $O(n^2/k^2)$ pairs is pretty expensive, especially for small k.

To improve upon this, the following observation is helpful. Consider two blocks A and B that do not form a candidate pair, i.e., $u(A, B) < \ell$. Then we have $\operatorname{maxdist}_G(A, B) < \ell$ and we can ignore all vertices $a \in A$ and $b \in B$ on the search for the diameter. More generally, let $A' \subset A$ and $B' \subset B$ be descendants of A and B, respectively. If A and B do not form a candidate pair, then $\operatorname{maxdist}_G(A, B) < \ell$

and hence also $\text{maxdist}_G(A', B') < \ell$. In this case we consider (A', B') to not be a candidate pair regardless of the actual value of $u(A', B')$.

We thus go through the recursive partition in a top-down fashion, i.e., instead of directly considering pairs of blocks in $\mathcal{B}$, we first evaluate the upper bounds for their ancestors in $\mathcal{P}$, starting at the root. This way we can already exclude pairs of blocks in $\mathcal{B}$ that have ancestors that do not form candidate pairs.

To make the approach more precise, we maintain an *intermediate flat partition* $\mathcal{B}_i$. Initially, $\mathcal{B}_0$ consists of the root block of $\mathcal{P}$. Afterwards, in step $i \geq 1$, we obtain $\mathcal{B}_i$ from $\mathcal{B}_{i-1}$ by replacing the largest block of $\mathcal{B}_{i-1}$ with its children in $\mathcal{P}$. At each step, we keep track of all candidate pairs in $\mathcal{B}_i$. This means that when replacing a block $B \in \mathcal{B}_{i-1}$ with its children $B'_1, \ldots, B'_b \in \mathcal{B}_i$, we compute the upper bound between each child and each candidate for B in $\mathcal{B}_{i-1}$. We stop this process with a *final* flat partition $\mathcal{B}_j = \mathcal{B}$. In Section 6.3.5 we discuss how j is chosen in order to minimize the running time. Before that, we summarize important properties of the intermediate flat partitions.

Clearly, the balance and bounded branching factor of $\mathcal{P}$ implies that after each step i, the blocks $\mathcal{B}_i$ are similarly sized, i.e., for any $A, B \in \mathcal{B}_i$ we have $|A| \in \Theta(|B|)$. With Property 4 (size-dependent diameters), this implies $\text{diam}_{G[A]} \in \Theta(\text{diam}_{G[B]})$. As $\mathcal{B}_i$ forms a partition of V, it consists of $\Theta(n/|A|)$ blocks for $A \in \mathcal{B}_i$. By Lemma 6.15, each block $A \in \mathcal{B}_i$ has only $O(1)$ candidates in $\mathcal{B}_i$ under lower bound $\ell \geq \text{diam}_G$, provided that $\text{diam}_{G[A]} \in \Omega(d_{\text{local}})$ (see Property 1). This allows us to bound the running time needed to compute the candidate pairs in B_j.

▷ **Lemma 6.17.** Assume that the flat partition $\mathcal{B}_j$ has blocks with diameter in $\Omega(d_{\text{local}})$ and $\ell \geq \text{diam}_G$. Then, in $O(n^{1+\alpha+\beta})$ time, we can compute $\mathcal{B}_0, \ldots, \mathcal{B}_j$ and enumerate all candidate pairs in $\mathcal{B}_i$ for all $i \in [0, j]$. ◁

Proof. At each step $i \leq j$ a block B from an intermediate flat partition $\mathcal{B}_{i-1}$ is replaced with its children and upper bounds are evaluated in order to maintain the set of candidate pairs. However by Property 4, for $i < j$ the diameters of blocks in B_i are asymptotically at least as large as the diameters of blocks in B_j and thus by Lemma 6.15 the replaced block B only has $O(1)$ candidate blocks in B_{i-1}. As each block of a well-spaced recursive partition only has $O(1)$ children, this means that the splitting step leads to $O(1)$ upper bound evaluations, which take $O(n^{\alpha+\beta})$ time (see Lemma 6.13).

It thus remains to bound the number of blocks across all considered flat partitions $\mathcal{B}_0, \ldots, \mathcal{B}_j$. Assume the blocks in $\mathcal{B}_j$ have size $\Theta(k)$. Then it consists of $O(\frac{n}{k})$ blocks. All other blocks that are part of an intermediate flat partition $\mathcal{B}_i$ with $i < j$ are ancestors of a block of $\mathcal{B}_j$. Further, every block in $\mathcal{P}$ has at least 2 children. Thus, the total number of unique blocks in any of the considered partitions is also in

$O(\frac{n}{k})$. This means that the total running time for all upper bound evaluations is in $O(\frac{n}{k} \cdot n^{\alpha+\beta})$. In particular this is dominated by the running time $O(n^{1+\alpha+\beta})$ for the construction the distance oracle (see Lemmas 6.9 and 6.13). ■

6.3.5 Putting Everything Together

In this section we combine the components laid out above and complete the algorithm. To recap the different steps, the fundamental approach expects a graph G, a recursive partition $\mathcal{P}$ and a bound ℓ and proceeds as follows. First, construct the distance oracle, then enumerate all candidate pairs in intermediate flat partitions $\mathcal{B}_i$ and, for the final flat partition $\mathcal{B}_j$, compute the maxdist of each candidate pair. This then yields the diameter of G or, if $\mathcal{B}_j$ contains no candidate pair, that $\text{diam}_G < \ell$.

To analyze the algorithm, we assume that G has n vertices, degeneracy d, satisfies Property 1, and that $\mathcal{P}$ is (α, β)-well-spaced. We first consider the setting where $\mathcal{B}_j$ is chosen such that the blocks have some specified size $\Theta(k)$. By Lemma 6.9 the time needed to construct the distance oracle is in

$$O(n^{1+\alpha+\beta} \cdot d). \tag{6.1}$$

Assuming the blocks in $\mathcal{B}_j$ have diameter in $\Omega(d_{\text{local}})$ and $\ell \geq \text{diam}_G$, enumerating all candidates in $\mathcal{B}_j$ causes no asymptotic overhead (Lemma 6.17). Next, by Lemma 6.12 the running time for one maxdist computation is in

$$\tilde{O}(k^2 d + k^{1+2\alpha} n^{2\beta} + k^{2\alpha} n^{\alpha+3\beta}). \tag{6.2}$$

Alternatively, by Lemma 6.10 the maxdist of a pair of distinct blocks can also be computed in time

$$\tilde{O}(k^2 n^{\alpha+\beta}). \tag{6.3}$$

To guarantee that all candidate pairs consist of distinct blocks, it suffices to assume $k \in o(n)$. Then, the diameters of the blocks in $\mathcal{B}_j$ are in $o(\text{diam}_G)$ and thus the upper bound of any block with itself is in $o(\text{diam}_G)$.

As $\mathcal{B}_j$ forms a partition of the vertices, it consists of $O(n/k)$ blocks. Recall that we assume that G has d_{local}-local diametric partners and that the blocks in $\mathcal{B}_j$ have diameter in $\Omega(d_{\text{local}})$. Consequently, by Lemma 6.15, each block has $O(1)$ candidate partners under $\ell \geq \text{diam}_G$. Thus, there are $O(n/k)$ candidate pairs. If G has d_{corner}-few corners and the blocks of $\mathcal{B}_j$ have diameters in $\Omega(d_{\text{corner}})$, then only $O(1)$ blocks have candidate partners, by Lemma 6.16. This means that there are only $O(1)$ candidate pairs. The total running time is thus given by the preprocessing (Equation (6.1)) and the maxdist computations for each candidate

pair (Equation (6.2), respectively Equation (6.3)). We summarize the algorithm as follows.

▷ **Lemma 6.18 (Size-based algorithm).** For G and $\mathcal{P}$ as above, $\ell \in [n]$, and $k \in o(n)$, there is an algorithm $\mathcal{A}(G, \mathcal{P}, \ell, k)$ that decides how diam_G compares to ℓ. If $\ell \geq \mathrm{diam}_G$ and $\mathcal{P}$ admits a flat partition into blocks of size $\Theta(k)$ and diameter $\Omega(d_{\text{local}})$ the running time of $\mathcal{A}(G, \mathcal{P}, \ell, k)$ is in

$$\tilde{O}\Big(n^{1+\alpha+\beta} \cdot d + \min\big\{nkd + k^{2\alpha} n^{1+2\beta} + k^{2\alpha-1} n^{1+\alpha+3\beta},\ kn^{1+\alpha+\beta}\big\}\Big).$$

If additionally G has d_{corner}-few corners and the diameter of the blocks is in $\Omega(d_{\text{corner}})$ the running time is in

$$\tilde{O}\Big(n^{1+\alpha+\beta} \cdot d + \min\big\{k^2 d + k^{1+2\alpha} n^{2\beta} + k^{2\alpha} n^{\alpha+3\beta},\ k^2 n^{\alpha+\beta}\big\}\Big).$$

◁

Assume that for $\ell \geq n$ the above algorithm terminates $T(n, k)$ time steps. Then, by executing the algorithm for $T(n, k)$ steps, one can use a binary search to determine diam_G. In fact, this strategy can be extended to also find an optimal value for the parameter k, such that the final algorithm only depends on G and $\mathcal{P}$. With the following lemma we describe this strategy in a generic way.

▷ **Lemma 6.19.** Let $\mathcal{A}(I, \ell, k)$ be an algorithm that takes as input some instance I along with two integer parameters $\ell, k \in [n]$ and that decides how ℓ compares to a numerical quantity $D(I)$ with $D(I) \in [n]$. Assume that

1. for $\ell \geq D(I)$, the algorithm runs in $T(n, k)$ time, and further that
2. there is a value $k^* \in [n]$ such that every $k \in \Theta(k^*)$ minimizes $T(n, k)$ for every n up to constant factors.

Then, there is an algorithm $A'(I)$ that taking an instance I that computes $D(I)$ in time $O(T(n, k^*) \cdot \log^2 n)$. ◁

Proof. The core idea is that if k^* and $T(n, k^*)$ are known, then a binary search on ℓ can be used to compute $D(I)$ using $O(\log n)$ executions of $\mathcal{A}(I, \ell, k^*)$. To see this, note that by assumption $D(s)$ takes a value between 1 and n and if $\mathcal{A}(I, \ell, k^*)$ terminates, it decides whether $\ell < D(I)$, $\ell = D(I)$, or $\ell > D(I)$. Otherwise, if $\mathcal{A}(I, \ell, k^*)$ does not terminate within $T(n, k^*)$ time, then this implies $\ell < D(I)$ and $\mathcal{A}$ can be halted.

It remains to find k^* and $T(n, k^*)$. For this, we use an exponential search, i.e., we find (up to a constant factor) the smallest time limit T and (up to a constant factor) the smallest value for k, such that the binary search outlined above succeeds in $O(T)$ time. To be more precise, we start with a constant time limit $T \in O(1)$ and iteratively increase it by a constant factor until a subroutine succeeds. In that subroutine, we start with $k = 1$ and iteratively increase k by a constant factor until either $k > n$ or a second subroutine succeeds. The second subroutine tries to determine $D(I)$ using a binary search on ℓ, by executing $\mathcal{A}(I, \ell, k)$ with a time limit T. If $k \in O(k^*)$ and $T \geq T(n, k^*)$, the binary search finds $D(I)$ in $O(T \log n)$ steps as discussed above. If otherwise k or T are too small, then the binary search may wrongly conclude that a probed value of ℓ is smaller than $D(I)$. In this case the binary search fails and larger values for k are tested until either $D(I)$ is found or $k > n$. In the latter case, T is increased and the exponential search on k starts again. This means that for each value of T, the subroutine tests $O(\log n)$ values for k in $O(T \log n)$ time each. At some point, T reaches $T(n, k^*)$. Then, the exponential search on k succeeds and the binary search identifies $D(I)$. As T is increased by a constant factor each time, the total running time is dominated by the last round and thus in $O(T(n, k^*) \log^2 n)$. ■

We apply this to the size-based algorithm from Lemma 6.18 and obtain the following.

▷ **Theorem 6.3.** Let G be a n-vertex graph with degeneracy d satisfying Property 1. Let $\mathcal{P}$ be a (α, β)-well-spaced recursive partition with leaf-block size k_{leaf}. For every $k \geq k_{\text{leaf}}$ such that $k \in o(n)$ and blocks of size $\Theta(k)$ have diameter $\Omega(d_{\text{local}})$, our algorithm computes the diameter of G in time

$$\tilde{O}\Big(n^{1+\alpha+\beta} \cdot d + \min\big\{nkd + k^{2\alpha} n^{1+2\beta} + k^{2\alpha-1} n^{1+\alpha+3\beta},\ kn^{1+\alpha+\beta}\big\}\Big).$$

If also Property 2 holds and blocks of size $\Theta(k)$ have diameter $\Omega(d_{\text{corner}})$, it runs in time

$$\tilde{O}\Big(n^{1+\alpha+\beta} \cdot d + \min\big\{k^2 d + k^{1+2\alpha} n^{2\beta} + k^{2\alpha} n^{\alpha+3\beta},\ k^2 n^{\alpha+\beta}\big\}\Big).$$

◁

We note that in our case, the second logarithmic factor of Lemma 6.19 can be avoided. To see how, note that the running time for the maxdist computations only depends on a few quantities known to the algorithm, such as the number of candidate pairs, the size of the blocks, and the size of their boundaries. This means that for each intermediate flat partition $\mathcal{B}_i$ the algorithm can make an (up

to constant factors) tight estimate for the time needed to compute the maxdist of all candidate pairs in $\mathcal{B}_i$. Thus, the algorithm can keep track of the elapsed time until each step i and calculate the cost for computing the maxdist over all candidate pairs of $\mathcal{B}_i$. If this estimate for the total running time is below the given time limit t the algorithm computes the maxdists on $\mathcal{B}_i = \mathcal{B}_j$. Otherwise it continues by considering the next flat partition $\mathcal{B}_{i+1}$. This continues until either the time limit is up or until a flat partition is found for which the time limit is sufficient. If there is some size k and a flat partition $\mathcal{B}_i$ with blocks of size roughly k such that the size-based algorithm $\mathcal{A}(G, \mathcal{P}, \ell, k)$ from Lemma 6.18 runs in time t^*, then for any given time limit $t \geq t^*$ the above algorithm also finds a flat partition for which it can calculate the maxdist in time t. This way only one logarithmic factor is added by the binary search.

6.4 Analysis on Random Geometric Graphs

In order to apply the algorithm from Theorem 6.3 on random geometric graphs, we show that Properties 1–5 hold asymptotically almost surely. We start with an overview of the general proof ideas and also give the main intuitions for our analysis of the iFUB algorithm.

Properties 1 and 2. Recall from the introduction that Properties 1 and 2 are based on simple geometric observations (Observations 1 and 2) that intuitively hold for the ground spaces of torus/square RGGs on a purely geometric level. Consequently, the main task is to transfer these geometric intuitions to the graph setting, i.e., it remains to show that the derived properties hold a.a.s. on the graphs. As our main tool we extend known results on the graph–geometry stretch, i.e., the relation between geometric distance and graph distance [Día+16]. Roughly speaking, we use that for vertices with known geometric distance d on a RGG with connection radius r, the graph distance likely lies within a narrow range around d/r. This allows us to show that the almost diametric partners of a vertex v have geometric distance close to the geometric diameter. They are thus contained in a small geometric ball and therefore also in a small ball in the graph.

Properties 3, 4, and 5. Showing that the properties related to recursive partitions likely hold on RGGs works similarly. Recall that these properties are derived from Observations 3–5 which describe intuitive geometric properties. We thus formally define a recursive partition based on the quadtree-like subdivision used for these observations, see also Figure 6.2 (d). Then, geometrically each cell with

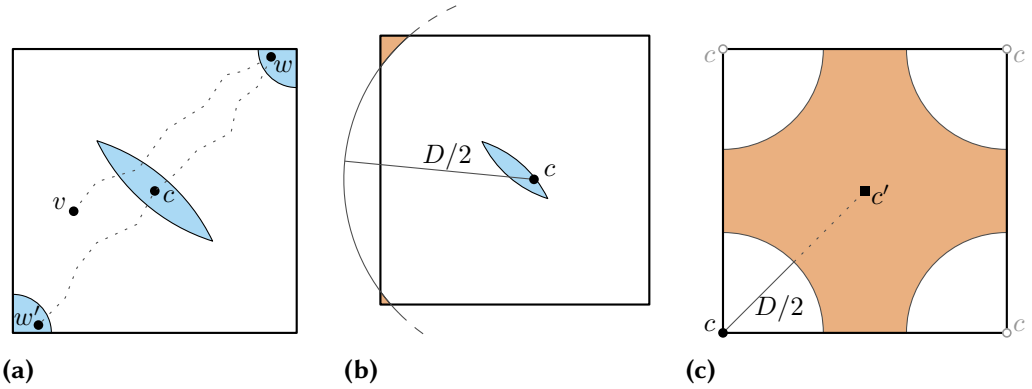
side length s has perimeter $4s$, area s^2, and diameter $\sqrt{2}s$. As the number of vertices within any polynomially sized region is highly concentrated, this gives us that the recursive partition of the graph is balanced and has small separators. With the bound on the graph–geometry stretch this also gives the claimed size-dependent diameters. Here, we need slightly stronger guarantees than already shown [Día+16], because we need short paths that do not only exist in the whole graph, but also in the subgraphs induced by the recursive partition. Finally, it is easy to formally prove a variant of Observation 5, which directly implies Property 5 via the concentration of the vertices.

Running Time of iFUB. For our analysis of the iFUB algorithm on random geometric graphs, we consider the variant of iFUB that chooses a central vertex c using the *2-sweep* heuristic as follows. First, the algorithm performs a BFS from an arbitrary vertex v and picks a vertex w in the last layer, i.e., with maximum distance from v. Then, a second BFS is performed from w and the vertex c is chosen half the way on a shortest path between w and a vertex w' with maximum distance from w. Subsequently, iFUB performs exactly one BFS from every vertex whose distance to c is more than half the diameter of G. In the settings we consider, these vertices also account for pretty much all BFS runs and thus directly determine the total running time.

For the upper bound on square RGGs, we begin by showing that the vertex w selected by the first BFS is likely located close to a corner of the square ground space. Afterwards, we show that c is located in a small lens close to the geometric center of the ground space, see also Figure 6.3 (a). Both of these steps use basic geometric arguments and rely on the graph–geometry stretch, however bounding the size of the lens in the second step is somewhat technical. Conditional on c being located close to the geometric center, it is then easy to show that there are not many vertices whose distance to c is at least half the diameter of G. This then gives the running time in the first part of Theorem 6.1.

For the torus, any chosen center c results in more than half of all points having distance more than half the diameter of G from c, see also Figure 6.3 (c). Combined with the graph–geometry stretch this directly shows that iFUB performs a BFS from $\Omega(n)$ vertices, covering the second part of Theorem 6.1.

Outline. In the remainder of this section we start with our definition of random geometric graphs and afterwards show the stretch bounds. In order to apply the algorithmic framework from Section 6.3, we then first define the recursive partition and show that Properties 3–5 are likely to hold, before considering Properties 1



(a) (b) (c)

Figure 6.3: Visualization for the analysis of the iFUB algorithm. Part (a) concerns the 2-sweep heuristic on square RGGs: for any chosen vertex v, the highly distant vertex w lies in a small region (blue) around a corner of the ground space and the central vertex c lies in a small lens (also blue, not to scale) in the center. Only few vertices lie in the small regions with distance at least half the diameter D from c (Part (b), orange), giving an upper bound on the running time on square RGGs. Part (c) shows the torus, a point c, and its antipodal partner c'; it can be seen that most points (orange) have distance at least half the diameter D from c.

and 2, and finally combining these results and apply Theorem 6.3 to get running times for the algorithm. Afterwards, in Section 6.4.6 we analyze the running time of iFUB.

