The Pennsylvania State University
The Graduate School

APPROXIMATION ALGORITHMS FOR GRAPH PROBLEMS

A Dissertation in
Computer Science and Engineering
by
Shiva Prasad Kasiviswanathan

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2008

The dissertation of Shiva Prasad Kasiviswanathan was reviewed and approved* by the following:

Martin Fürer
Professor of Computer Science and Engineering
Dissertation Advisor, Chair of Committee

Piotr Berman
Associate Professor of Computer Science and Engineering

Woodrow Dale Brownawell
Distinguished Professor of Mathematics

Sofya Raskhodnikova
Assistant Professor of Computer Science and Engineering

Adam Davison Smith
Assistant Professor of Computer Science and Engineering

Raj Acharya
Department Head of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

This thesis studies approximation algorithms for two fundamental problems arising in graph theory: counting copies of one graph in another graph and estimating distances in geometric graphs.

In the first part of this thesis, we look at the problem of counting the number of copies of one (template) graph in another (base) graph. This is the counting version of the subgraph isomorphism problem and is **#P**-complete. We present the first general subcase of this problem which is almost always efficiently approximable. The template families for which we almost always obtain an efficient approximation algorithm (formally captured by the notion of a fully polynomial randomized approximation scheme) include graphs of degree at most two, bounded-degree forests, bounded-width grid graphs, subdivisions of bounded-degree graphs, and major subclasses of outerplanar graphs, series-parallel and planar graphs. As a simple consequence, we also obtain a fully polynomial randomized approximation scheme for counting all cycles in a random graph (solving an open problem of Frieze and McDiarmid). We also provide a general technique which can easily be applied to proving many more similar results. We then investigate the famous **#P**-complete problem of counting perfect matchings in a graph. For this problem, we present a simple randomized algorithm. The algorithm runs in time almost linear in the size of the input adjacency matrix and is a randomized approximation scheme for almost all graphs. This algorithm is faster than the previous best algorithm of Chien by a factor on n (number of vertices in the graph).

In the second part of this thesis, we present algorithms for constructing approximate sparse representations of geometric intersection graphs. Given a collection of geometric objects, their intersection graph is an undirected graph that has one vertex per object, and connects two objects by an edge whenever the intersection of the two objects is nonempty. A disk graph is an intersection graph of a set of disks with arbitrary radii in the plane (the generalization to higher dimensions is known as a ball graph). We present the first algorithm for constructing sparse $(1+\epsilon)$-spanners for ball graphs. The algorithm takes sub-quadratic time. For the special case where all the balls have the same radius, we show that the spanner construction has complexity almost equivalent to the construction of a Euclidean minimum spanning tree. We also present efficient algorithms for answering approximate distance queries in ball graphs. Disk graphs have

been widely used to model wireless ad hoc networks. Spanners and distance queries are used for topology control and routing in these networks.

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am deeply grateful for the support and guidance of my advisor, Martin Fürer. Throughout my stay at Penn State, Martin's insightful comments and timely advice made my graduate life a fascinating and rewarding experience. I could not have had a better mentor, teacher, and collaborator.

I would like to thank Piotr Berman, Woodrow Dale Brownawell, Sofya Raskhodnikova, and Adam Smith for serving on my thesis committee.

Piotr has always shown a deep interest in my research. His excellent intuition and creative approach to problems have been a great influence on me. The weekend hikes with him were great fun and I am going to miss his company.

Adam and Sofya joining the department was the best thing that happened to me for the past two years. Adam never ceases to amaze with his clarity and ability to make the most difficult concepts appear easy. I have greatly benefited from his keen technical insights. My only regret is that we never managed to settle who among us is better at squash. I thank Sofya for the countless discussions we had. Over the past year, her advice on issues ranging from life to employment has helped me tremendously. Sofya has also helped me a lot with useful suggestions on my presentation and writing skills.

I am also indebted to my various other collaborators: Srivatsava Ranjit Ganta, Anders Hanson, Gabriel Istrate, Jieun Jeong, Homin Lee, Kobbi Nissim, Bhuvan Urgaonkar, Sudarshan Vasudevan, and Bo Zhao. It has been a pleasure working alongside them.