6.4.1 Definitions

For a side length $s \in \mathbb{R}$, we define $\mathcal{S}_s$ as the square $[0,s)^2 \subseteq \mathbb{R}^2$. We write $d_{\mathbb{E}}(u,v)$ for the Euclidean distance between two points $u,v \in \mathbb{R}^2$. Further, we define the *(flat) torus* $\mathcal{T}_s = \mathbb{R}^2/(s\cdot\mathbb{Z})^2$ with side length s as the equivalence classes of points in $[0,s)^2$, i.e., we write v as a shorthand for $[v] = \{v + m \mid m \in (s\cdot\mathbb{Z})^2\}$ for any $v \in \mathbb{R}^2$. The *toroidal distance* between two equivalence classes is defined as the minimum Euclidean distance between points in the equivalence classes, i.e., $d_{\mathcal{T}_s}(u,v) := \min\{d_{\mathbb{E}}(z,w) \mid z \in [u], w \in [v]\}$. In the context of random geometric graphs we write $\mathcal{S}$ (respectively $\mathcal{T}$) for $\mathcal{S}_{\sqrt{n}}$ (respectively $\mathcal{T}_{\sqrt{n}}$).

Then for a ground space $X \in \{\mathcal{T},\mathcal{S}\}$ we define the *random geometric graph* $G \in \mathcal{G}(X,n,r)$ as follows. Throughout this chapter we assume $r < n^{\rho}$ for a constant $\rho < \frac{1}{2}$. The vertex set $V(G)$ is obtained by drawing n points independently and uniformly in X. We identify each vertex v with its geometric position $(v_x,v_y) \in \mathbb{R}^2$. Then, two vertices are adjacent exactly if their distance in X is at most r, i.e., $E(G) =$

$\{\{v,w\} \in \binom{V(G)}{2} \mid d_X(v,w) \leq r\}$. Here, we let d_X refer to the Euclidean distance for $\mathcal{G}(\mathcal{S},n,r)$ and to the toroidal distance for $\mathcal{G}(\mathcal{T},n,r)$. We call $G \in \mathcal{G}(\mathcal{S},n,r)$ a *square* random geometric graph and $G \in \mathcal{G}(\mathcal{T},n,r)$ a *torus* random geometric graph.

We write $\tilde{\mathcal{G}}(X,n,r)$ for the related model of *Poisson RGGs*. Here, we first draw a Poisson random variable N with mean n and then draw $G \sim \tilde{\mathcal{G}}(X,n,r)$ as a random geometric graph with N vertices. Note that this is equivalent to setting $V(G)$ as the result of a Poisson point process with intensity 1 on X. The advantage of $\tilde{\mathcal{G}}(X,n,r)$ over $\mathcal{G}(X,n,r)$ is that the number of vertices in any region $A \subseteq X$ with area measure a follows a Poisson random variable with mean a and is independent from the number of vertices in a disjoint region $A' \subseteq X \setminus A$.

6.4.2 Graph-Geometry Stretch on RGGs

There is a tight relationship between the geometric distance and the graph distance of vertices in a random geometric graph. For a lower bound on the graph distance, note that in a random geometric graph with connection radius r, each edge connects vertices of distance at most r. Thus, regardless of the underlying geometry $X \in \{\mathcal{S},\mathcal{T}\}$ for any pair of vertices u,v we have

$$\operatorname{dist}_G(u,v) \geq \left\lceil \frac{d_X(u,v)}{r} \right\rceil. \tag{6.4}$$

Interestingly, on random geometric graphs we also get upper bounds for the graph distance conditional on the geometric distance of vertices. Below, we slightly adapt results by Díaz, Mitsche, Perarnau, and Pérez-Giménez [Día+16] to also give upper bounds for the graph distance within subgraphs induced by axis aligned squares.

▷ **Lemma 6.20 ([Día+16], Theorem 1.1).** Let $G \sim \mathcal{G}(\mathcal{S},n,r)$ be a square RGG with connection radius $r \in \omega(\log^{3/4} n)$. Asymptotically almost surely, for every pair of vertices $u,v \in V(G)$ with $d_{\mathbb{E}}(u,v) > r$, we have

$$\operatorname{dist}_G(u,v) \leq \left\lceil \frac{d_{\mathbb{E}}(u,v)}{r}(1+s) \right\rceil$$

with

$$s = s(d_{\mathbb{E}}(u,v),r) \in \begin{cases} O\left(r^{-4/3}\log n\right) & \text{if } d_{\mathbb{E}}(u,v) \leq r\ln n, \\ O\left(r^{-4/3}\right) & \text{if } d_{\mathbb{E}}(u,v) > r\ln n. \end{cases}$$

Furthermore, every axis-aligned square containing u and v contains a (u,v)-path of such length. For $G \in \tilde{\mathcal{G}}(\mathcal{S}, n, r)$, the same event holds with probability $1 - o(n^{-5/2})$. ◁

Proof. The upper bound on the graph distance between u and v is given in Statement (ii) of Theorem 1.1 in [Día+16] as $\text{dist}_G(u,v) \leq \left\lceil \frac{d_{\mathbb{E}}(u,v)}{r}(1+s) \right\rceil$ with

$$s = s(d_{\mathbb{E}}(u,v), r) \in O\left(\left(\frac{\log n}{r^2 + r \cdot d_{\mathbb{E}}(u,v)}\right)^{2/3} + \left(\frac{\sqrt{\log n}}{r}\right)^4 + r^{-4/3}\right).$$

For $r > \log^{3/4} n$ we have $\left(\frac{\sqrt{\log n}}{r}\right)^4 < r^{-4/3}$. Thus, the second summand is dominated by the third one. For the first summand we have

$$\left(\frac{\log n}{r^2 + r \cdot d_{\mathbb{E}}(u,v)}\right)^{2/3} < \left(\frac{\log n}{r^2}\right)^{2/3} = r^{-4/3} \log^{2/3} n.$$

Additionally, for $d_{\mathbb{E}}(u,v) \geq r \ln n$ the first summand is smaller than $r^{-4/3}$.

It remains to show that one (u,v) path of such length is contained in a square bounding box of u and v. For this we need to consider some details made in the proof of Corollary 2.1 [Día+16]. This corollary considers a disk intersection graph of a Poisson point process in the plane, where vertex u is planted at the origin and vertex v at $(t,0)$. The authors then consider the rectangle $R = [1.01\alpha, t - 1.01\alpha] \times [0, \alpha]$ for $\alpha \in \Theta(r)$ with $\alpha < 0.004r$. Relying on further lemmas that we do not need to discuss here, the authors show that with probability $1 - o(n^{-5/2})$ there exists a path of the desired length from u to v that only uses vertices in R. The theorem then follows with a de-Poissonization, reducing the probability to $1 - o(n^{-2})$, followed by a union bound over all pairs of vertices in the random geometric graph G, reducing the probability to $1 - o(1)$.

In order for the union bound to work, the authors show that the path within the rectangle R implies a path within the $[0, \sqrt{n}] \times [0, \sqrt{n}]$ ground space of G, even if u and v lie on the boundary of the ground space. The union bound afterwards does not distinguish between vertices close to the boundary and the many more vertices far from the boundary. Thus, the proof given in [Día+16] also implies that for any pair of vertices u and v in G there is a path of the desired length that does not leave any axis aligned square containing u and v. ■

We use a coupling argument to show that Lemma 6.20 holds analogously on the torus.

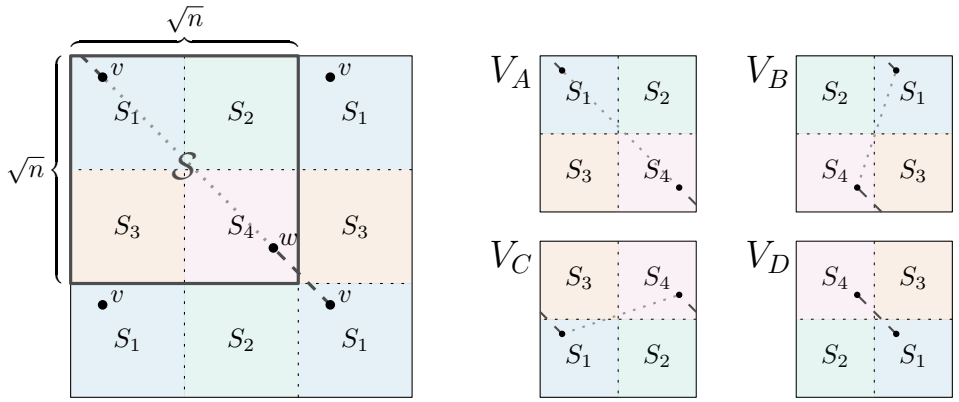


Figure 6.4: Sketch of the coupling argument. The square $\mathcal{S}$ consists of four smaller squares and vertex sets V_A to V_D are formed by rearranging the points of a Poisson point process on $\mathcal{S}$ as depicted. This way the torus distance between points v and w in $\mathcal{S}$ is realized as the Euclidean distance in one of the rearrangements.

▷ **Lemma 6.21.** Let $G \sim \mathcal{G}(\mathcal{T}, n, r)$ be a torus-RGG with connection radius $r \in \omega(\log^{3/4} n)$. Asymptotically almost surely, for every pair of vertices $v, w \in V(G)$ with $d_{\mathcal{T}}(v,w) > r$, we have $\mathrm{dist}_G(v,w) \leq \left\lceil \frac{d_{\mathcal{T}}(v,w)}{r}(1 + s(v,w,r)) \right\rceil$ with the error term $s(d_{\mathcal{T}}(v,w), r)$ as in Lemma 6.20, and further, every minimal[14] axis-aligned square containing v and w contains a (v,w)-path of such length. ◁

Proof. Consider the torus-RGG G. Then, for any pair of vertices (v,w) the torus $\mathcal{T}$ can be mapped into the square $\mathcal{S}$ such that the distance between v and w is preserved, i.e. such that the toroidal distance between v and w is equal to their Euclidean distance in the mapping. In fact, four different mappings are sufficient for all pairs of vertices, see also Figure 6.4.

More formally, we introduce a coupling between $\tilde{\mathcal{G}}(\mathcal{T}, n, s)$ and $\tilde{\mathcal{G}}(\mathcal{S}, n, s)$ as follows. The rough idea is to define four coupled RGGs by suitably re-shuffling the points sampled from a Poisson point process, such that the Torus distance between vertices is realized by the minimum Euclidean distance in one of the four coupled RGGs. By Lemma 6.20, in each of the four square RGGs graph distances are a.a.s. not much longer than implied by the geometry, so the same holds on the torus RGGs.

By restricting the Poisson point process Π on $\mathbb{R}^2$ to $\mathcal{S} = [0, \sqrt{n})^2$ we obtain the vertex set V_A of a Poisson random geometric graph G_A. We define vertex sets V_B, V_C,

14 Requiring *minimal* squares is the main difference to the statement of Lemma 6.20. This difference is necessary, because on the torus there are squares containing u and v that do not contain the geodesic between u and v.

and V_D, by translating the points sampled by $\Pi \cap \mathcal{S}$ as indicated in Figure 6.4. Let S_1, S_2, S_3, and S_4 be the four half-open squares of side length $s = \frac{\sqrt{n}}{2}$ that tile $\mathcal{S}$. For $i \in [1,4]$ denote $\Pi \cap S_i$ as Π_i and for a point $p \in \mathbb{R}^2$ write $\tau_p : \mathbb{R}^2 \to \mathbb{R}^2,\ q \mapsto q+p$ for the translation that maps the origin to p. We define

$$\begin{aligned} V_B &= \tau_{(s,0)}(\Pi_1 \cup \Pi_3) \cup \tau_{(-s,0)}(\Pi_2 \cup \Pi_4) \\ V_C &= \tau_{(0,s)}(\Pi_3 \cup \Pi_4) \cup \tau_{(0,-s)}(\Pi_1 \cup \Pi_2) \\ V_D &= \tau_{-s,s}(\Pi_4) \cup \tau_{(s,s)}(\Pi_3) \cup \tau_{(-s,-s)}(\Pi_2) \cup \tau_{(s,-s)}(\Pi_1); \end{aligned}$$

see also Figure 6.4. Note that these vertex sets follow the distribution of the Poisson point process Π restricted to $\mathcal{S}$. Combining these vertex sets with the threshold radius r we obtain geometric graphs G_B, G_C, G_D that are each uniform Poisson random geometric graphs sampled from $\tilde{\mathcal{G}}(\mathcal{S}, n, s)$.

Additionally, we define $G_T \sim \tilde{\mathcal{G}}(\mathcal{S}, n, s)$ by connecting the vertices of V_A according to the torus metric. For simplicity, we refer to vertices of the different graphs as $v_i \in V_i$, such that, for instance each $v_A \in V_A$ has a copy $v_B \in V_B$ that is shifted along the x-axis by $\frac{\sqrt{n}}{2}$ either to the left or to the right. Then for any pair of vertices $v_A, w_A \in V_A$ we have

$$d_{\mathcal{T}}(v_A, w_A) = \min_{i \in \{A,B,C,D\}} d_{\mathbb{E}}(v_i - w_i), \text{ and } \operatorname{dist}_{G_T}(v_A, w_A) = \min_{i \in \{A,B,C,D\}} \operatorname{dist}_{G_i}(v_i, w_i),$$

i.e., the torus distance of v_A and v_B is the minimum Euclidean distance of any of their copies in V_A, V_B, V_C, and V_D. Asymptotically almost surely, the stretch event $\mathcal{E}_{\text{stretch}}$ of Lemma 6.20 holds on all four graphs. Thus a.a.s. for any two vertices v_T, w_T of G_T we have

$$\begin{aligned} \operatorname{dist}_{G_T}(v_T, w_T) &= \min_{i \in \{A,B,C,D\}} \operatorname{dist}_{G_i}(v_i, w_i) \\ &\leq \min_{i \in \{A,B,C,D\}} \left\lceil \frac{\operatorname{dist}_{\mathbb{E}}(v_i, w_i)}{r} (1 + s(v_i, w_i, r)) \right\rceil, \\ &= \left\lceil \frac{d_{\mathcal{T}}(v_T, w_T)}{r} (1 + s(v_T, w_T, r)) \right\rceil, \end{aligned}$$

where $s(v_i, w_i, r)$ is the error term from Lemma 6.20. We also get that at least one such path is contained in the smallest axis-aligned square containing u and v. ■

We refer to events from Lemma 6.20, respectively Lemma 6.21, as the *stretch event* and denote it with $\mathcal{E}_{\text{stretch}}$. Note in particular that conditioning on $\mathcal{E}_{\text{stretch}}$ gives an upper bound on the graph distance of all pairs of vertices.

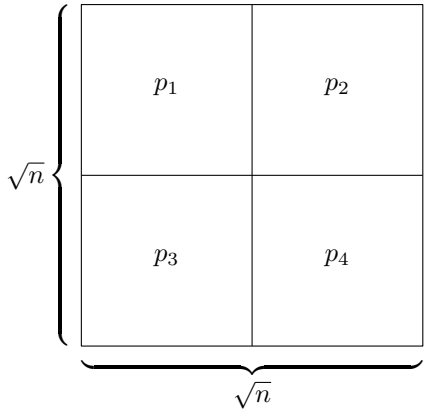


Figure 6.5: Four child parts p_1, p_2, p_3, and p_4 of the root part p_ε of $\mathcal{P}$.

6.4.3 Recursive partition

We now give a formal definition for our recursive partition and show that with high probability it is balanced and Properties 3–5 hold. We first define the infinite *quadtree partition* $\mathcal{P}$ as a recursive subdivision of the of the square $\mathcal{S} = [0, \sqrt{n}) \times [0, \sqrt{n})$. We index parts using words $\sigma \in \{1, 2, 3, 4\}^*$. The root of $\mathcal{P}$ is $p_\varepsilon = \mathcal{S}$, where ε stands for the empty word. For any $p_\sigma = [x_1, x_2) \times [y_1, y_2)$, let $x' = \frac{x_1+x_2}{2}$ and $y' = \frac{y_1+y_2}{2}$. Then, the children of p_σ (numbered from 1 to 4) are

$$p_{\sigma 1} = [x_1, x') \times [y', y_2), \qquad p_{\sigma 2} = [x', x_2) \times [y', y_2),$$
$$p_{\sigma 3} = [x_1, x') \times [y_1, y'), \qquad p_{\sigma 4} = [x', x_2) \times [y_1, y').$$

Figure 6.5 shows p_ε and its children. Clearly, $\mathcal{P}$ has constant branching factor. We define the level ℓ of a part as the length of its index, e.g., p_ε is on level 0 and p_{13} on level 2. Note that $\mathcal{P}$ contains 4^ℓ level ℓ parts and each level ℓ part $p \in \mathcal{P}$ has side length $\frac{\sqrt{n}}{2^\ell}$ and area $\frac{n}{4^\ell}$. For our algorithm we only need a finite subset of $\mathcal{P}$. For this, we define $\mathcal{P}^\ell$ as $\mathcal{P}$ restricted to parts of level at most ℓ. We call $\mathcal{P}^\ell$ the *ℓ-layered* quadtree partition.

For a geometric graph G with $V(G) \subseteq [0, \sqrt{n}) \times [0, \sqrt{n})$ the partition $\mathcal{P}^\ell$ induces a recursive partition $G[\mathcal{P}^\ell]$ of the vertices, via the intersection of $V(G)$ with the parts of $\mathcal{P}^\ell$. In the following, we show that for a (square or torus) random geometric graph G the recursive partition $G[\mathcal{P}^\ell]$ a.a.s. is balanced, has small separators, size-dependent diameters, and bounded fragmentation.

Balance and Separator Sizes

With respect to the area measures of its parts, the infinite quadtree partition $\mathcal{P}$ already is balanced and has small separators, so it remains to show the same for the induced recursive partition $G[\mathcal{P}^\ell]$. We use concentration bounds for the number of vertices in regions of sufficient area to show that the induced recursive partition of G is also balanced and has small separators.

▷ **Lemma 6.22.** Let $G \sim \mathcal{G}(X, n, r)$ be a random geometric graph with ground space $X \in \{\mathcal{S}, \mathcal{T}\}$ and let $R \subseteq X$ be a measurable subset of X with area $A(R) \in \omega(\log n)$. Then, for any constant c the probability that the number of vertices of G that lie in R is between $A(R) \cdot \left(1 - \sqrt{\frac{3c \ln n}{A(R)}}\right)$ and $A(R) \cdot \left(1 + \sqrt{\frac{3c \ln n}{A(R)}}\right)$ is at least $1 - O(n^{-c})$. ◁

Proof. Let $n_R = |V(G) \cap R|$ be the number of vertices in R. Using $V(G) = v_1, \ldots, v_n$, we define X_i as a Bernoulli random variable that indicates whether the ith vertex of G lies in R. Then we have $n_R = \sum_{i=0}^{n} X_i$ and $\mathrm{E}[n_R] = A(R)$. Applying Chernoff bounds (e.g., see Theorem 4.4 and 4.5 in [MU17]), for $0 < \delta < 1$ we have

$$\Pr[n_R > \mathrm{E}[n_R](1+\delta)] \leq e^{-\frac{\delta^2}{3}\mathrm{E}[n_R]}$$

and

$$\Pr[n_R > \mathrm{E}[n_R](1-\delta)] \leq e^{-\frac{\delta^2}{2}\mathrm{E}[n_R]}.$$

We set $\delta = \sqrt{\frac{3c \ln n}{\mathrm{E}[n_R]}}$. Then, we have $\delta < 1$ as by assumption $\mathrm{E}[n_p] = A(r) \in \omega(\log n)$ and thus $\delta \in o(1)$. The above probabilities simplify to

$$\Pr[n_R > \mathrm{E}[n_R](1+\delta)] \leq e^{-c \ln n} \in O(n^{-c})$$

and

$$\Pr[n_R > \mathrm{E}[n_R](1-\delta)] \leq e^{-\frac{3}{2}c \ln n} \leq e^{-c \ln n} \subseteq O(n^{-c}).$$ ■

We apply this to derive bounds for the size of an individual block induced by $\mathcal{P}^\ell$.