I am grateful to the many friends who have made my years at State College memorable, especially my roommates: Hari, Bharat, and Rajaram. Hari has been a great friend since my undergraduate days and I could not have survived these five years without him. Bharat and Rajaram have been wonderful roommates. I must also thank Aditya, Andrew, Amitayu, Arjun, Arvindhan, Guru, Heesok, Jieun, Lav, Ranjit, Ramki, Shiva (C), Srikar, and Suresh for making my graduate life exciting in their own different ways.

My most important acknowledgment is to my family. My father (Krishnamoorty Kasi Viswanathan) and mother (Vasanthi Viswanathan) have always encouraged me to pursue my passions. My brother (Harish Kasiviswanathan) and sister-in-law (Priya Unnikrishnan) have been my pillars of support. I thank them all for their love, encouragement, and everything else.

Dedication

In memory of my father Krishnamoorty Kasi Viswanathan.

Chapter 1

Introduction

A *graph* $G = (V, E)$ consists a finite set of vertices V and a finite set of unordered pairs of distinct[1] nodes called edges, and *graph theory* is the study of graphs. Many real-world situations can conveniently be described by means of graphs. For example, a communication network can be represented as a graph, where the vertices are the communication centers and the edges are the communication links. One can also assign weights to each edge of the graph to form a *weighted graph*. Weighted graphs can be used to represent situations where the pairwise connections have some numerical values. For example, a weighted graph can be used to represent a road network where the weight on each edge is the distance between the two end points. Recently, the internet offers a particularly good example where graph-theoretic concepts and methods have been used successfully to solve real-life engineering problems [12].

Ever since the birth of graph theory (historically attributed to Euler's solution of the famous Königsberg bridge problem) there has been an unstoppable flow of interesting questions about graphs. Graph-theoretic problems have been extensively studied as optimization problems under various objective functions in the computer science literature. Many of these problems turn out to be computationally intractable. In order to circumvent this hardness, we resort to approximation algorithms, that is, algorithms that run in polynomial time and always deliver provably near-optimal solutions. An α-approximation algorithm is a polynomial-time algorithm that always returns a feasible solution with an objective function value within a factor of α of the optimum.

This thesis focuses on designing approximation algorithms for several fundamental graph-theoretic problems. It has two parts. The first part presents approximation algorithms for the problem of counting copies of one graph in another. The second part presents approximation algorithms for distance problems in geometric settings. In the following two sections, we sum-

[1] Throughout this thesis we assume that the graphs are simple (with no loops or multiple edges).

marize the most important results of this thesis.

1.1 Approximation Algorithms for Graph Counting Problems

Counting is a pervasive human activity, and so it is only natural that counting problems arise in diverse fields. Examples include: statistical physics, network design, and enumerative combinatorics. One famous example is the *monomer-dimer* problem arising in statistical mechanics. The input is usually a two- or three-dimensional grid graph. The vertices of the grid can be covered with monomers (molecules occupying one vertex) and dimers (molecules occupying two adjacent vertices). The objective is to determine the number of such coverings. Counting monomer-dimer covers of a graph is equivalent to the same as counting all matchings in it.

Most of the counting problems arise in a purely mathematical context as generalizations of decision problems. Decision problems ask about the existence of some particular *certificate*, whereas the counting problems ask about the number of these certificates. Clearly, a counting problem is at least as hard as the corresponding decision problem. The complexity class **NP** captures the difficulty of finding a certificate and the complexity class **#P** captures the difficulty of counting the number of such certificates. A counting problem is contained in **#P** if its results can be represented by the number of accepting runs of a polynomial time non-deterministic Turing machine. More precisely, the class #P is defined as $\{f : \exists$ a non-deterministic polynomial time Turing machine M such that on input x, the computation tree of M has exactly $f(x)$ accepting leaves$\}$. Valiant [105] introduced this class and showed that the problem of computing the *permanent* of a matrix is complete for this class. Problems complete for **#P**-class are believed to be very difficult, as Toda's theorem [104] implies that calls to a **#P**-oracle suffice to solve any problem in the polynomial hierarchy in polynomial time.

While there are a few interesting counting problems such as counting the spanning trees in a graph or counting the number of solutions to a linear equation, that can be solved in polynomial time, most interesting counting problems are **#P**-complete. For example, counting analogs of many natural **NP**-complete problems are **#P**-complete. However, surprisingly there are also **#P**-complete problems for which the corresponding decision problem can be solved in polynomial time (is in **P**). One such example is the problem of computing the permanent of a non-negative matrix, introduced by Cauchy and Binet in their memoirs in 1812. The permanent of an $n \times n$ matrix $A = \{a(i,j)\}_{i,j}$ is defined as

$$per(A) = \sum_{\pi} \prod_{i=1}^{n} a(i, \pi(i)),$$

where the sum is over all permutations π of $\{1, 2, \ldots, n\}$. The only difference from the *determinant* is in the sign of terms of the monomial:

$$det(A) = \sum_{\pi} \text{sgn}(\pi) \prod_{i} a(i, \pi(i)),$$

where $\text{sgn}(\pi) \in \{-1, 1\}$ is the sign of permutation π. The sign of a permutation π denoted by $\text{sgn}(\pi)$ equals 1 if the number of inversions[2] in π is even and -1 if the number of inversions is odd. With its syntactic similarity to the determinant (which is computable in polynomial time by Gaussian elimination [102]), it is surprising that computing the permanent is so hard. The permanent of a Boolean matrix also has a simple combinatorial interpretation: $per(A)$ counts the perfect matchings in the bipartite graph whose bipartite adjacency matrix is A. The decision version (problem of deciding whether a bipartite graph has a perfect matching or not) can be solved in $O(n^{2.5})$ time [54].

Since most of the interesting counting problems are hard to solve exactly, most of the research has focused on designing *efficient approximation algorithms* or on finding *restrictions* of the problems under consideration that can be solved (or approximated) efficiently. This thesis pursues both of these routes for several interesting **#P**-complete problems.

Most of the algorithms presented in this thesis are randomized. Counting is an area where randomization makes a difference in our ability to (approximately) solve problems. The relevant notion of efficiency is formalized in the definition of a *fully polynomial randomized approximation scheme*, introduced by Karp and Luby [67]. A randomized algorithm is a fully polynomial randomized approximation scheme (a.k.a. FPRAS) if it has a relative error of $(1 \pm \epsilon)$ with high probability (i.e., probability tending to 1). The algorithm must run in time polynomial in both n and ϵ^{-1}, where n is the input size. Note also that we do not believe that FPRAS exist for counting versions of **NP**-complete problems, because an FPRAS can be used to distinguish instances with zero solutions from those with at least one solution with reasonable probability, thereby implying a randomized polynomial time algorithm for the decision version of this problem. Hence, we believe that FPRAS can only be devised for counting problems whose decision versions are in **BPP**. Since it is widely believed that **NP** is not contained in **BPP**, we do not expect to find such schemes for counting versions of **NP**-complete problems.

There are many examples of **#P**-complete problems where FPRAS have been obtained, including: counting matchings (monomer-dimer systems) in general graphs [62], counting matchings of arbitrary cardinality in d-dimensional lattice graphs [68], counting perfect matchings (dimer systems) in bipartite graphs [63], computing the partition function of the Ising model [61],

[2] Pair of indices i, j such that $1 \leq i < j \leq n$, and $\pi(i) > \pi(j)$.

computing the volume of a convex body [25], special cases of counting independent sets (the hard core model) [27, 106], counting bases of a matroid [33], etc. An interested reader is referred to the surveys [59, 62] on this topic.

Very closely related to the problem of approximate counting is the problem of *sampling*. The sampling problem is the following computational task. Let Ω be a very large (but finite) set of combinatorial structures (e.g., the set of feasible solutions of a combinatorial optimization problem), and let p be a probability distribution over Ω. The task is to sample an element from Ω at random according to distribution p (or a distribution close to p). Sampling problems are fundamental for statistics and Monte Carlo integration in science. Jerrum, Valiant and Vazirani [64] showed that for all *self-reducible* **NP** problems, there exists an FPRAS for counting if and only if there exists a polynomial time algorithm for (almost) uniform sampling. Informally, self-reducibility implies that solving an instance of the problem reduces to solving a few smaller instances. A similar connection between counting and sampling can also be made for the case where items are counted with weights and sampling is non-uniform [64].