▷ **Lemma 6.23.** Let $G \sim \mathcal{G}(X, n, r)$ be a random geometric graph with ground space $X \in \{\mathcal{S}, \mathcal{T}\}$ and connection radius r. For constant $\alpha \in [0, 1]$ let $\ell = \alpha \log_4 n$. For every part P of the ℓ-layered quadtree partition $\mathcal{P}^\ell$ with side length s at least $2r$ it holds w.h.p. that the subgraph $G[V(G) \cap P]$ induced by P contains $s^2(1 \pm o(1))$ vertices and at most $4(s - r)r(1 + o(1))$ separator vertices. ◁

Proof. Let P be a part of $\mathcal{P}^\ell$ with side length $s \geq 2r$. Let $G' = G[V(G) \cap P]$ be the subgraph of G induced by P and let S be the set of its separator vertices, i.e., the subset of $V(G')$ with neighbors in $G \setminus G'$. The separator vertices are contained in a strip of width r around the boundary of the square defined by P. With a side length s the area of P is s^2 and the separator vertices are contained in a region of area $4r(s-r)$. We have $s \geq \frac{\sqrt{n}}{2^\ell} = n^{\frac{1}{2}-\frac{\alpha}{2}}$, thus both areas are in $\omega(\log n)$. Thus, by Lemma 6.22 for any constant c we have $|V(G')| = s^2(1 \pm o(\frac{c \ln n}{s^2}))$ and $|S| \leq 4r(s-r)(1 + o(\frac{c \ln n}{4r(s-r)}))$ with probability $1 - O(n^{-c})$.

The recursive partition $\mathcal{P}^\ell$ contains $O(4^\ell) = O(n^\alpha)$ parts. Thus we can apply the union bound for the considered event over all $P \in \mathcal{P}^\ell$. We obtain that the probability of the respective vertex sets being within the desired interval is at least $1 - O(n^\alpha n^{-c})$. For $c \geq 1 + \alpha$ the desired event occurs with high probability. ■

To show that $\mathcal{P}^\ell$ induces a balanced recursive partition with has small separators, we apply the above lemma to every part.

▷ **Lemma 6.24.** Let $G \sim \mathcal{G}(X, n, r)$ be a random geometric graph with ground space $X \in \{\mathcal{S}, \mathcal{T}\}$ and connection radius r. Further, let $\alpha \in (0, 1)$ be a constant such that $r \in o(n^{1/2-\alpha/2})$, and let $\ell = \alpha \log_4(n)$. Then, with high probability, the partition $G[\mathcal{P}^\ell]$ of G induced by $\mathcal{P}^\ell$ is balanced, has $(1/2, \rho)$-small separators, and the leaf blocks have $\Theta(n^{1-\alpha})$ vertices. ◁

Proof. We condition on the event of Lemma 6.23 that holds with high probability and show the claims one by one.

Balance. Let G_σ be a subgraph induced by a level $j = |\sigma|$ part $P_\sigma \in \mathcal{P}^\ell$, with $j < \ell$. Let $G_{\sigma'}$ and $G_{\sigma''}$ be children of G_σ induced by $\mathcal{P}^\ell$. By Lemma 6.23 G_σ has $\frac{n}{4^j}(1 \pm o(1))$ vertices and $G_{\sigma'}$ and $G_{\sigma''}$ have $\frac{n}{4^{j+1}}(1 \pm o(1))$ vertices. This means that the relative size difference of $G_{\sigma'}$ and $G_{\sigma''}$ tends to 1 and thus the entire induced recursive partition in ε-balanced for arbitrarily small constant $\varepsilon > 0$.

Small separators. Let G' be a subgraph induced by a level i part $P \in \mathcal{P}^\ell$ with side length $s = n^{1/2}2^{-i}$ and let S be the separator of G'. Then by Lemma 6.23, $|V(G')| = s^2(1 \pm o(1))$ and $|S| \leq 4(s-r)r(1 + o(1))$. We have $i \leq \ell = \alpha \log_4 n$ and thus $s \geq n^{1/2}2^{-\alpha \log_4 n} = n^{1/2-\alpha/2}$. By assumption $r \in o(n^{1/2-\alpha/2})$, so $|S| \in O(r\sqrt{|V(G')|})$.

Leaf size. Let G' be a subgraph induced by a leaf part $P_\sigma \in \mathcal{P}^\ell$. Then by Lemma 6.23 $|V(G')| \leq \frac{n}{4^{\alpha \log_4 n}}(1 + o(1)) = n^{1-\alpha}(1 + o(1))$. ■

Diameters in the Recursive Partition

Next, we want to show that the recursive partition a.a.s. has size-dependent diameters, i.e., that similarly sized blocks have similar diameters and these are smaller for smaller blocks. Clearly this holds for the geometric diameters of squares, so it remains to apply the stretch bounds from Section 6.4.2. Importantly, we that for every block bounds on the diameter hold with sufficiently high probability, such that they a.a.s. hold for all blocks of the recursive partition.

Conditional on the stretch event, for every pair of vertices the graph distance is not much larger than necessary based on the geometric distance. With an upper bound for the geometric diameter of each block, this directly translates to an upper bound for the graph diameter. To also get a lower bound, we need to show that each block also contains vertices with almost diametric geometric distance, i.e., there are vertices close to two opposite corners of the block. To this end, we introduce the following lemma, which shows that any region with sufficiently large area is likely to contain at least one vertex.

▷ **Lemma 6.25.** Let $G \in \mathcal{G}(X, n, r)$ be a random geometric graph with n vertices and connection radius r on the ground space $X \in \{\mathcal{S}, \mathcal{T}\}$. Let $R \subseteq \mathcal{S}$ be a region with area a. We have $\Pr[|V(G) \cap R| > 0] \geq 1 - e^{-a}$. ◁

Proof. Let X be a random variable for the number of vertices in R. We have

$$\Pr[X > 0] = 1 - \Pr[X = 0] = 1 - \Pr\left[\bigcap_{v \in V} v \notin R\right].$$

These events are independent and for each $v \in V$ we have $\Pr[v \notin R] = 1 - \frac{a}{n}$. Thus we get

$$\Pr[X > 0] = 1 - \left(1 - \frac{a}{n}\right)^n.$$

For any x we have $1 + x \leq e^x$ and thus $1 - \frac{a}{n} \leq e^{-a/n}$ and $(1 - \frac{a}{n})^n \leq e^{-a}$, which concludes the proof. ■

Note that in particular a region with area $c \cdot \ln n$ is non-empty with probability at least $1 - n^{-c}$. We use this to show bounds on the diameter of individual blocks, first considering square-RGGs.

▷ **Lemma 6.26.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a random geometric graph with connection radius $r \in \omega(\log^{3/4} n)$. Further, let $S \subseteq \mathcal{S}$ be an axis-aligned square of side length

$s > r \ln n$. Then, conditional on the stretch event, we have

$$\operatorname{diam}_{G[V(G)\cap S]} \leq \left\lceil \frac{\sqrt{2}s}{r}(1 + \Theta(r^{-4/3})) \right\rceil.$$

Further, for any constant $C > 0$ we have with probability at least $1 - n^{-C}$

$$\operatorname{diam}_{G[V(G)\cap S]} \geq \frac{\sqrt{2}s}{r} - 1.$$

◁

Proof. The upper bound directly follows via Lemma 6.20. Note that this is one of the places where we need our slightly strengthened version of the lemma, as we need a path using only vertices in S.

For the lower bound let $x = \sqrt{C \cdot \ln n + \ln 2}$. We consider two squares C_1, C_2 of side length x located at two opposite corners of S. The area of these squares is $x^2 = C \cdot \ln n + \ln 2$. Thus, by Lemma 6.25 the probability that both C_1 and C_2 are non-empty is at least $1 - 2e^{-C\cdot\ln n - \ln 2} = 1 - n^{-C}$. In this case, let $v_1 \in V(G) \cap C_1$ and $v_2 \in V(G) \cap C_2$ be such vertices. Then the distance between these vertices is at least $d_{\mathbb{E}}(v_1, v_2) \geq \sqrt{2}s - 2x$. With Equation (6.4) this means that

$$\operatorname{diam}_{G[V(G)\cap S]} \geq d_{G[V(G)\cap S]}(v_1, v_2) \geq \frac{\sqrt{2}s}{r} - \frac{2x}{r}.$$

By assumption $r \in \omega(\sqrt{\log n})$. Hence, $\frac{2x}{r} \in o(1)$, which concludes the proof. ■

We obtain analogous bounds on the diameter of (square regions of) torus RGGs.

▷ **Lemma 6.27.** Let $G \sim \mathcal{G}(\mathcal{T}, n, r)$ be a torus random geometric graph with $r \in \omega(\log^{3/4} n)$. Further, let $S \subseteq \mathcal{T}$ be an axis aligned square of side length s such that $r \ln n < s < \sqrt{n}$. Then, conditional on the stretch event, we have

$$\operatorname{diam}_G \leq \left\lceil \frac{\sqrt{n}}{\sqrt{2}r}(1 + \Theta(r^{-4/3})) \right\rceil$$

and

$$\operatorname{diam}_{G[V(G)\cap S]} \leq \left\lceil \frac{\sqrt{2}s}{r}(1 + \Theta(r^{-4/3})) \right\rceil.$$

Further, for any constant $C > 0$ we have with probability at least $1 - n^{-C}$

$$\operatorname{diam}_G \geq \frac{\sqrt{n}}{\sqrt{2}r} - 1.$$

and

$$\operatorname{diam}_{G[V(G)\cap S]} \geq \frac{\sqrt{2}s}{r} - 1.$$

◁

Proof. The geometric diameter of $\mathcal{T}$ is $\frac{\sqrt{2n}}{2} = \frac{\sqrt{n}}{\sqrt{2}}$ and with the stretch bounds on torus RGGs from Lemma 6.21, the bounds for diam_G follow analogously to Lemma 6.26. The subgraph $G' = G[V(G) \cap S]$ is equal in distribution an analogous subgraph of a square random geometric graph, as with a side length $s < \sqrt{n}$ we avoid paths or geodesics that wrap around the torus $\mathcal{T}$. Thus, the claimed bounds follow directly from Lemma 6.26. ■

It remains to apply a union bound to show that the events from Lemma 6.26, respectively Lemma 6.27, likely hold for each block. In the following lemma, we summarize the result for both the square and torus setting. Note that the geometric diameter of a side length s part P of a quadtree partition is $\sqrt{2}s$ unless $s = \sqrt{n}$ and the setting is on the torus. In that case the geometric diameter is $\frac{2s}{\sqrt{2}}$.

▷ **Lemma 6.28.** Let $G \sim \mathcal{G}(X, n, r)$ be a random geometric graph with ground space $X \in \{\mathcal{S}, \mathcal{T}\}$ and connection radius $r \in \omega(\log^{3/4} n)$. Further, let $\alpha \in (0, 1)$ be a constant such that $r \in o(n^{1/2-\alpha/2})$, and let $\ell = \alpha \log_4(n)$. Then, asymptotically almost surely, for each subgraph G' induced by a part P of the recursive partition $G[\mathcal{P}^\ell]$ we have

$$\frac{d}{r} - 1 \leq \operatorname{diam}_{G'} \leq \frac{d}{r} \cdot (1 + s),$$

where d is the geometric diameter of P in X and $s \in O(r^{-4/3})$. In particular, we get that $\operatorname{diam}_{G'} \in \Theta(|V(G')|^{1/2} r^{-1})$, $G[\mathcal{P}^\ell]$ has size-dependent diameters, and leaf-blocks of $G[\mathcal{P}^\ell]$ have diameter in $\Theta(n^{1/2-\alpha/2} r^{-1})$. ◁

Proof. The stretch event holds asymptotically almost surely on square and torus random geometric graphs, by Lemma 6.20 and Lemma 6.21. Thus, by Lemma 6.26, respectively Lemma 6.27, the diameter of the subgraph induced by a part P of $\mathcal{P}^\ell$ has diameter as claimed with probability $1 - n^{-C}$ for any constant $C > 0$. As there are only $4^\ell = n^\alpha$ such parts, a union bound gives that all diameters fall within the

claimed range with high probability. To conclude, recall that w.h.p. every block with side length x induced by a square has $x^2(1+o(1))$ vertices (Lemma 6.23). Thus, the diameter of a block with k vertices is in $\Theta(k^{1/2}r^{-1})$. Further with $r \in o(n^{1/2-\alpha/2})$ leaf blocks w.h.p. have $O(n^{1-\alpha})$ vertices, by Lemma 6.24. Thus their diameter is in $\Theta(n^{1/2-\alpha/2}r^{-1})$. ■

Fragmentation

We show that the recursive partition $G[\mathcal{P}^\ell]$ has bounded fragmentation. Again, we start with the purely geometric setting and prove that a disk does not intersect too many squares of a grid.

▷ **Lemma 6.29.** Consider an axis aligned grid tiling of $\mathbb{R}^2$ with squares of side length $s > 0$ and a disk D of radius $r > 0$. Then the number of squares intersected by D is at most $O\left(\frac{r^2}{s^2}+1\right)$. ◁

Proof. Without loss of generality assume that D is centered at the origin. If a square $Q = [x_1, x_1+s] \times [y_1, y_1+1]$ intersects D, then it must be contained in the square $Q^\star = [-(r+s), r+s] \times [-(r+s), r+s]$. The area of $Q^\star$ is $(2r+2s)^2$, while each square only has area s^2. This means that $\frac{(2r+2s)^2}{s^2} \in O\left(\frac{r^2}{s^2}+1\right)$ non-intersecting squares of the tiling can lie inside $Q^\star$. ■

We translate this to the setting of the recursive partition of a random geometric graph and obtain the following theorem about its fragmentation.

▷ **Lemma 6.30.** Let $G \sim \mathcal{G}(X, n, r)$ be a random geometric graph with ground space $X \in \{\mathcal{S}, \mathcal{T}\}$ and connection radius $r \in \omega(\log^{3/4} n)$. Further, for a constant $\alpha \in (0,1)$ such that $r \in o(n^{1/2-\alpha/2})$, let $\ell = \alpha \log_4(n)$. Then, asymptotically almost surely, the recursive partition $G[\mathcal{P}^\ell]$ has bounded-fragmentation. ◁

Proof. We consider a set of vertices A that is contained in a ball of radius k in G. Further, let $c_1 > 1$ be a constant and denote by $\mathcal{B}_k$ the blocks of $\mathcal{P}^\ell$ with diameter between k/c_1 and kc_1. We need to show that only a bounded number of blocks of $\mathcal{B}_k$ intersect A.

We condition on the stretch event, which holds asymptotically almost surely (Lemmas 6.20 and 6.21). Then, A is contained in a geometric ball B of radius $O(kr)$. By Lemma 6.29, B intersects only $O\left(\frac{(kr)^2}{s^2}\right)$ squares of side length s in any grid tiling of $\mathbb{R}^2$. This upper bound also applies to the number of blocks intersecting B in each level of $\mathcal{P}^\ell$, as these can be extended into a tiling of $\mathbb{R}^2$. The blocks of $\mathcal{B}_k$ come from $O(\log c_1)$ many different levels in $\mathcal{P}^\ell$. As they have diameter in $\Theta(k)$, the geometric

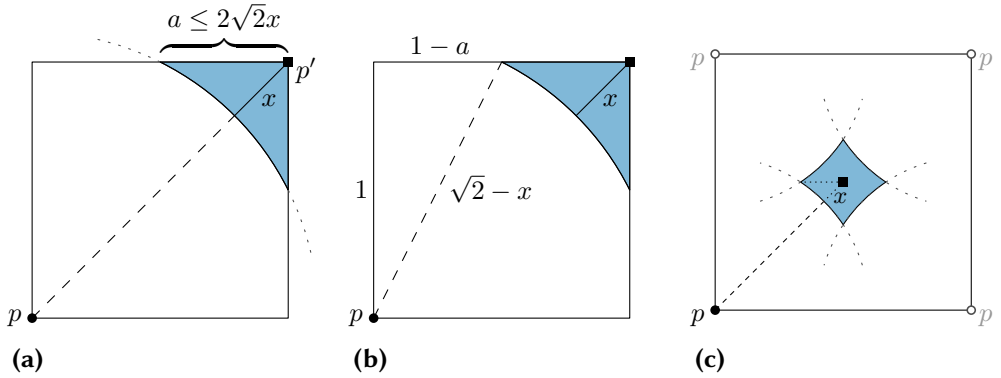


Figure 6.6: Part (a): visualization of point P in corner of the unit square S, with the set of points $Q \subseteq S$ with distance at least $\sqrt{2}-x$ from P shown in blue. Part (b): right triangle used in the proof of Lemma 6.31. Part (c): analogous situation on the torus, see also Lemma 6.32.

diameter and thus also the side length of these blocks is a.a.s. in $\Theta(kr)$ (Lemma 6.28). Thus, the number of blocks of $\mathcal{B}_k$ intersecting B is in $O\left(\frac{(kr)^2}{(kr)^2}\log c_1\right) = O(1)$, which concludes the proof. ■

6.4.4 Local Diametric Partners and Few Corners

In this section we show that Property 1 holds a.a.s. on both square and torus RGGs while Property 2 additionally holds on square RGGs. We begin by considering the purely geometric setting and give a proof for Observation 1 on the unit square.

▷ **Lemma 6.31.** Let S be the unit square and p a point on S. For every $x > 0$, the set of points with distance at least $\sqrt{2}-x$ from p can be covered by a disk of radius $O(x)$. ◁

Proof. Denote the set of points with distance at least $\sqrt{2}-x$ from p by Q and let $r = \sqrt{2}-x$. Without loss of generality, we assume that p is in the left and lower quadrant of S. Note that moving p towards the bottom left corner only increases Q inclusion-wise, so we can even assume that p is located in the corner. Then, Q forms a region around the opposite corner p' of p, see Figure 6.6 (a). To show that Q is contained in a disk of radius $O(x)$, we make a case distinction on x. We first consider large x. For $x \geq \sqrt{2}-1$, any point on the square S has distance at most $\sqrt{2} \leq x \cdot \frac{\sqrt{2}}{\sqrt{2}-1} \in O(x)$ from p'.

Otherwise, we have $x < \sqrt{2}-1$. Then, the boundary of Q consists of two line segments and a circular arc, see also Figure 6.6 (a). Denote the length of these line

segments by a. Every point $q \in Q$ is at distance at most a from the corner p' of S opposite to p, so it remains to find an upper bound on a. Applying the Pythagorean theorem (see Figure 6.6 (b)), we obtain

$$(1-a)^2 + 1^2 = (\sqrt{2}-x)^2$$

and thus

$$a^2 - 2a = x^2 - 2\sqrt{2}x.$$

This quadratic equation has two solutions,

$$a = 1 - \sqrt{x^2 - 2\sqrt{2}x + 1} \quad \text{or} \quad a = 1 + \sqrt{x^2 - 2\sqrt{2}x + 1}.$$

We have $a \leq 1$ as S is a unit square, so only the first solution is relevant. We derive an upper bound as follows. Let $t = x^2 - 2\sqrt{2}x + 1$. Then

$$a = 1 - \sqrt{t} = \frac{(1-\sqrt{t})(1+\sqrt{t})}{1+\sqrt{t}} = \frac{1-t}{1+\sqrt{t}} = \frac{x(2\sqrt{2}-x)}{1+\sqrt{t}} \leq 2\sqrt{2}x.$$

To summarize, either $x \geq \sqrt{2}-1$ and any point on the square S has distance at most $O(x)$ from p', or $x < \sqrt{2}-1$ and by the derivation above, the distance between Q and p' is at most $2\sqrt{2}x \in O(x)$. In both cases, Q is contained in a disk of radius $O(x)$. ■

The same argument works analogously on the torus, yielding the following.

▷ **Lemma 6.32.** Let T be the unit torus and $P \in T$ a point. For $x > 0$, the set of points with distance at least $\frac{\sqrt{2}}{2} - x$ from P is contained in a disk of radius $O(x)$ ◁

Proof. On the torus, the set of points with distance $\frac{\sqrt{2}}{2}$ is shaped like four mirrored and scaled down copies of the analogous set on a square, see also Figure 6.6 (c). The statement thus follows from Lemma 6.31. ■

We can scale the distances considered in Lemma 6.31 by a factor of $\sqrt{n}$ and obtain statements about the geometric ground space of square and torus RGGs. We get that for any vertex v and $x > 0$, the set of points with distance from v at least the geometric diameter minus x is contained in a geometric disk of radius $O(x)$. By the results of Section 6.4.2 the graph distance of vertices with geometric distance d is between $\lceil \frac{d}{r} \rceil$ and $\lceil \frac{d}{r} \rceil (1 + O(r^{-4/3}))$. Together, this allows us to show the following.

▷ **Lemma 6.33.** Asymptotically almost surely, a square or torus random geometric graph $G \sim \mathcal{G}(X, n, r)$ with $X \in \{\mathcal{S}, \mathcal{T}\}$ and connection radius $r \in \omega(\log^{3/4} n)$ has d_{local}-local diametric partners with $d_{\text{local}} \in O(n^{1/2}r^{-7/3} + 1)$. ◁

Proof. Let $v \in V(G)$ be a vertex and for $x \in \mathbb{N}$ let w be a x-diametric partner of v, i.e., $\text{dist}_G(v, w) \geq \text{diam}_G - x$. If $x \in \Omega(\text{diam}_G)$, all vertices $V(G)$ have distance $O(x)$ from v and are thus contained in a ball of radius $O(x)$. Otherwise, $d_G(v, w) \in \Omega(diam_G)$. By Lemmas 6.26 and 6.27, the diameter of G is at least $\text{diam}_G \geq \frac{D}{r} - 1$, where $D \in \Theta(\sqrt{n})$ is the geometric diameter of the ground space of G. This means that a.a.s. $\text{dist}_G(v, w) \in \Omega(n^{1/2}r^{-1})$ and in particular the geometric distance d between v and w is at least $r \ln n$. Thus by Lemmas 6.20 and 6.21 we have $d_G(v, w) \leq \frac{d}{r}(1 + s)$ for $s \in O(r^{-4/3})$. We thus get

$$\begin{aligned} \frac{d}{r}(1+s) &\geq \frac{D}{r} - 1 - x \\ d(1+s) &\geq D - r - xr \\ d &\geq (D - r - xr)\frac{1}{1+s}. \end{aligned}$$

With $s_G \geq 0$ we have $\frac{1}{1+s_G} \geq 1 - s_G$. Hence,

$$\begin{aligned} d \geq (D - r - xr)(1 - s) &\geq D - r - xr - Ds + sr + sxr \\ &\geq D - r - xr - Ds. \end{aligned}$$

This means that the vertex w has distance from v at least the geometric diameter of $\mathcal{S}$, respectively $\mathcal{T}$, minus $(r + xr + Ds)$. By Lemma 6.31, respectively Lemma 6.32, points at this distance from v are contained in a geometric disk of radius $O(r + xr + Ds) = O(r + xr + n^{1/2}r^{-4/3})$. By Lemma 6.25, G w.h.p. contains a vertex v_c within distance $O(\sqrt{\log n})$ of the center of this disk. Any vertex within the disk then has graph distance at most $O(1 + x + n^{1/2}r^{-4/3})$ from v_c. This means that all x-diametric partners of v are contained in a G-ball of radius $O(x + n^{1/2}r^{-4/3} + 1)$. ■

We also show that square RGGs have few corners (Property 2).

▷ **Lemma 6.34.** There exists $d_{\text{corner}} \in \Theta(n^{1/2}r^{-7/3} + 1)$ such that a square random geometric graph $G \sim \mathcal{G}(\mathcal{S}, n, r)$ with connection radius $r \in \omega(\log^{3/4} n)$ has d_{corner}-few corners, asymptotically almost surely. ◁

Proof. We consider four small regions around the corners of $\mathcal{S}$ that we call *corner squares*. We show that for $x \geq 0$ all vertices outside these regions have no x-diametric partners. Then, by contraposition any vertex with at least one x-diametric

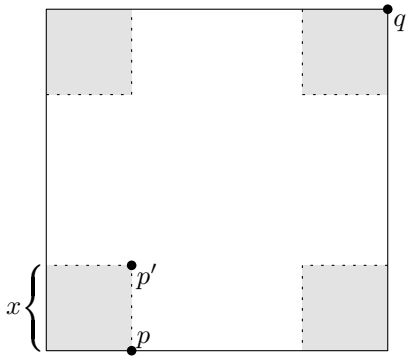


Figure 6.7: Visualization of square $\mathcal{S}$ with the four corner squares in gray; the points p and q maximize the distance between a point in the lower left quadrant and any other point and an upper bound for their distance is easily found by considering the detour via p'.

partner lies in a corner square. Like before, we show this by first making a purely geometric argument and then applying the stretch bounds. Afterwards, it remains to show that each corner square can be covered by a ball of radius $O(x + d_{\text{corner}})$.

For the geometric argument, let $\ell > 0$ be a length parameter to be determined later. We define the corner squares with side length ℓ as follows, see also Figure 6.7. Let $c_1, \dots, c_4$ be the four corners of $\mathcal{S}$. Then, for $i \in [4]$ we define the corner square C_i as the subset of $\mathcal{S}$ that lies within the axis aligned square of side length 2ℓ and center c_i, i.e.,

$$C_i = \{p \in \mathcal{S} \mid \exists d_x, d_y \text{ with } -\ell < d_x, d_y < \ell \text{ such that } p = c_i + (d_x, d_y)\}.$$

Without loss of generality let C be the corner square in the bottom left corner. We consider a point $p \in \mathcal{S}\setminus$ and give an upper bound for the maximum distance from p to any other point $q \in \mathcal{S}$. We can pessimistically assume that p lies at $(0, \ell)$ and q lies at the top right corner of $\mathcal{S}$, see also Figure 6.7. Choosing $p' = (\ell, \ell)$ as the top right corner of C, we have

$$d_{\mathbb{E}}(p, q) < d_{\mathbb{E}}(p, p') + d_{\mathbb{E}}(p', q) = \ell + \sqrt{2n} - \sqrt{2}\ell < \sqrt{2n} - 0.4\ell,$$

which concludes the geometric argument.

We condition on the stretch event of Lemma 6.20, which holds asymptotically almost surely. This means that for any pair of vertices with geometric distance at least $d \geq r \ln n$, the graph distance is at most $\lceil d/r \cdot (1 + s) \rceil$ for some $s \in O(r^{-4/3})$. Thus for any vertex $v \in V(G) \cap \mathcal{S} \setminus (C_1 \cup C_2 \cup C_3 \cup C_4)$ located outside the corner

squares and any other vertex $w \in V(G)$ we have

$$\begin{aligned} d_G(v,w) &\leq \left\lceil \frac{d_{\mathbb{E}}(v,w)}{r}(1+s) \right\rceil \\ &< \frac{d_{\mathbb{E}}(p,q)}{r}(1+s) + 1 \\ &< \frac{\sqrt{2n} - 0.4\ell}{r}(1+s) + 1 \\ &< \frac{\sqrt{2n} + \sqrt{2n}s - 0.4\ell}{r} + 1. \end{aligned}$$

By Lemma 6.26, we have w.h.p. $\operatorname{diam}_G \geq \frac{\sqrt{2n}}{r} - 1$. Setting $\ell = \frac{\sqrt{2n}s + xr + 2r}{0.4}$, we thus have

$$d_G(v,w) < \frac{\sqrt{2n}}{r} - x - 1 \leq \operatorname{diam}_G - x.$$

In other words, no vertex of G has distance $\operatorname{diam}_G - x$ or more from v, hence v has no x-diametric partner. Note that we have $\ell \in O(xr + n^{1/2}s + r) = O(xr + n^{1/2-4/3r} + r)$.

This means that for some $d_{\text{corner}} \in \Theta(n^{1/2-7/3r} + 1) \subseteq \Omega(\sqrt{n}sr^{-1})$ and any $x \geq 0$ we can choose $\ell \in O(xr + d_{\text{corner}}r)$ such that any vertex outside the corner squares of side length ℓ does not have x-diametric partners. This means that any vertex with at least one x-diametric partner lies inside one of four squares of side length ℓ. It remains to show that each of these squares can be covered by $O(1)$ G-balls of radius $O(x + d_{\text{corner}})$. To this end, consider without loss of generality the corner square C in the bottom left corner. By Lemma 6.25, G w.h.p. contains a vertex v_C within geometric distance $O(\sqrt{\log n})$ of the geometric center of C. Any other vertex $w_C \in V(G) \cap C$ then has geometric distance $O(\ell) = O(xr + d_{\text{corner}}r)$ and thus also graph distance at most $O(\ell/r) = O(x + d_{\text{corner}})$ from v_C. Thus, $V(G) \cap C$ is contained in the closed $O(x + d_{\text{corner}})$ neighborhood of some vertex of G.

To conclude, we have shown that there is a $d_{\text{corner}} \in \Theta(n^{1/2}r^{-7/3} + 1)$ and such that for any $x \geq 0$ the set of vertices with at least one x-diametric partner can be covered by a constant number of G-balls of radius $O(x + d_{\text{corner}})$. ■

6.4.5 Computing the Diameter

We are now ready to apply the diameter algorithm from Section 6.3. For a better overview we first give a summary of the properties we have shown above. Let G be a random geometric graph with connection radius $r = n^{\rho}$ for constant $\rho \in (0,1)$. Then, asymptotically almost surely G has d_{local}-local diametric partners for $d_{\text{local}} \in$

$O(n^{\frac{1}{2}-\frac{7}{3}\rho}+1)$ (see Lemma 6.33). Additionally, if G has a square ground space, it a.a.s. has d_{corner}-few corners with $d_{\text{corner}} \in \Theta(n^{\frac{1}{2}-\frac{7}{3}\rho}+1)$ (see Lemma 6.34).

Moreover, for a constant $x \in (0,1)$ with $x > 2\rho$, let $\ell = (1-x)\log_4 n$. Then, the recursive partition $G[\mathcal{P}^\ell]$ a.a.s. is balanced (Lemma 6.24) and has $(\frac{1}{2},\rho)$-small separators (see Lemma 6.24), size-dependent diameters (see Lemma 6.28), and bounded fragmentation (see Lemma 6.30). Moreover, the leaf blocks of $G[\mathcal{P}^\ell]$ have $\Theta(n^x)$ vertices and each block of size k has diameter $\Theta(k^{\frac{1}{2}}n^{-\rho})$ (see Lemma 6.28) and thus leaf blocks have diameter $\Theta(n^\varepsilon)$ for some constant $\varepsilon > 0$.

To apply Theorem 6.3, we additionally need an upper bound on the degeneracy of G. This is easily obtained using the concentration bounds on the number of vertices inside a sufficiently large region.

▷ **Lemma 6.35.** A random geometric graph $G \sim \mathcal{G}(X,n,r)$ with ground space $X \in \{\mathcal{S},\mathcal{T}\}$ and connection radius $r = n^x$ for constant $x \in (0,1)$ has an expected average degree in $O(n^{2x})$ and a maximum degree in $O(n^{2x})$ with high probability. ◁

Proof. Let $v \in V(G)$ be a vertex. Then the degree of v is equal to the number of vertices falling into a region of radius r and hence area $A_x \in O(n^{2x})$ around v, i.e., the expected average degree is in $O(n^{2x})$. With Lemma 6.22, this also means that the degree of v is at most $A_x \cdot \left(1+\sqrt{\frac{3c\ln n}{A_x}}\right)$ with probability at least $1-O(n^{-c})$ for any constant c. For a sufficiently high constant c, a union bound over all vertices shows that all vertices have degree in $O(n^{2x})$ with high probability. ■

Applying Theorem 6.3, we thus get the following running times, depending on the exponent of the connection radius ρ.

▷ **Lemma 6.36.** Let G be a torus or a square random geometric graph with connection radius $r = n^\rho$ for constant $\rho \in (0,\frac{1}{2})$. Then, G admits a recursive partition $\mathcal{P}$ such that asymptotically almost surely the algorithm from Theorem 6.3 computes the diameter of G in time $\tilde{O}\left(n^{\max(\frac{3}{2}+3\rho,2-\frac{2}{3}\rho)}\right)$. If the ground space of G is the square $\mathcal{S}$, the running time is in $\tilde{O}\left(n^{\max(\frac{3}{2}+3\rho,2-\frac{10}{3}\rho)}\right)$. ◁

Proof. Asymptotically almost surely, we can rely on the properties summarized above. The running time guarantee from Theorem 6.3 depends on a parameter k such that the recursive partition contains a flat partition with blocks of size k and diameter in $\Omega(d_{\text{local}})$ and optionally in $\Omega(d_{\text{corner}})$. We have $\Omega(d_{\text{local}}) = \Omega(d_{\text{corner}}) = \Omega(n^{\frac{1}{2}-\frac{7}{3}\rho}+1)$. This means that there is a phase transition at $\rho = \frac{3}{14}$, above which d_{local} and d_{corner} are no longer growing in n. We thus consider the two cases $\rho < \frac{3}{14}$ and $\rho \geq \frac{3}{14}$ separately.

For the first case, we use the recursive partition $G[\mathcal{P}^\ell]$ with $\ell = (1-x)\log_4 n$ for $x = \frac{6}{14}$, i.e., leaves of $\mathcal{P}$ have size $n^{\frac{6}{14}}$ and diameter $n^{\frac{3}{14}-\rho}$. Then, for $k = n^{1-\frac{8}{3}\rho}$ blocks of size k have diameter $n^{\frac{1}{2}-\frac{7}{3}\rho} \in \Omega(d_{\text{local}}) = \Omega(d_{\text{corner}})$. Also, we have $1-\frac{8}{3}\rho > 1-\frac{8}{3}\cdot\frac{3}{14} = \frac{6}{14}$, and thus $k \in \Omega(n^x)$, i.e., $\mathcal{P}$ contains flat partitions of size $\Theta(k)$.

With this choice for the parameters, the running time for torus RGGs given by Theorem 6.3 is in

$$\tilde{O}\Big(n^{1+\alpha+\beta}\cdot d + \min\big\{nkd + k^{2\alpha}n^{1+2\beta} + k^{2\alpha-1}n^{1+\alpha+3\beta},\ kn^{1+\alpha+\beta}\big\}\Big),$$

with $\alpha = \frac{1}{2}$, $\beta = \rho$, $d = n^{2\rho}$. Considering each term separately, we have

$$\begin{aligned}
n^{1+\alpha+\beta}\cdot d &= n^{1+\frac{1}{2}+\rho}\cdot n^{2\rho} = n^{\frac{3}{2}+3\rho},\\
nkd &= n\cdot n^{1-\frac{8}{3}\rho}\cdot n^{2\rho} = n^{2-\frac{2}{3}\rho},\\
k^{2\alpha}n^{1+2\beta} &= n^{1-\frac{8}{3}\rho}\cdot n^{1+2\rho} = n^{2-\frac{2}{3}\rho},\\
n^{1+\alpha+3\beta}k^{2\alpha-1} &= n^{1+\frac{1}{2}+3\rho}k^0 = n^{\frac{3}{2}+3\rho},\\
kn^{1+\alpha+\beta} &= n^{1-\frac{8}{3}\rho}\cdot n^{1+\frac{1}{2}+\rho} = n^{\frac{5}{2}-\frac{5}{3}\rho}.
\end{aligned}$$

For $0 < \rho < \frac{3}{14}$ the term $kn^{1+\alpha+\beta} = n^{\frac{5}{2}-\frac{5}{3}\rho}$ is always larger than the other terms in the minimum. This means that the running time is in

$$\tilde{O}\Big(n^{\max\left(\frac{3}{2}+3\rho,\,2-\frac{2}{3}\rho\right)}\Big).$$

Square RGGs additionally have d_{corner}-few corners, so Theorem 6.3 gives a running time in

$$\tilde{O}\Big(n^{1+\alpha+\beta}\cdot d + \min\big\{k^2d + k^{1+2\alpha}n^{2\beta} + k^{2\alpha}n^{\alpha+3\beta},\ k^2n^{\alpha+\beta}\big\}\Big),$$

with $\alpha = \frac{1}{2}$, $\beta = \rho$, $d = n^{2\rho}$, and $k = n^{1-\frac{8}{3}\rho}$. We again consider each term separately. We have

$$\begin{aligned}
n^{1+\alpha+\beta}\cdot d &= n^{\frac{3}{2}+3\rho},\\
k^2d &= n^{2(1-\frac{8}{3}\rho)}d = n^{2-\frac{16}{3}\rho}d = n^{2-\frac{10}{3}\rho},\\
k^{1+2\alpha}n^{2\beta} &= n^{2(1-\frac{8}{3}\rho)}\cdot n^{2\rho} = n^{2-\frac{10}{3}\rho},\\
n^{\alpha+3\beta}k^{2\alpha} &= n^{\frac{1}{2}+3\rho}\cdot n^{1-\frac{8}{3}\rho} = n^{\frac{3}{2}+\frac{1}{3}\rho},
\end{aligned}$$

$$k^2 n^{\alpha+\beta} = n^{2(1-\frac{8}{3}\rho)} \cdot n^{\frac{1}{2}+\rho} = n^{\frac{5}{2}-\frac{13}{3}\rho}.$$

Similar to the torus case, for $0 < \rho < \frac{3}{14}$ the term $n^{\frac{5}{2}-\frac{5}{3}\rho}$ is always larger than the other terms in the minimum. This means that the running time is in

$$\tilde{O}\left(n^{\max(2-\frac{10}{3}\rho, \frac{3}{2}+3\rho)}\right).$$

For the case $\rho \geq \frac{3}{14}$, let $\varepsilon < \frac{1}{2} - \rho$ be a positive constant.[15] Then, $\rho + \varepsilon < \frac{1}{2}$ and $2(\rho + \varepsilon) < 1$. We use a recursive partition $G[P^\ell]$ with leaf size $n^{2\rho+\frac{\varepsilon}{2}}$ and subgraph sizes $k = n^{2\rho+\varepsilon}$. Then, the diameter of size k blocks is $d_k \in \Theta(n^{\frac{2\rho+\varepsilon}{2}-\rho}) = \Theta(n^{\varepsilon/2})$ and thus we have $d_k \in \Omega(\min\{d_{\text{local}}, d_{\text{corner}}\}) = \Omega(1)$. Again, considering each term of the running time separately, we have

$$\begin{aligned}
n^{1+\alpha+\beta} \cdot d &= n^{\frac{3}{2}+3\rho}, \\
nkd &= n \cdot n^{2\rho+\varepsilon} \cdot n^{2\rho} = n^{1+4\rho+\varepsilon}, \\
k^{2\alpha} n^{1+2\beta} &= n^{2\rho+\varepsilon} \cdot n^{1+2\rho} = n^{1+4\rho+\varepsilon}, \\
n^{1+\alpha+3\beta} k^{2\alpha-1} &= n^{1+\frac{1}{2}+3\rho} k^0 = n^{\frac{3}{2}+3\rho}, \\
k^2 n^{\alpha+\beta} &= n^{2(2\rho+\varepsilon)} \cdot n^{\frac{1}{2}+\rho} = n^{\frac{1}{2}+5\rho+2\varepsilon}
\end{aligned}$$

for torus RGGs. However, we have $\rho + \varepsilon < \frac{1}{2}$ thus $n^{\frac{3}{2}+3\rho}$ dominates $n^{1+4\rho+\varepsilon}$ and $n^{\frac{1}{2}+5\rho+2\varepsilon}$. Together with the running time analysis for the case $\rho < \frac{3}{14}$, this means that for any value of $\rho \in (0, \frac{1}{2})$, the running time on torus RGGs is in

$$\tilde{O}\left(n^{\max(\frac{3}{2}+3\rho, 2-\frac{2}{3}\rho)}\right).$$

For square RGGs we have

$$\begin{aligned}
n^{1+\alpha+\beta} \cdot d &= n^{1+\frac{1}{2}+\rho} \cdot n^{2\rho} = n^{\frac{3}{2}+3\rho}, \\
k^2 d &= n^{2(2\rho+\varepsilon)} d = n^{4\rho+2\varepsilon} n^{2\rho} = n^{6\rho+2\varepsilon}, \\
k^{1+2\alpha} n^{2\beta} &= k^2 n^{2\rho} = n^{6\rho+2\varepsilon}, \\
n^{\alpha+3\beta} k^{2\alpha} &= n^{\frac{1}{2}+3\rho} \cdot n^{2\rho+\varepsilon} = n^{\frac{1}{2}+5\rho+\varepsilon}, \\
k^2 n^{\alpha+\beta} &= n^{\frac{1}{2}+5\rho+2\varepsilon}.
\end{aligned}$$

15 For the positivity, recall that we generally assume $\rho < \frac{1}{2}$, see Section 6.4.1.

Again, with $\rho + \varepsilon < 0.5$ we have $\frac{1}{2} + 5\rho + \varepsilon < \frac{3}{2} + 3\rho$ and $\frac{1}{2} + 5\rho + 2\varepsilon < \frac{3}{2} + 3\rho$. Together with the running time analysis for the case $\rho < \frac{3}{14}$, this means that for any value of $\rho \in (0, \frac{1}{2})$, the running time on square RGGs is in

$$\tilde{O}\left(n^{\max\left(\frac{3}{2}+3\rho, 2-\frac{10}{3}\rho\right)}\right). \qquad \blacksquare$$

Equivalently the running times can be written as $\tilde{O}\left(n^{\max(\frac{1}{2}+\rho, 1-\frac{8}{3}\rho)}m\right)$ (torus) and $\tilde{O}\left(n^{\max(\frac{1}{2}+\rho, 1-\frac{16}{3}\rho)}m\right)$ (square). The following theorem follows directly, as RGGs with connection radius $r = n^\rho$ have expected average degree $\Theta(n^{2\rho})$ (see Lemma 6.35).

▷ **Theorem 6.2.** On RGGs with expected average degree $\Theta(n^\delta)$ for constant $\delta \in (0, 1)$, asymptotically almost surely the diameter can be computed in $\tilde{O}(n^{\frac{3}{2}(1+\delta)} + n^{2-\frac{1}{3}\delta})$ time for torus RGGs, respectively $\tilde{O}(n^{\frac{3}{2}(1+\delta)} + n^{2-\frac{5}{3}\delta})$ time for square RGGs. ◁

6.4.6 Analysis of iFUB

In this section we rely on the stretch bounds and the concentration of the vertices to analyze the running time of the iFUB algorithm on random geometric graphs. We consider iFUB with the *2-sweep heuristic*, i.e., the algorithm chooses a central vertex as follows. First, the algorithm performs a BFS from an arbitrary vertex v and picks a vertex w in the last layer, i.e., with maximum distance from v. Then, a second BFS is performed from w and the vertex c is chosen half the way on a shortest path between w and a vertex with maximum distance from w.

In the following, we begin by showing that the vertex w selected by the first BFS is likely located close to a corner of the square ground space.

Analysis of 2-sweep. We begin with a geometric argument showing that for any point there is a corner that is further away than a second point not close to any corner. See also Figure 6.8 for a visualization.

▷ **Lemma 6.37.** Let S be a square and let p and q be points on S, such that q has distance at least x from every corner of S. Then, there is a corner c^* of S such that $d_{\mathbb{E}}(p, q) + 0.23x \leq d_{\mathbb{E}}(p, c^*)$, i.e., the distance from p to c^* is at least $0.23x$ longer than the distance from p to q. ◁

Proof. Without loss of generality we assume that S is the unit square $[0, 1) \times [0, 1)$. Further, we can assume that $p = (p_x, p_y)$ lies in the upper left diagonal half of the

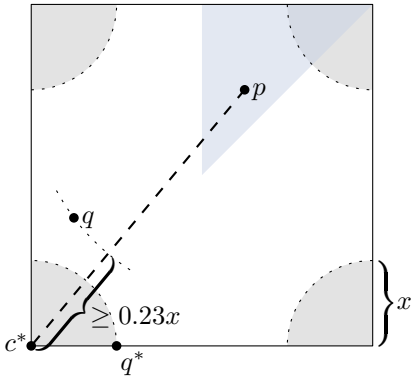


Figure 6.8: Visualization of the setting in Lemma 6.37. Without loss of generality we assume that p is located in the blue triangle. If q lies far from every corner, it is closer to p than the furthest corner c^*, additionally q^* marks the position of q that maximizes the distance to p. Without loss of generality we assume that p is located in the blue triangle.

upper right quadrant of S, i.e., $p_x \geq \frac{1}{2}$ and $p_y \geq p_x$. See also Figure 6.8. We choose c^* as the bottom left corner.

The permissive region for q is obtained from S by removing quarter circles of radius x centered at the corners of S. We first consider the case $x \leq \frac{1}{2}$. Then, with p in the upper left diagonal half of the upper right quadrant of S, the furthest position q^* of q is at the intersection of the bottom side of S and the bottom left quarter circle.

We have $|pq| \leq |pq^*| = \sqrt{(p_x - x)^2 + p_y^2}$ and $|pc^*| = \sqrt{p_x^2 + p_y^2}$. Thus we have

$$|pc^*| - |pq| \geq \sqrt{p_x^2 + p_y^2} - \sqrt{(p_x - x)^2 + p_y^2}.$$

This difference is increasing in p_x and decreasing in p_y. Therefore, it is minimized at $p = (\frac{1}{2}, 1)$, giving us $|pc^*| - |pq| \geq \frac{\sqrt{5}}{2} - \sqrt{(\frac{1}{2} - x)^2 + 1}$.

We consider $\left(\frac{\sqrt{5}}{2} - \sqrt{(\frac{1}{2} - x)^2 + 1}\right)/x$ and find that it is decreasing in x. This means that the expression has its minimum of $\sqrt{5} - 2$ at $x = \frac{1}{2}$. We have thus shown

$$(|pc^*| - |pq^*|)/x \geq \sqrt{5} - 2 > 0.23,$$

which implies $|pq| + 0.23x \leq |pc^*|$ as claimed.

For $x > 0.5$ the worst-case position of q is at $q^* = (\frac{1}{2}, \sqrt{x^2 - \frac{1}{2}^2})$ and the worst-

case position of p is still $(\frac{1}{2}, 1)$. We have

$$\frac{|pc^*| - |pq^*|}{x} \geq \frac{\frac{\sqrt{5}}{2} - \left(1 - \sqrt{x^2 - \frac{1}{4}}\right)}{x} = \frac{\frac{\sqrt{5}}{2} - 1 + \sqrt{x^2 - \frac{1}{4}}}{x}.$$

This is at least $\sqrt{5} - 2$ for all $x > \frac{1}{2}$, which concludes the proof. ■

We apply this to show that in a square RGG the furthest neighbor of every vertex lies close to a corner of the square.

▷ **Lemma 6.38.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a square random geometric graph with $r \in \omega(\log^{3/4} n)$. Consider a vertex $v \in V(G) \cap S$ and a maximally distant $w \in N(v, \mathrm{ecc}(v))$ of v. Then, asymptotically almost surely, the geometric distance of w to some corner of $\mathcal{S}$ is in $O(n^{\frac{1}{2}} r^{-\frac{4}{3}} + \sqrt{\log n})$. ◁

Proof. Without loss of generality, we assume v to be in the upper right quadrant of $\mathcal{S}$ and we show that $w \in N(v, \mathrm{ecc}(v))$ has geometric distance in $O(n^{\frac{1}{2}} r^{-\frac{4}{3}})$ from the lower left corner c^* of $\mathcal{S}$ located at the origin.

We condition on the stretch event of Lemma 6.20, which holds asymptotically almost surely. Then every pair of vertices with geometric distance at least $d \geq r \ln n$ has graph distance at most $\lceil \frac{d}{r} \cdot (1 + s) \rceil$ for some $s \in O(r^{-\frac{4}{3}})$.

Assume towards a contradiction that w has distance more than $\varepsilon = 5\sqrt{2ns} + 5\sqrt{\ln n}$ from all corners of $\mathcal{S}$. We show that then there is another vertex with higher graph distance than w from v. By Lemma 6.25, there is a vertex w' in G with distance from the origin at most $\sqrt{\ln n}$ asymptotically almost surely.

It remains to show that w' has higher distance from v than w in the graph. By Lemma 6.37 we have

$$d_{\mathbb{E}}(v, w) \leq d_{\mathbb{E}}(v, c^*) - 0.23\varepsilon.$$

Further, by the triangle inequality,

$$d_{\mathbb{E}}(v, c^*) \leq d_{\mathbb{E}}(v, w') + d_{\mathbb{E}}(w', c^*) \leq d_{\mathbb{E}}(v, w') + \sqrt{\ln n}.$$

Combining these two inequalities we derive

$$\begin{aligned} d_{\mathbb{E}}(v, w) &\leq d_E(v, w') + \sqrt{\ln n} - 0.23\varepsilon \\ &= d_{\mathbb{E}}(v, w') + \sqrt{\ln n} - 0.23 \cdot (5\sqrt{2ns} + 5\sqrt{\ln n}) \\ &< d_{\mathbb{E}}(v, w') - 1.15\sqrt{2ns}. \end{aligned}$$

As $d_{\mathbb{E}}(v,w')$ is at most the diagonal of $\mathcal{S}$, $\sqrt{2n}$, we further have

$$d_{\mathbb{E}}(v,w) < d_{\mathbb{E}}(v,w') - 1.15\, d_{\mathbb{E}}(v,w')\, s = d_{\mathbb{E}}(v,w')(1 - 1.15\, s).$$

We now compare the graph theoretic distance from v to w and to w'. By Equation (6.4) we get a lower bound

$$\operatorname{dist}_G(v,w') \geq \left\lceil \frac{d_{\mathbb{E}}(v,w')}{r} \right\rceil.$$

Applying the stretch bounds, we further get

$$\begin{aligned}
\operatorname{dist}_G(v,w) &\leq \left\lceil \frac{d_{\mathbb{E}}(v,w)}{r}(1+s) \right\rceil \\
&\leq \left\lceil \frac{d_{\mathbb{E}}(v,w')}{r}(1-1.15s)(1+s) \right\rceil \\
&< \left\lceil \frac{d_{\mathbb{E}}(v,w')}{r} \right\rceil.
\end{aligned}$$

This means that $\operatorname{dist}_G(v,w') > \operatorname{dist}_G(v,w)$, contradicting the assumption that no other vertex has higher distance from v as w. We conclude that asymptotically almost surely for every vertex v, every vertex $w \in N(v, \operatorname{ecc}(v))$ has distance at most $\varepsilon = 5\sqrt{2ns} + 5\sqrt{\ln n}$ from c^*. ■

This means that a 2-sweep gives a good lower bound for the diameter of square RGGs. Additionally, we show that on square RGGs a central vertex chosen this way is located close to the geometric center of the square.

▷ **Lemma 6.39.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a square random geometric graph with $r \in \omega(\log^{3/4} n)$ and let v_c be the central vertex chosen after a 2-sweep. Then, asymptotically almost surely, v_c lies within a geometric distance of $O(n^{\frac{1}{2}} r^{-\frac{2}{3}} + \sqrt{\log n})$ from the geometric center of $\mathcal{S}$. ◁

Proof. Let $v \in V(G)$ be an arbitrary starting vertex, let $w \in N_G(v, \operatorname{ecc}(v))$ be a maximally distant vertex from v, and let $w' \in N_G(w, \operatorname{ecc}(w))$ be maximally distant from w. Further let v_c with $|\operatorname{dist}_G(w,v_c) - \operatorname{dist}_G(w',v_c)| \leq 1$ and $\operatorname{dist}_G(w,v_c) + \operatorname{dist}_G(w',v_c) = \operatorname{dist}_G(w,w')$ the central vertex chosen with the 2-sweep. We condition on the stretch event, i.e., in the following for every pair of vertices u, v with geometric distance d in $\omega(r \log n)$ we can assume $d_G(u,v) \leq \frac{d}{r}(1+s)$, with $s \in O(r^{-\frac{4}{3}})$. By Lemma 6.38, w and w' each lie within geometric distance of $O(n^{\frac{1}{2}} r^{-\frac{4}{3}} + \sqrt{\log n})$ from some corner

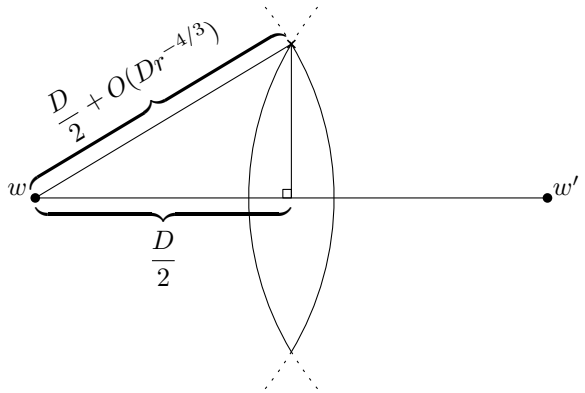


Figure 6.9: Visualization of lens and right triangle in proof of Lemma 6.39.

of $\mathcal{S}$. As the opposite corners have geometric distance $\sqrt{2n}$ and all other corners have distance at most $\sqrt{n}$ it follows that w and w' lie within geometric distance of $O(n^{\frac{1}{2}}r^{-\frac{4}{3}} + \sqrt{\log n})$ from opposite corners of $\mathcal{S}$. With the approximate location of w and w' known it remains to locate v_c.

Denote the Euclidean midpoint of the segment ww' by m and its length $d_{\mathbb{E}}(w,w')$ as D. For $x \in \{w,w'\}$, we have $\text{dist}_G(x,v_c) \leq \left\lceil \frac{dist_G(w,w')}{2} \right\rceil \leq \frac{D}{2r}(1+s)+1$. Using Equation (6.4) this implies $\text{dist}_{\mathbb{E}}(x,v_c) \leq \frac{D}{2}(1+s)+2 = \frac{D}{2} + O(Dr^{-\frac{4}{3}})$. Thus v_c lies in the lens formed by the intersection of circles of radius $\frac{D}{2} + O(Dr^{-\frac{4}{3}})$ centered at w and w'. Along the line through w and w', the lens has length $O(Dr^{-4/3})$. To bound the width h in the orthogonal direction, we need the distance from m to either of intersection point of the circles, see also Figure 6.9. Observe that x, m, and either intersection point form a right triangle. By the Pythagorean theorem,

$$\left(\frac{D}{2} + O(Dr^{-\frac{4}{3}})\right)^2 = \left(\frac{D}{2}\right)^2 + h^2.$$

To bound h from above, set $a = \frac{D}{2}$ and $b = O(Dr^{-\frac{4}{3}})$. Then

$$h = \sqrt{(a+b)^2 - a^2} = \sqrt{2ab + b^2} = \sqrt{2ab\left(1 + \frac{b}{2a}\right)} = \sqrt{2ab}\sqrt{1 + \frac{b}{2a}},$$

and thus, using that $\sqrt{1+x} \leq 1 + \frac{x}{2}$ holds for $x > 0$,

$$h \leq \sqrt{2ab}\left(1 + \frac{b}{4a}\right).$$

We have $\sqrt{2ab} \in O(D \cdot r^{-\frac{2}{3}})$, $\frac{b}{4a} \in O(r^{-\frac{4}{3}})$ and hence

$$h \leq D \cdot r^{-\frac{2}{3}}(1 + O(r^{-\frac{4}{3}}))$$

With $D \in O(\sqrt{n})$, it follows that v_c lies within a Euclidean distance of $O(D \cdot r^{-\frac{2}{3}})$ from m. As w and w' are within $O(n^{\frac{1}{2}} \cdot r^{-\frac{4}{3}} + \sqrt{\log n})$ from the corners, m is also within $O(n^{\frac{1}{2}} \cdot r^{-\frac{4}{3}} + \sqrt{\log n})$ from the geometric center of $\mathcal{S}$. Combining these bounds, we conclude that v_c is located within $O(n^{\frac{1}{2}} \cdot r^{-\frac{2}{3}} + \sqrt{\log n})$ from the geometric center. ■

Analysis of iFUB. We briefly explain how the algorithm proceeds after selecting a central vertex c, see also [Cre+13]. Let $v_1, \ldots, v_n = c$ be an ordering of the vertices sorted in descending order of their distance to c. Such an ordering is easily obtained after running a BFS from c. To find the diameter, the algorithm computes $\mathrm{ecc}(v_i)$ for each vertex v_i in this sequence and maintains the largest found eccentricity as a lower bound, i.e., $L_i = \max_{0 \leq j \leq i} \mathrm{ecc}(v_j)$. The algorithm stops and reports L_i as the diameter, once $2 \cdot \mathrm{dist}(c, v_i) \leq L_i$. To see why this is correct, note that there exists a diametrical vertex s with $\mathrm{dist}(c, s) \geq \lceil \frac{\mathrm{diam}_G}{2} \rceil$ and that the stopping criterion ensures that such a vertex has been processed.

As discussed in the introduction of this chapter, the running time of iFUB depends on the choice of the central vertex and the metric structure of the graph. To be exact, the running time depends on the number of vertices with distance at least half the diameter from c. This has already been observed and used in the literature [BCT17], but to the best of our knowledge not formally proved. We consequently give a complete argument below.

▷ **Lemma 6.40.** Let G be a graph with diameter D and let c be the central vertex for iFUB. Then, iFUB explores every vertex with distance at least $\lceil \frac{D}{2} \rceil + 1$ from c and every explored vertex has distance at least $\lceil \frac{D}{2} \rceil$ from c. ◁

Proof. We begin with the first direction, i.e., we show that a vertex v with $\mathrm{dist}(c, v) \geq \lceil \frac{D}{2} \rceil + 1$ is explored by iFUB. Let $L_i \leq D$ be the value of the lower bound at the time when iFUB decides whether to explore v. Then we have $2\,\mathrm{dist}(w, c) \geq \frac{D}{2} + 2 > L_i$, i.e., iFUB explores w.

For the other direction, let w be a vertex that is explored by iFUB. Then, at the time when iFUB explores w, we have $2\,\mathrm{dist}(c, w) > L_i$. If $L_i = D$, this concludes the proof, as we have $\mathrm{dist}(v, w) \geq \lceil D/2 \rceil$. If otherwise $L_i < D$, then no diametrical vertex has been explored yet. However, there is a diametrical vertex x with $\mathrm{dist}(c, x) \geq \lceil D/2 \rceil$.

As x has not yet been explored when w is explored, we have $\text{dist}(c,w) \geq \text{dist}(c,x)$ and thus also $\text{dist}(c,w) \geq \lceil D/2 \rceil$. ■

We already analyzed the 2-sweep and showed that the central vertex is likely to be located close to the geometric center of the square. It remains to show that in this case iFUB does not perform too many BFS.

▷ **Lemma 6.41.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a square random geometric graph with $r \in \omega(\log^{3/4} n)$ and let v_c be a vertex with geometric distance $h \in o(\sqrt{n})$ from the geometric center of $\mathcal{S}$. Then, with v_c as central vertex iFUB performs at most $O(h^2 + nr^{-\frac{8}{3}})$ BFS runs, asymptotically almost surely. ◁

Proof. With v_c chosen as the central vertex, iFUB performs a BFS for every vertex w with $\text{dist}_G(v_c, w) \geq \frac{\text{diam}_G}{2}$. We show that vertices close to the geometric center of $\mathcal{S}$ do not have graph distance at least $\frac{\text{diam}_G}{2}$. Conversely, the vertices from which a iFUB runs a BFS lie in regions far from the geometric center. We show that these regions do not contain many vertices. We condition on the stretch event, which holds a.a.s. (Lemma 6.20).

We choose $d = \frac{\sqrt{n}}{\sqrt{2}} - x$ for some $x > 0$ to be specified later and consider a vertex u with geometric distance at most d from the center of $\mathcal{S}$. Then $\text{dist}_{\mathbb{E}}(v_c, u) \leq \frac{\sqrt{n}}{\sqrt{2}} - x + h$ and thus $\text{dist}_G(v_c, u) \leq \left(\frac{\sqrt{n}}{\sqrt{2}} - x + h\right) r^{-1}(1 + s)$ for $s \in O(r^{-\frac{4}{3}})$. By Lemma 6.26 we have $\text{diam}_G \geq \frac{\sqrt{2n}}{r}$. Thus, we can choose $x \in \Theta\left(h + \sqrt{n}s\right)$ such that $\text{dist}_G(v_c, u) < \frac{\text{diam}_G}{2}$.

By Lemma 6.40 iFUB only runs BFS from vertices with distance at least $\frac{\text{diam}_G}{2}$ from v_c. This means that any such vertex has distance at least $\frac{\sqrt{n}}{\sqrt{2}} - x$ from the geometric center of $\mathcal{S}$. Comparing this with Lemma 6.31, we get that any such vertex has distance at most $O(h + \sqrt{n}s)$ from a corner of $\mathcal{S}$ and thus lies in a region with area $O(h^2 + n \cdot s^2)$. The number of vertices in this region is in $O(h^2 + n \cdot s^2)$ with high probability by Lemma 6.22, which concludes the proof. ■

Together with Lemma 6.41 this results in the following running time bound, which is (truly) subquadratic for (polynomially) growing r.

▷ **Lemma 6.42.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a square random geometric graph with $r \in \omega(\log^{3/4} n)$. Then, asymptotically almost surely 2-sweep iFUB has running time in $O((nr^{-\frac{4}{3}} + \log n)m)$. ◁

Proof. By Lemma 6.39, the central vertex v_c chosen after the 2-sweep has distance at most $h \in O(n^{\frac{1}{2}} r^{-\frac{2}{3}} + \log^{\frac{1}{2}} n)$ from the geometric center of $\mathcal{S}$, asymptotically almost

surely. Thus by Lemma 6.41 iFUB performs only

$$O(h^2 + n \cdot r^{-\frac{8}{3}}) = O(nr^{-\frac{4}{3}} + \log n)$$

BFS runs, asymptotically almost surely. ■

We also want to show a lower bound for the running time on torus RGGs. For this, we use that the iFUB algorithm performs a BFS for all vertices with distance at least $\lceil \frac{\text{diam}_G}{2} \rceil + 1$ from the central vertex v_c. By observing that on torus RGGs there are many vertices at such a distance from any chosen central vertex, this gives us a linear lower bound for the number of BFS runs.

▷ **Lemma 6.43.** Let $G \sim \mathcal{G}(\mathcal{T}, n, r)$ be a torus random geometric graph with $r \in \omega(\log^{3/4} n)$. Then, asymptotically almost surely, for every central vertex, iFUB performs $\Omega(n)$ BFS runs. ◁

Proof. Let v_c be the central vertex for iFUB. Then iFUB performs a BFS for any vertex w with $\text{dist}_G(v_c, w) \geq \lceil \frac{\text{diam}_G}{2} \rceil + 1$ (Lemma 6.40). By Lemma 6.27, we have $\text{diam}_G \leq \frac{\sqrt{n}}{\sqrt{2}r}(1+s)$ with $s \in O(r^{-\frac{4}{3}})$, asymptotically almost surely. By Equation (6.4), for a vertex w with

$$\text{dist}_{\mathbb{E}}(v_c, w) \geq \frac{\sqrt{n}}{2\sqrt{2}}(1 + 1.1s) + 2r$$

we have $\text{dist}_G(v_c, w) \geq \frac{\sqrt{n}}{2\sqrt{2}r}(1+1.1s)+2 > \lceil \frac{\text{diam}_G}{2} \rceil + 1$. It remains to show that many vertices have such a distance from v_c. The geometric disk of radius $\frac{\sqrt{n}}{2\sqrt{2}}(1+1.1s)+2r$ around v_c has area

$$\frac{\pi n}{8}\left(1 + 2.2s + 1.21s^2 + \frac{16r}{\sqrt{2n}}(1 + 1.1s) + \frac{32r^2}{n}\right),$$

where $s, s^2, \frac{r}{n}$, and r^2/n are all in $o(1)$. The entire torus $\mathcal{T}$ has area n and thus the region of $\mathcal{T}$ where vertices are chosen as BFS sources by iFUB has area at least $(1 - \frac{\pi}{8})n(1 - o(1)) \geq 0.6n(1 - o(1))$. As the number of vertices within such a region is sufficiently concentrated by Lemma 6.22, this means that iFUB performs $\Omega(n)$ BFS runs and thus has a running time in $\Omega(nm)$. ■

Together, the above two lemmas give the following.

▷ **Theorem 6.1.** Let $G \sim \mathcal{G}(\mathcal{S}, n, r)$ be a square random geometric graph with expected average degree $d \in \Omega(\log^{\frac{3}{2}} n)$. Then, a.a.s., 2-sweep iFUB has running

time in $O((nd^{-\frac{2}{3}} + \log n)m)$. If $G \sim \mathcal{G}(\mathcal{T}, n, r)$ is a torus RGG with expected average degree in $\Omega(\log^{\frac{3}{2}} n)$, then a.a.s. for every choice of the central vertex, the running time of iFUB is in $\Omega(nm)$. ◁

6.5 Conclusion and Open Problems

In this chapter we gave a set of natural deterministic properties allowing for efficient diameter computation and demonstrated that these properties a.a.s. hold on square and torus RGGs. We note that our formulation of the properties is not the only possible one, but represents a trade-off between simplicity and generality. To show this, we point out multiple possible generalizations for the assumptions used in our algorithm.

In Property 1 we demand that for all $x > 0$ and every vertex v, the x-diametric partners lie in $O(1)$ balls of radius $O(x + d_{\text{local}})$. Here, the linear dependence on x was mostly chosen for its simplicity. By considering how the property is used (e.g. Lemma 6.15) one can see that this dependence can be significantly relaxed. For instance, one could demand that the radius of the balls has some arbitrary non-decreasing dependency $f(x)$ on x, or even depends on n and x as $f_n(x)$. Then in Theorem 6.3, the new requirement is that blocks of size k need to have diameter in $\Omega(f_n(x))$. Similarly, instead of requiring a constant number of balls, one could also specify the number of balls as a parameter, which then appears as an additional factor in the running time of Theorem 6.3. Both of these generalizations apply analogously to Property 2. Moreover, for Property 5 one could allow a non-constant parameter for the number of intersecting blocks, which then appears as an additional factor in the number of candidate pairs and thus the running time.

Regardless of such details about possible more general formulations of our properties, it would be interesting to complement our theoretical results with an empirical evaluation of both the proposed algorithm as well as its underlying assumptions. For the algorithm, it would be interesting whether an implementation using a heuristically chosen recursive partition can outcompete other known approaches on graphs with underlying geometry. Regarding the deterministic properties, it would be interesting to verify to which degree generated and real-world networks fulfill Properties 1–5.

We also want to point out some directions for improvement regarding the analysis on random geometric graphs. For the application of our algorithm on RGGs we assumed that the algorithm receives the graph along with a suitable recursive partition, see Theorem 6.2. While Section 6.4 demonstrates that such a partition is obtained very easily by subdividing the graph along its geometry, it would be

interesting to also give an algorithm that finds a suitable partition using only a combinatorial representation of the graph. We believe that a simple approach based on graph Voronoi diagrams should already work. Unfortunately, the known stretch bounds (Lemmas 6.20 and 6.21) do not give satisfying bounds for the size of (recursive) graph Voronoi separators in random geometric graphs. Can this challenge be overcome or is it maybe possible to find a different approach that is easier to analyze? It would also be rewarding to strengthen our bounds on the radius dependence of Property 1 beyond our analysis in Section 6.4.4 and to consider generalizations to other settings like smaller average degrees, other ground spaces, or higher dimensions. Conditional lower bounds rule out subquadratic algorithms for 3-D unit-ball graphs [Bri+22], so extending our results to 3-D random geometric graphs might show a separation between the worst-case and average-case setting. Finally, our running time analysis for iFUB on square RGGs is likely pessimistic and better stretch bounds or more generally a better understanding of the distribution of graph distances can be expected to improve this.

7 Conclusions & Outlook

In this thesis we explored the use of deterministic properties as a method for algorithm analysis. As outlined in Chapter 1, our motivation was to leverage additional assumptions about the input of algorithms in order to construct better models for their performance and explain the surprising tractability of practical instances. Going beyond univariate parameterized analysis, we explicitly allow more complex assumptions about the input and do not require the hardness of practical instances to be encoded within a single parameter. In contrast to average-case analysis, deterministic properties do not rely on an underlying probability distribution and are already defined on individual inputs.

Overall, we demonstrated the usefulness and applicability of deterministic properties as a method for algorithm analysis across a wide range of settings. Throughout Chapters 3–6 we showed how the method can be applied to both hard and efficiently solvable problems, how it can be useful both for the analysis of known algorithms as well as the design of new ones, and how it can further provide non-randomized alternatives to probabilistic models and explanations for algorithm efficiency.

In the following, we conclude with a brief summary, discussion, and comparison with other methods. Additionally, we discuss open problems and directions for future work.

7.1 Discussion

In Chapter 3 we demonstrated that even within the setting of classical parameterized complexity and FPT algorithms, an empirical mindset provides useful directions for the design of new and practically relevant parameters. The proposed parameter clique-partitioned treewidth can be smaller than treewidth in practice, while retaining algorithmic potential. It also connects naturally to the class of hyperbolic uniform disk graphs considered in Chapter 4. On these graphs, separators can be covered with few cliques, implying small clique-partitioned treewidth. Hyperbolic uniform disk graphs also form a deterministic generalization of hyperbolic random graphs, which allowed us to re-derive a previous average-case result about solving the vertex cover problem in polynomial time, without relying on the randomization of vertex positions. Next, in Chapter 5 we used deterministic properties to explain

asymptotic speedups observed for the bidirectional BFS algorithm. Notably, we experimentally showed that our theoretical conditions for sublinear running time are not only fulfilled on many real-world graphs, but are also good predictors for the algorithm's performance. Finally, in Chapter 6 we gave deterministic properties that capture certain combinatorial features of graphs with underlying geometry and used these as an interface between our newly proposed diameter algorithm and its average-case analysis on random geometric graphs.

One common thread present throughout this thesis is the trade-off between weaker versus stronger assumptions for the analysis of algorithms. Worst-case analysis sits on one end of the spectrum, considering all possible instances of a problem. It thus gives the most general upper bounds for the performance of an algorithm, at the cost of potentially being pessimistic. Parameterized analysis still considers arbitrary inputs, but classifies the complexity of problem instances according to a parameter. Thus, any parameterized attempt to explain the practical performance of an algorithm crucially hinges on the plausibility of the considered parameter being small. While parameterized complexity often brings valuable structural insights, it is often difficult to find parameterizations that both capture practical instances and enable algorithmic improvements. Average-case analysis considers a weighted average of an algorithm's performance, thus making statements about what happens on typical instances as given by the probability distribution defining the weighting. This allows the definition of richer input models, such as the random network models discussed in Chapter 1. At best, such models combine properties of practical instances while being mathematically tractable, thus enabling convincing explanations for the practically observed performance of algorithms. One downside of average-case analysis is that it only makes statements about the whole probability distribution or typical samples from it. Predictions about individual problem instances are not possible and can at best be derived indirectly. With deterministic properties for algorithm analysis, this thesis proposes a way to combine the advantages of both parameterized analysis and average-case analysis. Concretely, deterministic properties allow richer input models, while still making statements about individual instances and avoiding the assumption of randomness.

Of course, even though deterministic properties address some of the downsides of average-case analysis and (univariate) parameterizations, any particular analysis of an algorithm depends on the chosen assumptions used to restrict the input. Weaker assumptions lead to stronger and more general mathematical statements, but may fail to accurately reproduce practical observations. Stronger assumptions allow the definition of richer input models, but require additional justification. Throughout the different chapters, this thesis presents a range of different assumptions, roughly

ordered by their complexity. Chapter 3 uses a single numerical parameter that can be defined on any possible input. Chapter 4 also uses a single numerical parameter (the radius r), but additionally makes assumptions about an underlying geometry. Moreover, here the radius parameter is simply describing structural properties and not trying to encapsulate algorithmic complexity. Despite the connection between HUDGs and clique-partitioned treewidth, the algorithm for the independent set problem developed in Chapter 4 does not fit into the framework of fixed-parameter tractability, but instead depends on the parameter in a more complex way. Chapters 5 and 6 go further by specifying more elaborate properties that rely on multiple parameters and structural assumptions about the input. Overall, we find that such more complex properties make it much easier to achieve both sufficient flexibility to capture relevant practical inputs and rigidity with strong algorithmic consequences. When picking assumptions from a larger space of more complex candidates, the final choice needs careful justification. In the two case studies this is achieved with an empirical validation and using an established random network model. In particular, the analysis of the bidirectional search algorithm shows that more complex properties can give very specific, accurate, and predictive models for the practical performance of algorithms.

A natural concern with this approach is that stronger and more complex structural assumptions might come at the cost of less elegant or less broadly applicable theoretical results. While this thesis argues that this can be a worthwhile trade-off considering the potential for better empirical grounding, our case studies also suggest that this criticism need not hold in general. Chapter 5 shows that even complex assumptions relying on multiple parameters have a potential for elegant and satisfying theoretical results, by proving that the third condition for sublinear running time is in fact tight. Moreover, while Chapter 6 presents an algorithm for a restricted geometric setting, the use of deterministic properties makes the result more portable, as shown by its application to multiple variants of random geometric graphs. Overall, we argue that the ability of more complex assumptions to define richer input models makes them sufficiently useful to overcome the burden of needing additional justification. Importantly, even if specific assumptions might be debatable, they can still be better and more informative than the implicit assumptions behind a worst-case analysis.

Taken together, the preceding chapters argue for deterministic properties as a useful method for the analysis of algorithms that addresses several shortcomings of both average-case and classical parameterized methods. None of this implies, however, that they should replace the other approaches. There are many algorithms for which worst-case analysis without specific assumptions about the input already represents an adequate model for algorithm performance. Likewise, random input

models such as the ones for real-world networks discussed in Chapter 1 have the advantage that they are generative and can produce instances amenable to experimental study. Furthermore, even the analysis of univariate input parameters often brings valuable theoretical insights, similar to, for instance, the study of restricted graph classes.

To conclude, in this thesis we investigated the use of deterministic properties as a method for algorithm analysis. We argued that this perspective complements average-case analysis and classical parameterized approaches, and demonstrated deterministic properties as a useful tool that can contribute towards narrowing the gap between theory and practice. We believe that the technique offers significant unexplored potential and we hope that it will inspire further progress and insights.

7.2 Outlook

Chapters 3–6 each end with a discussion of open problems and directions for future research. In the following, we give pointers to a few more general directions that go beyond those directly connected to the respective chapters.

Deterministic models for complex networks. First, while random models for complex networks have proven highly successful at capturing structural properties of real-world networks (see also Chapter 1, Page 3), deterministic models for complex networks are still somewhat lacking. Such models should be able to explain and advance algorithmic performance on similar real-world networks in a convincing way, ideally by allowing empirical verification of the underlying assumptions. Even though hyperbolic uniform disk graphs Chapter 4 inherit some plausibility through their connection to hyperbolic random graphs, they fall short on the latter aspect. In fact, deciding whether a given graph is a HUDG is $\exists\mathbb{R}$-Complete [Bie+23; Jun24]. Thus even though the structural and algorithmic results from Chapter 4 are interesting on their own as well as in connection to hyperbolic random graphs, it is very hard to verify the underlying assumptions on real-world networks.

It would thus be desirable to develop deterministic properties capturing complex real-world networks that use easier to verify assumptions. Recall that Chapter 1 discussed three papers that use deterministic properties to formalize power-law distributed degrees, clustering, or the small world property in order to derive algorithmic results [BCT17; Bra+16; Fox+20]. Unfortunately, as discussed on Pages 8–9 each of these results comes with some disadvantages. Thus one first step towards more convincing deterministic models for complex networks would be to address some of these limitations. Concretely, this could involve the following directions.

To expand upon [Bra+16], one could study conditions that capture power-law degree distributions without including regular or similarly homogeneous graphs. Going beyond [BCT17] it would be interesting to study the considered algorithms and heuristics for metric properties of graphs also in geometric settings without a power-law distribution. While Chapter 6 goes in this direction and also develops a new algorithm, it otherwise only analyzes the iFUB algorithm on random geometric graphs. Finally, it remains to find explanations for the practical feasibility of enumerating all maximal cliques in a graph that go beyond those of [Fox+20] and also address the empirical observations of [BF24].

Further algorithmic applications. Beyond using deterministic properties to formulate assumptions that capture specific types of practically occurring inputs such as complex networks, we also want to motivate the study of concrete algorithmic problems. One natural starting point for this could be previous algorithmic analyses on random input models such as HRGs. There, in addition to the bidirectional BFS algorithm [Blä+22b] and the vertex cover problem [Blä+23a] which we addressed in Chapters 4 and 5, recent studies have also considered vertex cover approximation and the chromatic number [BFK23; MR26]. How fast can these problems be solved on other deterministic input models? It would also be interesting to better understand the effectiveness of reduction rules for the vertex cover problem.

Beyond these examples of problems already studied on HRGs, a second direction would be to consider algorithms and problems with well-known or large gaps between theory and practice. First, consider the PACE-challenge.[16] A common pattern among successful implementations is that they make heavy use of heuristics especially for data reduction or pruning [Blä+22a; HS18; Tam17; Wie24]. Is it possible to identify deterministic properties that help explain the effectiveness of such heuristics? One particularly intriguing case is treewidth, where compared with earlier iterations of the PACE-challenge, subsequent advances brought very large improvements [Tam17; Tam22]. At the same time, the effectiveness of even basic heuristics remains elusive, despite some initial analysis [BHS03].

Another example of a wide gap between theory and practice is given by the Louvain algorithm for community detection [Blo+08], a heuristic approach that partitions a given graph into dense clusters with few edges between different clusters. Based on a greedy local search, the Louvain algorithm has been observed to be very efficient in practice despite the existence of worst-case instances with running time in $\Omega(n^2)$ [BF24]. While there exists an average-case analysis that shows linear running times on the stochastic block model with two communities [Coh+20], it

16 https://pacechallenge.org/

remains open to transfer this to other (for instance geometric) settings. An analysis using empirically verifiable deterministic properties, might provide an even clearer explanation of the algorithm's practical performance.

More speculatively, a natural long-term question is whether deterministic properties as a methodology can be successfully applied beyond graph algorithms. For instance, boolean satisfiability features a large gap between worst-case theory and practical algorithm performance, while also having natural graph-theoretic structure. It is conceivable that deterministic structural properties of instances could help to find explanations for the surprising performance of modern solvers on practical instances. Overall, we hope that the case studies presented in this thesis demonstrate the potential of deterministic properties as a methodology and serve as a starting point for further progress towards a better theoretical understanding of practical algorithm performance.

Bibliography

[AB98] Mohamad A. Akra and Louay Bazzi. **On the Solution of Linear Recurrence Equations**. *Computational Optimization and Applications* 10:2 (1998), 195–210. DOI: 10.1023/A:1018373005182 (see page 108).

[AFB17] Mohammed Amin Abdullah, Nikolaos Fountoulakis, and Michel Bode. **Typical distances in a geometric model for complex networks**. *Internet Math.* 2017 (2017). DOI: 10.24166/IM.13.2017 (see page 94).

[Alo+24] Noga Alon, Allan Grønlund, Søren Fuglede Jørgensen, and Kasper Green Larsen. **Sublinear Time Shortest Path in Expander Graphs**. In: *Proceedings of the 49th International Symposium on Mathematical Foundations of Computer Science (MFCS'2024)*. Vol. 306. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024, 8:1–8:13. DOI: 10.4230/LIPIcs.MFCS.2024.8 (see page 95).

[AM05] S.A. Aldosari and J.M.F. Moura. **Distributed Detection in Sensor Networks: Connectivity Graph and Small World Networks**. In: *Conference Record of the 39th Asilomar Conference on Signals, Systems and Computers, 2005.* 2005, 230–234. DOI: 10.1109/ACSSC.2005.1599738 (see page 97).

[AMW17] Michael Abseher, Nysret Musliu, and Stefan Woltran. **htd - A Free, Open-Source Framework for (Customized) Tree Decompositions and Beyond**. In: *Proceedings of the 14th International Conference Integration of AI and OR Techniques in Constraint Programming (CPAIOR'2017)*. Vol. 10335. Lecture Notes in Computer Science. Springer, 2017, 376–386. DOI: 10.1007/978-3-319-59776-8_30 (see page 33).

[Ari+96] Srinivasa Rao Arikati, Danny Z. Chen, L. Paul Chew, Gautam Das, Michiel H. M. Smid, and Christos D. Zaroliagis. **Planar Spanners and Approximate Shortest Path Queries among Obstacles in the Plane**. In: *Proceedings of the 4th Annual European Symposium on Algorithms (ESA'1996)*. Vol. 1136. Lecture Notes in Computer Science. Springer, 1996, 514–528 (see page 107).

[Aro21] Chris Aronis. **The Algorithmic Complexity of Tree-Clique Width**. *CoRR* abs/2111.02200 (2021). URL: https://arxiv.org/abs/2111.02200 (see page 17).

[AVW16] Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. **Approximation and Fixed Parameter Subquadratic Algorithms for Radius and Diameter in Sparse Graphs**. In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2016)*. Society for Industrial and Applied Mathematics, 2016, 377–391. DOI: 10.1137/1.9781611974331.ch28 (see pages 6, 40).

[BA99] Albert-László Barabási and Réka Albert. **Emergence of Scaling in Random Networks**. *Science* 286:5439 (1999), 509–512. DOI: 10.1126/science.286.5439.509 (see page 4).

[Bas+16] Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. "Route Planning in Transportation Networks." In: *Algorithm Engineering - Selected Results and Surveys*. Ed. by Lasse Kliemann and Peter Sanders. Vol. 9220. Lecture Notes in Computer Science. Springer International Publishing, 2016, 19–80. URL: https://doi.org/10.1007/978-3-319-49487-6%5C_2 (see page 68).

[BCT17] Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. **An Axiomatic and an Average-Case Analysis of Algorithms and Heuristics for Metric Properties of Graphs**. In: *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2017)*. Society for Industrial and Applied Mathematics, 2017, 920–939. DOI: 10.1137/1.9781611974782.58 (see pages 2, 8, 9, 11, 98, 146, 154, 155).

[Ber+20] Mark de Berg, Hans L. Bodlaender, Sándor Kisfaludi-Bak, Dániel Marx, and Tom C. van der Zanden. **A Framework for Exponential-Time-Hypothesis-Tight Algorithms and Lower Bounds in Geometric Intersection Graphs**. *SIAM Journal on Computing* 49:6 (2020), 1291–1331. DOI: 10.1137/20M1320870 (see pages 10, 15–19, 21, 62–65).

[Ber23] Mark de Berg. **A Note on Reachability and Distance Oracles for Transmission Graphs**. *Computing in Geometry and Topology* 2:1 (May 2023), 4:1–4:15. DOI: 10.57717/cgt.v2i1.25 (see page 107).

[BF00] Michael A. Bender and Martin Farach-Colton. **The LCA Problem Revisited**. In: *Proceedings of the 4th Latin American Symposium (LATIN'2000)*. Vol. 1776. Lecture Notes in Computer Science. Springer, 2000, 88–94. DOI: 10.1007/10719839_9 (see page 108).

[BF22] Thomas Bläsius and Philipp Fischbeck. *3006 Networks (unweighted, undirected, simple, connected) from Network Repository*. Zenodo, May 2022. DOI: 10.5281/zenodo.6586185 (see pages 34, 85).

[BF24] Thomas Bläsius and Philipp Fischbeck. **On the External Validity of Average-case Analyses of Graph Algorithms**. *ACM Transactions on Algorithms* 20:1 (Jan. 2024). URL: https://doi.org/10.1145/3633778 (see pages 2, 3, 5, 9, 11, 33, 34, 85, 93, 97, 98, 155).

[BFK16] Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. **Hyperbolic Random Graphs: Separators and Treewidth**. In: *Proceedings of the 24th Annual European Symposium on Algorithms (ESA'2016)*. Vol. 57. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016, 15:1–15:16. DOI: 10.4230/LIPIcs.ESA.2016.15 (see pages 16, 44).

[BFK18] Thomas Bläsius, Tobias Friedrich, and Anton Krohmer. **Cliques in Hyperbolic Random Graphs**. *Algorithmica* 80:8 (2018), 2324–2344. DOI: 10.1007/s00453-017-0323-3 (see pages 5, 44).

[BFK23] Thomas Bläsius, Tobias Friedrich, and Maximilian Katzmann. **Efficiently Approximating Vertex Cover on Scale-Free Networks with Underlying Hyperbolic Geometry**. *Algorithmica* 85:12 (2023), 3487–3520. DOI: 10.1007/S00453-023-01143-X (see pages 5, 155).

[BFW21] Thomas Bläsius, Tobias Friedrich, and Christopher Weyand. **Efficiently Computing Maximum Flows in Scale-Free Networks**. In: *Proceedings of the 29th Annual European Symposium on Algorithms (ESA'2021)*. Vol. 204. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, 21:1–21:14. DOI: 10.4230/LIPIcs.ESA.2021.21 (see page 2).

[BHS03] Anne Berry, Pinar Heggernes, and Geneviève Simonet. **The Minimum Degree Heuristic and the Minimal Triangulation Process**. In: *Proceedings of the 29th Workshop on Graph-Theoretic Concepts in Computer Science (WG'2003)*. Lecture Notes in Computer Science. Springer, 2003, 58–70. DOI: 10.1007/978-3-540-39890-5_6 (see page 155).

[Bie+23] Nicholas Bieker, Thomas Bläsius, Emil Dohse, and Paul Jungeblut. **Recognizing Unit Disk Graphs in Hyperbolic Geometry is $\exists\mathbb{R}$-Complete**. *CoRR* (2023). DOI: 10.48550/ARXIV.2301.05550 (see page 154).

[BJR07] Béla Bollobás, Svante Janson, and Oliver Riordan. **The phase transition in inhomogeneous random graphs**. *Random Structures & Algorithms* 31:1 (Aug. 2007), 3–122. DOI: 10.1002/rsa.20168 (see page 4).

[BKL19] Karl Bringmann, Ralph Keusch, and Johannes Lengler. **Geometric inhomogeneous random graphs**. *Theoretical Computer Science* 760 (2019), 35–54. DOI: 10.1016/j.tcs.2018.08.014 (see pages 5, 33, 91).

[BKL25] Karl Bringmann, Ralph Keusch, and Johannes Lengler. **Average distance in a general class of scale-free networks**. *Advances in Applied Probability* 57:2 (2025), 371–406. DOI: 10.1017/apr.2024.43 (see page 94).

[BKW23] Thomas Bläsius, Maximilian Katzmann, and Marcus Wilhelm. **Partitioning the Bags of a Tree Decomposition into Cliques**. In: *Proceedings of the 21st International Symposium on Experimental Algorithms (SEA'2023)*. Vol. 265. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 3:1–3:19. DOI: 10.4230/LIPIcs.SEA.2023.3 (see page 15).

[Blä+21] Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. **PACE Solver Description: The KaPoCE Exact Cluster Editing Algorithm**. In: *Proceedings of the 16th International Symposium on Parameterized and Exact Computation (IPEC'2021)*. Vol. 214. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 27:1–27:3. DOI: 10.4230/LIPICS.IPEC.2021.27 (see page 7).

[Blä+22a] Thomas Bläsius, Philipp Fischbeck, Lars Gottesbüren, Michael Hamann, Tobias Heuer, Jonas Spinner, Christopher Weyand, and Marcus Wilhelm. **A Branch-And-Bound Algorithm for Cluster Editing**. In: *Proceedings of the 20th International Symposium on Experimental Algorithms (SEA'2022)*. Vol. 233. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 13:1–13:19. DOI: 10.4230/LIPICS.SEA.2022.13 (see pages 7, 155).

[Blä+22b] 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**. *ACM Transactions on Algorithms* 18:2 (2022), 19:1–19:32. URL: https://doi.org/10.1145/3516483 (see pages 5, 44, 67, 68, 71, 91, 155).

[Blä+22c] Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, Ulrich Meyer, Manuel Penschuck, and Christopher Weyand. **Efficiently generating geometric inhomogeneous and hyperbolic random graphs**. *Netw. Sci.* 10:4 (2022), 361–380. DOI: 10.1017/NWS.2022.32 (see pages 5, 33, 91).

[Blä+22d] Thomas Bläsius, Tobias Friedrich, David Stangl, and Christopher Weyand. **An Efficient Branch-and-Bound Solver for Hitting Set**. In: *Proceedings of the 24th Symposium on Algorithm Engineering and Experiments (ALENEX'2022)*. Society for Industrial and Applied Mathematics, 2022, 209–220. DOI: 10.1137/1.9781611977042.17 (see page 27).

[Blä+23a] Thomas Bläsius, Philipp Fischbeck, Tobias Friedrich, and Maximilian Katzmann. **Solving Vertex Cover in Polynomial Time on Hyperbolic Random Graphs**. *Theory of Computing Systems* 67:1 (2023), 28–51. DOI: 10.1007/S00224-021-10062-9 (see pages 5, 10, 43, 44, 65, 155).

[Blä+23b] Thomas Bläsius, Tobias Friedrich, Maximilian Katzmann, and Daniel Stephan. **Strongly Hyperbolic Unit Disk Graphs**. In: *40th International Symposium on Theoretical Aspects of Computer Science (STACS'2023)*. Vol. 254. LIPIcs.

Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, 13:1–13:17. DOI: 10.4230/LIPICS.STACS.2023.13 (see pages 43, 45, 46).

[Blä+24] Thomas Bläsius, Jean-Pierre von der Heydt, Sándor Kisfaludi-Bak, Marcus Wilhelm, and Geert van Wordragen. **Structure and Independence in Hyperbolic Uniform Disk Graphs**. *Computing Research Repository (CoRR)* (2024). DOI: 10.48550/arXiv.2407.09362 (see pages 45, 51, 64, 65).

[Blä+25] Thomas Bläsius, Jean-Pierre von der Heydt, Sándor Kisfaludi-Bak, Marcus Wilhelm, and Geert van Wordragen. **Structure and Independence in Hyperbolic Uniform Disk Graphs**. In: *Proceedings of the 41st Annual Symposium on Computational Geometry (SoCG'2025)*. Vol. 332. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025, 21:1–21:16. DOI: 10.4230/LIPICS.SOCG.2025.21 (see pages 16, 43, 46, 50, 51, 63–65).

[Blä+26] Thomas Bläsius, Emil Dohse, Deborah Haun, and Laura Merker. **Product Structure and Treewidth of Hyperbolic Uniform Disk Graphs**. In: *Proceedings of the 42nd Annual Symposium on Computational Geometry (SoCG'2026)*. Vol. 367. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2026, 18:1–18:17. DOI: 10.4230/LIPIcs.SoCG.2026.18 (see pages 16, 46).

[Blo+08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. **Fast unfolding of communities in large networks**. *Journal of Statistical Mechanics: Theory and Experiment* 2008:10 (Oct. 2008), P10008. DOI: 10.1088/1742-5468/2008/10/P10008 (see pages 2, 155).

[BMT03] Vincent Bouchitté, Frédéric Mazoit, and Ioan Todinca. **Chordal embeddings of planar graphs**. *Discrete Mathematics* 273:1-3 (2003), 85–102. DOI: 10.1016/S0012-365X(03)00230-9 (see page 51).

[BN19] Michele Borassi and Emanuele Natale. **KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation**. *Journal of Experimental Algorithmics* 24:1 (2019), 1.2:1–1.2:35. URL: https://doi.org/10.1145/3284359 (see pages 5, 67, 68, 71).

[Böc+09] S. Böcker, S. Briesemeister, Q.B.A. Bui, and A. Truss. **Going weighted: Parameterized algorithms for cluster editing**. *Theoretical Computer Science* 410:52 (2009). Combinatorial Optimization and Applications, 5467–5480. DOI: https://doi.org/10.1016/j.tcs.2009.05.006 (see page 7).

[Böc12] Sebastian Böcker. **A golden ratio parameterized algorithm for Cluster Editing**. *Journal of Discrete Algorithms* 16 (2012). Selected papers from the 22nd International Workshop on Combinatorial Algorithms (IWOCA 2011), 79–89. DOI: https://doi.org/10.1016/j.jda.2012.04.005 (see page 7).

[Bod98] Hans L. Bodlaender. **A Partial *k*-Arboretum of Graphs with Bounded Treewidth**. *Theoretical Computer Science* 209:1-2 (1998), 1–45. DOI: 10.1016/S0304-3975(97)00228-4 (see pages 19, 51, 58).

[Bol+01] Béla Bollobás, Oliver Riordan, Joel Spencer, and Gábor Tusnády. **The degree sequence of a scale-free random graph process**. *Random Structures & Algorithms* 18:3 (2001), 279–290. DOI: https://doi.org/10.1002/rsa.1009. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rsa.1009 (see page 4).

[Bol01a] Béla Bollobás. **Random Graphs**. 2nd ed. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 2001. URL: http://doi.org/10.1017/CBO9780511814068 (see pages 3, 5).

[Bol01b] Béla Bollobás. **The Diameter**, 251–281. In: *Random Graphs*. Cambridge Studies in Advanced Mathematics. Chapter 10. Cambridge University Press, 2001. DOI: 10.1017/CBO9780511814068.012 (see page 4).

[Bon+22] Édouard Bonnet, Eun Jung Kim, Stéphan Thomassé, and Rémi Watrigant. **Twin-width I: Tractable FO Model Checking**. *Journal of the ACM* 69:1 (2022), 3:1–3:46. DOI: 10.1145/3486655 (see page 41).

[Bra+16] Paweł Brach, Marek Cygan, Jakub Łącki, and Piotr Sankowski. **Algorithmic Complexity of Power Law Networks**. In: *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2016)*. Society for Industrial and Applied Mathematics, 2016, 1306–1325. DOI: 10.1137/1.9781611974331.CH91 (see pages 8, 11, 154, 155).

[Bri+22] Karl Bringmann, Sándor Kisfaludi-Bak, Marvin Künnemann, André Nusser, and Zahra Parsaeian. **Towards Sub-Quadratic Diameter Computation in Geometric Intersection Graphs**. In: *Proceedings of the 38th Annual Symposium on Computational Geometry (SoCG'2022)*. Vol. 224. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 21:1–21:16. DOI: 10.4230/LIPIcs.SoCG.2022.21 (see page 150).

[BSW26] Thomas Bläsius, Annemarie Schaub, and Marcus Wilhelm. **Diameter Computation on (Random) Geometric Graphs**. *Computing Research Repository (CoRR)* (2026). DOI: 10.48550/arXiv.2603.16684 (see page 97).

[BW23] Thomas Bläsius and Marcus Wilhelm. **Deterministic Performance Guarantees for Bidirectional BFS on Real-World Networks**. In: *Procedings of the 34th International Workshop on Combinatorial Algorithms (IWOCA'2023)*. Vol. 13889. Lecture Notes in Computer Science. Springer, 2023, 99–110. DOI: 10.1007/978-3-031-34347-6_9 (see page 67).

[BW26] Thomas Bläsius and Marcus Wilhelm. **Deterministic Performance Guarantees for Bidirectional BFS on Real-World Networks**. *Journal of Computer and System Sciences* 159 (2026), 103774. DOI: 10.1016/j.jcss.2026.103774 (see page 67).

[CER93] Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. **Handle-Rewriting Hypergraph Grammars**. *Journal of Computer and System Sciences* 46:2 (1993), 218–270. DOI: 10.1016/0022-0000(93)90004-G (see page 41).

[Cha+07] Augustin Chaintreau, Abderrahmen Mtibaa, Laurent Massoulié, and Christophe Diot. **The diameter of opportunistic mobile networks**. In: *Proceedings of the 2007 ACM Conference on Emerging Network Experiment and Technology (CoNEXT'2007)*. ACM Press, 2007, 12. DOI: 10.1145/1364654.1364670 (see page 98).

[Cha+25] Timothy M. Chan, Hsien-Chih Chang, Jie Gao, Sándor Kisfaludi-Bak, Hung Le, and Da Wei Zheng. **Truly Subquadratic Time Algorithms for Diameter and Related Problems in Graphs of Bounded VC-dimension**. In: *Proceedings of the 66th Annual Symposium on Foundations of Computer Science (FOCS'2025)*. IEEE, 2025, 2728–2765. DOI: 10.1109/FOCS63196.2025.00140 (see pages 11, 99, 100).

[Chu86] F. R. K. Chung. *Diameters of communication networks*. 1986. DOI: 10.1090/psapm/034/846852 (see page 97).

[CL02a] Fan Chung and Linyuan Lu. **Connected Components in Random Graphs with Given Expected Degree Sequences**. *Annals of Combinatorics* 6:2 (Nov. 2002), 125–145. DOI: 10.1007/PL00012580 (see page 4).

[CL02b] Fan Chung and Linyuan Lu. **The average distances in random graphs with given expected degrees**. *Proceedings of the National Academy of Sciences* 99:25 (2002), 15879–15882. DOI: 10.1073/pnas.252631999. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.252631999 (see page 94).

[Coh+20] Vincent Cohen-Addad, Adrian Kosowski, Frederik Mallmann-Trenn, and David Saulpic. **On the Power of Louvain in the Stochastic Block Model**. In: *Proceedings of the 33rd Conference on Advances in Neural Information Processing Systems (NeurIPS'2020)*. 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/29a6aa8af3c942a277478a90aa4cae21-Abstract.html (see page 155).

[Coo71] Stephen A. Cook. **The complexity of theorem-proving procedures**. In: *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing (STOC'1971)*. Association for Computing Machinery, 1971, 151–158. DOI: 10.1145/800157.805047 (see page 2).

[Cou90] Bruno Courcelle. **The monadic second-order logic of graphs. I. Recognizable sets of finite graphs**. *Information and Computation* 85:1 (Mar. 1990), 12–75. DOI: 10.1016/0890-5401(90)90043-H (see page 6).

[Cre+13] Pilu Crescenzi, Roberto Grossi, Michel Habib, Leonardo Lanzi, and Andrea Marino. **On computing the diameter of real-world undirected graphs**. *Theoretical Computer Science* 514 (2013), 84–95. DOI: 10.1016/J.TCS.2012.09.018 (see pages 98, 146).

[Cyg+15] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. **Parameterized Algorithms**. Springer International Publishing, 2015. ISBN: 978-3-319-21274-6. DOI: 10.1007/978-3-319-21275-3 (see pages 6, 15, 21).

[Dal+26] Clément Dallard, Fedor V. Fomin, Petr A. Golovach, Tuukka Korhonen, and Martin Milanic. **Computing Tree Decompositions with Small Independence Number**. *ACM Transactions on Algorithms* 22:1 (2026), 12:1–12:25. DOI: 10.1145/3767730 (see pages 17, 41).

[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: *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC'2016)*. Vol. 63. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017, 30:1–30:9. DOI: 10.4230/LIPIcs.IPEC.2016.30 (see page 16).

[Del+18] Holger Dell, Christian Komusiewicz, Nimrod Talmon, and Mathias Weller. **The PACE 2017 Parameterized Algorithms and Computational Experiments Challenge: The Second Iteration**. In: *Proceedings of the 12th International Symposium on Parameterized and Exact Computation (IPEC'2017)*. Vol. 89. LIPIcs. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018, 30:1–30:12. DOI: 10.4230/LIPIcs.IPEC.2017.30 (see pages 2, 16).

[Den07] Peter J. Denning. **Computing is a natural science**. *Commun. ACM* 50:7 (July 2007), 13–18. DOI: 10.1145/1272516.1272529 (see page 1).

[DF13] Rodney G. Downey and Michael R. Fellows. **Fundamentals of Parameterized Complexity**. Texts in Computer Science. Springer, 2013. ISBN: 978-1-4471-5558-4. DOI: 10.1007/978-1-4471-5559-1 (see pages 1, 6, 8).

[DFH19] M. Ayaz Dzulfikar, Johannes K. Fichte, and Markus Hecher. **The PACE 2019 Parameterized Algorithms and Computational Experiments Challenge: The Fourth Iteration**. In: *Proceedings of the 14th International Symposium on Parameterized and Exact Computation (IPEC'2019)*. Vol. 148. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, 25:1–25:23. DOI: 10.4230/LIPIcs.IPEC.2019.25 (see page 2).

[DHH10] Sander Dommers, Remco van der Hofstad, and Gerard Hooghiemstra. **Diameters in Preferential Attachment Models**. *Journal of Statistical Physics* 139:1 (Apr. 2010), 72–107. DOI: 10.1007/s10955-010-9921-z (see page 4).

[Día+16] J. Díaz, D. Mitsche, G. Perarnau, and X. Pérez-Giménez. **On the relation between graph distance and Euclidean distance in random geometric graphs**. *Advances in Applied Probability* 48:3 (2016), 848–864. DOI: 10.1017/apr.2016.31 (see pages 119, 120, 122, 123).

[DMŠ21] Clément Dallard, Martin Milanič, and Kenny Štorgel. **Treewidth versus Clique Number. I. Graph Classes with a Forbidden Structure**. *SIAM Journal on Discrete Mathematics* 35:4 (2021), 2618–2646. DOI: 10.1137/20M1352119 (see page 17).

[DMŠ24a] Clément Dallard, Martin Milanič, and Kenny Štorgel. **Treewidth versus clique number. II. Tree-independence number**. *Journal of Combinatorial Theory, Series B* 164 (2024), 404–442. DOI: 10.1016/J.JCTB.2023.10.006 (see page 17).

[DMŠ24b] Clément Dallard, Martin Milanič, and Kenny Štorgel. **Treewidth versus clique number. III. Tree-independence number of graphs with a forbidden structure**. *Journal of Combinatorial Theory, Series B* 167 (2024), 338–391. DOI: 10.1016/J.JCTB.2024.03.005 (see page 17).

[Dor+10] Frederic Dorn, Eelko Penninkx, Hans L. Bodlaender, and Fedor V. Fomin. **Efficient Exact Algorithms on Planar Graphs: Exploiting Sphere Cut Decompositions**. *Algorithmica* 58:3 (2010), 790–810. DOI: 10.1007/s00453-009-9296-1 (see page 53).

[ELS13] David Eppstein, Maarten Löffler, and Darren Strash. **Listing All Maximal Cliques in Large Sparse Real-World Graphs**. *Journal of Experimental Algorithmics* 18 (2013). DOI: 10.1145/2543629 (see page 23).

[ER59] P. Erdős and A. Rényi. **On Random Graphs I**. *Publicationes Mathematicae* 6 (1959), 290–297. URL: https://www.renyi.hu/~p_erdos/1959-11.pdf (see pages 3, 68).

[ER60] Paul Erdős and Alfréd Rényi. **On the Evolution of Random Graphs**. *Publications of the Mathematical Institute of the Hungarian Academy of Sciences* 5 (1960), 17–61 (see page 3).

[FK14] Arash Farzan and Shahin Kamali. **Compact Navigation and Distance Oracles for Graphs with Small Treewidth**. *Algorithmica* 69:1 (2014), 92–116 (see page 107).

[FK18] Tobias Friedrich and Anton Krohmer. **On the Diameter of Hyperbolic Random Graphs**. *SIAM Journal on Discrete Mathematics* 32:2 (2018), 1314–1334. DOI: 10.1137/17M1123961 (see pages 4, 43).

[Fox+20] Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. **Finding Cliques in Social Networks: A New Distribution-Free Model**. *SIAM Journal on Computing* 49:2 (2020), 448–464. DOI: 10.1137/18M1210459 (see pages 8, 9, 11, 154, 155).

[Gei23] Sven Geißler. **Exploring Clique-Partitioned Treewidth**. Advisor: Marcus Wilhelm, Reviewers: Thomas Bläsius and Torsten Ueckerdt. Bachelor's Thesis. Karlsruhe Institute of Technology, 2023. URL: https://scale.iti.kit.edu/_media/resources/theses/bachelorarbeit_sven_geissler.pdf (see pages 15, 20).

[Gil59] E. N. Gilbert. **Random Graphs**. *The Annals of Mathematical Statistics* 30:4 (Dec. 1959), 1141–1144. DOI: 10.1214/aoms/1177706098 (see page 3).

[GJ79] M. R. Garey and David S. Johnson. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. W. H. Freeman, 1979. ISBN: 0-7167-1044-7 (see pages 2, 23).

[GMN17] Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. **Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs**. *Theoretical Computer Science* 689 (2017), 67–95. DOI: 10.1016/J.TCS.2017.05.017 (see page 6).

[GPP12] Luca Gugelmann, Konstantinos Panagiotou, and Ueli Peter. **Random Hyperbolic Graphs: Degree Sequence and Clustering**. In: *Proceedings of the 39th International Colloquium on Automata, Languages and Programming (ICALP'2012)*. 2012, 573–585. DOI: 10.1007/978-3-642-31585-5_51 (see pages 4, 43).

[Gur23] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2023. URL: https://www.gurobi.com (see page 27).

[GV21] Vijay Ganesh and Moshe Y. Vardi, 547–566. In: *Beyond the Worst-Case Analysis of Algorithms*. Ed. by Tim Roughgarden. Cambridge University Press, 2021. DOI: 10.1017/9781108637435.032 (see page 2).

[Har14] Sariel Har-Peled. **Quasi-Polynomial Time Approximation Scheme for Sparse Subsets of Polygons**. In: *Proceedings of the 30th Annual Symposium on Computational Geometry (SoCG'2014)*. ACM, 2014, 120. DOI: 10.1145/2582112.2582157 (see page 63).

[HK02] Petter Holme and Beom Jun Kim. **Growing scale-free networks with tunable clustering**. *Physical Review E* 65 (2 Jan. 2002), 026107. DOI: 10.1103/PhysRevE.65.026107 (see page 4).

[HLL83] Paul W. Holland, Kathryn Blackmond Laskey, and Samuel Leinhardt. **Stochastic blockmodels: First steps**. *Social Networks* 5:2 (1983), 109–137. DOI: https://doi.org/10.1016/0378-8733(83)90021-7 (see page 4).

[Hof24] Remco van der Hofstad. **Random Graphs and Complex Networks**. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 2024. DOI: 10.1017/9781316795552 (see page 4).

[HS18] Michael Hamann and Ben Strasser. **Graph Bisection with Pareto Optimization**. *Journal of Experimental Algorithmics* 23 (Feb. 2018). DOI: 10.1145/3173045 (see page 155).

[JPY88] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. **On Generating All Maximal Independent Sets**. *Information Processing Letters* 27:3 (1988), 119–123. DOI: 10.1016/0020-0190(88)90065-8 (see page 23).

[Jun24] Paul Jungeblut. **Capturing the Computational Complexity of Geometric Problems by the First-Order Theory of the Reals**. PhD thesis. Karlsruher Institut für Technologie (KIT), 2024. DOI: 10.5445/IR/1000170385 (see page 154).

[Kar72] Richard M. Karp. **Reducibility Among Combinatorial Problems**. In: *Proceedings of a symposium on the Complexity of Computer Computations*. The IBM Research Symposia Series. Plenum Press, New York, 1972, 85–103. DOI: 10.1007/978-1-4684-2001-2_9 (see page 2).

[Kar83] Richard M. Karp. **The probabilistic analysis of combinatorial optimization algorithms**. In: *International Congress of Mathematicians (ICM)*. Vol. 2. 1983, 1601–1609. URL: https://www.mathunion.org/fileadmin/ICM/Proceedings/ICM1983.2/ICM1983.2.ocr.pdf (see page 3).

[Kat23] Maximilian Katzmann. **About the analysis of algorithms on networks with underlying hyperbolic geometry**. PhD thesis. Universität Potsdam, 2023, 8690 KB, xi, 191 pages. DOI: 10.25932/PUBLISHUP-58296 (see page 5).

[KE02] Konstantin Klemm and Víctor M. Eguíluz. **Highly clustered scale-free networks**. *Physical Review E* 65 (3 Feb. 2002), 036123. DOI: 10.1103/PhysRevE.65.036123 (see page 4).

[Kis20] Sándor Kisfaludi-Bak. **Hyperbolic intersection graphs and (quasi)-polynomial time**. In: *Proceedings of the 31st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2020)*. Society for Industrial and Applied Mathematics, 2020, 1621–1638. DOI: 10.1137/1.9781611975994.100 (see pages 10, 15–17, 43, 46, 62, 64).

[KM12] Ross J. Kang and Tobias Müller. **Sphere and Dot Product Representations of Graphs**. *Discrete & Computational Geometry* 47:3 (2012), 548–568. DOI: 10.1007/S00454-012-9394-8 (see page 100).

[Kri+10] Dmitri Krioukov, Fragkiskos Papadopoulos, Maksim Kitsak, Amin Vahdat, and Marián Boguñá. **Hyperbolic geometry of complex networks**. *Physical Review E* 82 (3 Sept. 2010), 036106. DOI: 10.1103/PhysRevE.82.036106 (see page 4).

[LCW05] Dmitri Loguinov, Juan Casas, and Xiaoming Wang. **Graph-theoretic analysis of structured peer-to-peer systems: routing distances and fault resilience**. *IEEE/ACM Trans. Netw.* 13:5 (2005), 1107–1120. DOI: 10.1109/TNET.2005.857072 (see page 98).

[Lev73] Leonid A. Levin. **Universal Sequential Search Problems**. *Problems of Information Transmission* 9:3 (1973). Translated from *Problemy Peredachi Informatsii* 9(3):115–116, 265–266 (see page 2).

[Lov05] László Lovász. **Graph minor theory**. en. *Bulletin of the American Mathematical Society* 43:1 (Oct. 2005), 75–86. DOI: 10.1090/S0273-0979-05-01088-8 (see pages 6, 15).

[LT79] Richard J. Lipton and Robert Endre Tarjan. **A separator theorem for planar graphs**. *SIAM Journal on Applied Mathematics* 36:2 (1979), 177–189 (see page 6).

[LT80] Richard J. Lipton and Robert Endre Tarjan. **Applications of a planar separator theorem**. *SIAM Journal on Computing* 9:3 (1980), 615–627. DOI: 10.1137/0209046 (see page 65).

[MP22] Dániel Marx and Michal Pilipczuk. **Optimal Parameterized Algorithms for Planar Facility Location Problems Using Voronoi Diagrams**. *ACM Transactions on Algorithms* 18:2 (2022), 13:1–13:64. DOI: 10.1145/3483425 (see pages 53, 63).

[MR26] Yannic Maus and Janosch Ruff. **On Distributed Colouring of Hyperbolic Random Graphs**. In: *Proceedings of the 37th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2026)*. Society for Industrial and Applied Mathematics, 2026, 2540–2553. DOI: 10.1137/1.9781611978971.91 (see pages 44, 155).

[MS19] Tobias Müller and Merlijn Staps. **The Diameter of KPKVB Random Graphs**. *Advances in Applied Probability* 51:2 (2019), 358–377. DOI: 10.1017/apr.2019.23 (see pages 4, 43).

[MSJ19] Silviu Maniu, Pierre Senellart, and Suraj Jog. **An Experimental Study of the Treewidth of Real-World Graph Data**. In: *Proceedings of the 22nd International Conference on Database Theory (ICDT'2019)*. Vol. 127. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 12:1–12:18. DOI: 10.4230/LIPIcs.ICDT.2019.12 (see pages 7, 10, 33).

[MU17] Michael Mitzenmacher and Eli Upfal. **Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis**. 2nd. Cambridge University Press, 2017. ISBN: 110715488X (see page 127).

[New03] M. E. J. Newman. **The Structure and Function of Complex Networks**. *SIAM Review* 45:2 (2003), 167–256. DOI: 10.1137/S003614450342480 (see page 4).

[Pen03] Mathew Penrose. **Random Geometric Graphs**. Oxford University Press, May 2003. ISBN: 978-0-19-850626-3. DOI: 10.1093/acprof:oso/9780198506263.001.0001 (see page 4).

[PY00] Behrooz Parhami and Chi-Hsiang Yeh. **Why network diameter is still important**. In: *Proceedings of the International Conference on Communications in Computing, (CIC'2000)*. 2000, 271–274 (see page 97).

[RA15] Ryan A. Rossi and Nesreen K. Ahmed. **The Network Data Repository with Interactive Graph Analytics and Visualization**. In: *Proceedings of the 29th AAAI Conference on Artificial Intelligence (AAAI'2015)*. AAAI Press, 2015, 4292–4293. URL: http://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9553 (see pages 34, 85).

[RB03] Erzsébet Ravasz and Albert-László Barabási. **Hierarchical organization in complex networks**. *Physical Review E* 67 (2 Feb. 2003), 026112. DOI: 10.1103/PhysRevE.67.026112 (see page 4).

[Rou21] Tim Roughgarden. **Beyond the Worst-Case Analysis of Algorithms**. Cambridge University Press, 2021. DOI: 10.1017/9781108637435 (see pages 1, 2, 8).

[RR95] Arlan Ramsay and Robert D. Richtmyer. **Introduction to Hyperbolic Geometry**. Universitext. Springer, 1995. ISBN: 978-0-387-94339-8. DOI: 10.1007/978-1-4757-5585-5 (see page 46).

[RS04] Neil Robertson and Paul D. Seymour. **Graph Minors. XX. Wagner's conjecture**. *Journal of Combinatorial Theory, Series B* 92:2 (2004), 325–357. DOI: 10.1016/J.JCTB.2004.08.001 (see page 6).

[RS21] Tim Roughgarden and C. Seshadhri. **28**, 606–625. In: *Beyond the Worst-Case Analysis of Algorithms*. Cambridge University Press, 2021. DOI: 10.1017/9781108637435.035 (see page 9).

[RS90] Neil Robertson and Paul D. Seymour. **Graph minors. IV. Tree-width and well-quasi-ordering**. *Journal of Combinatorial Theory, Series B* 48:2 (1990), 227–254. DOI: 10.1016/0095-8956(90)90120-O (see page 15).

[RS91] Neil Robertson and P. D Seymour. **Graph minors. X. Obstructions to tree-decomposition**. *Journal of Combinatorial Theory, Series B* 52:2 (1991), 153–190. DOI: 10.1016/0095-8956(91)90061-N (see pages 52, 58).

[Sch07] Satu Elisa Schaeffer. **Graph clustering**. *Computer Science Review* 1:1 (2007), 27–64. DOI: https://doi.org/10.1016/j.cosrev.2007.05.001 (see page 98).

[Sch24] Annemarie Schaub. **Computing the Diameter of Toroidal Random Geometric Graphs**. Advisor: Marcus Wilhelm, Reviewers: Thomas Bläsius and Marvin Künnemann. Bachelor's Thesis. Karlsruhe Institute of Technology, 2024. URL: https://scale.iti.kit.edu/_media/resources/theses/ba_annemarie_schaub.pdf (see page 97).

[Sei95] R. Seidel. **On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs**. *Journal of Computer and System Sciences* 51:3 (1995), 400–403. DOI: https://doi.org/10.1006/jcss.1995.1078 (see page 99).

[ST04] Daniel A. Spielman and Shang-Hua Teng. **Smoothed analysis of algorithms: Why the simplex algorithm usually takes polynomial time**. *Journal of the ACM* 51:3 (May 2004), 385–463. DOI: 10.1145/990308.990310 (see page 3).

[ST94] Paul D. Seymour and Robin Thomas. **Call Routing and the Ratcatcher**. *Combinatorica* 14:2 (1994), 217–241. DOI: 10.1007/BF01215352 (see page 52).

[Tam17] Hisao Tamaki. **Positive-Instance Driven Dynamic Programming for Treewidth**. In: *Proceedings of the 25th Annual European Symposium on Algorithms (ESA'2017)*. Vol. 87. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017, 68:1–68:13. DOI: 10.4230/LIPIcs.ESA.2017.68 (see pages 7, 155).

[Tam22] Hisao Tamaki. **Heuristic Computation of Exact Treewidth**. In: *Proceedings of the 20th International Symposium on Experimental Algorithms (SEA'2022)*. Vol. 233. LIPIcs. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022, 17:1–17:16. DOI: 10.4230/LIPIcs.SEA.2022.17 (see page 155).

[Weg76] Peter Wegner. **Research paradigms in computer science**. In: *Proceedings of the 2nd International Conference on Software Engineering (ICSE'76)*. IEEE Computer Society Press, 1976, 322–330 (see page 1).

[Wey23] Christopher Weyand. **Geometric Inhomogeneous Random Graphs for Algorithm Engineering**. PhD thesis. Karlsruher Institut für Technologie (KIT), 2023. 144 pp. DOI: 10.5445/IR/1000158520 (see page 5).

[Wie24] Marcel Wienöbst. *mwien/pingpong: pace-2024*. Version v1.0. See also https://github.com/mwien/pingpong. June 2024. DOI: 10.5281/zenodo.11533179 (see page 155).

[Wil20] Marcus Wilhelm. **Beating the Worst-Case: Analysis of a Practical Algorithm for Treewidth**. Advisors: Thomas Bläsius and Maximilian Katzmann, Reviewers: Tobias Friedrich and Anja Lehmann. Master's Thesis. University of Potsdam, 2020. URL: https://scale.iti.kit.edu/_media/people/marcus_thesis.pdf (see page 7).

[WS98] Duncan J. Watts and Steven H. Strogatz. **Collective dynamics of 'small-world' networks**. *Nature* 393:6684 (June 1998), 440–442. DOI: 10.1038/30918 (see page 4).

[XKY04] Jun (Jim) Xu, Abhishek Kumar, and Xingxing Yu. **On the fundamental tradeoffs between routing table size and network diameter in peer-to-peer networks**. *IEEE J. Sel. Areas Commun.* 22:1 (2004), 151–163. DOI: 10.1109/JSAC.2003.818805 (see page 97).