1.1.1 Approximately Counting Embeddings

One of the most fundamental counting problems is that of counting combinatorial substructures of given structures, for example, counting *embeddings* of a graph into another graph. Given a template graph H and a base graph G, we call an injection φ from vertices of H to vertices of G an embedding of H into G if φ maps every edge of H to an edge of G. In other words, φ is an isomorphism between H and a subgraph (not necessarily induced) of G. Deciding whether such an injection exists is known as the *subgraph isomorphism problem*. Subgraph isomorphism is an important and general form of pattern matching. It generalizes many interesting graph problems, including Clique, Hamiltonian Path, Maximum Matching, and Shortest Path. This problem arises in application areas ranging from text processing to physics and chemistry [7, 23, 74, 100]. This counting problem is **#P**-complete. One of the goals of this thesis is to investigate under what restriction on the base graphs G and/or template graphs H does the subgraph isomorphism counting problem have fully polynomial randomized approximation schemes.

Over the past two decades, FPRAS have been found for counting the following quantities: perfect matchings in a bipartite graph [63], all matchings in a graph [61], labeled subgraphs of a given degree sequence in a bipartite graph [15], combinatorial quantities encoded by the Tutte polynomial in a dense graph [4], and Hamilton cycles in a dense graph [26]. But problems like approximating the number perfect matchings in a general graph are still open.

The question of what other special cases of this counting problem can be efficiently approximated in the worst case is very interesting. But given the apparent lack of progress, a natural

question is whether there exists an FPRAS that works well *for almost all graphs*. The statement can be made precise as: Let G_n be a graph chosen uniformly at random from the set of all n-vertex graphs. We say that a predicate $\mathcal{P}$ holds *for almost all graphs* if $\Pr[\mathcal{P}(G_n) = true] \to 1$ as $n \to \infty$. One attractive feature of random analysis is that it banishes the pessimism of worst-case analysis. The theory of **NP**-completeness casts a much smaller shadow. Some **NP**-complete problems, such as Hamiltonian cycles, become tractable. Also random models are generally close to models used for empirical testing of algorithms.

There has been a lot of interest in using random graph models for graph counting problems. One of the well-studied problem is that of counting perfect matchings in graphs. For this problem, Jerrum and Sinclair [60] have presented a simulation of a Markov chain that almost always is an FPRAS (extended to all bipartite graphs in [63]). Similar results using other approaches were obtained later in [21, 34, 88]. Another well-studied problem is that of counting Hamiltonian cycles in random digraphs. For this problem, Frieze and Suen [37] have obtained an FPRAS, and later Rasmussen [88] has presented a simpler FPRAS. Afterwards, Frieze *et al.* [35] have obtained similar results in random regular graphs. Randomized approximation schemes are also available for counting the number of cliques in a random graph [89]. However, prior to this work there were no general results for counting copies of an arbitrary given graph.

In Chapter 3, we present the first general subcase of the subgraph isomorphism counting problem which is almost always efficiently approximable. Previous attempts at solving these kinds of problems have not been very fruitful. For example, even seemingly simple problems like counting cycles in a random graph have remained open for a long time (stated as an open problem in the survey by Frieze and McDiarmid [36]). We present new techniques that can not only handle simple template graphs like cycles and trees, but also major subclasses of more complicated template graph classes like outerplanar, series-parallel, planar, etc.

For achieving this result we introduce a new graph decomposition that we call an *ordered bipartite decomposition*. Informally, the ordered bipartite decomposition is a labeling of the vertices such that every edge is between vertices with different labels, and for every vertex all neighbors with a higher label have identical labels. The algorithm itself is based on a simple randomized sampling idea. Our randomized algorithm will try to embed H into G. If the algorithm succeeds in finding an embedding of H in G, it outputs the inverse probability of finding this embedding.

The results in this chapter are joint work with Martin Fürer [39].

1.1.2 Towards Counting Perfect Matchings in Graphs

Counting perfect matchings in a graph is the most well-studied **#P**-complete problem. Recently, an FPRAS for counting the number of perfect matchings of a bipartite graph has been presented by Jerrum, Sinclair, Vigoda [63]. It is based on the Markov chain Monte-Carlo approach. However, their proof relies crucially on the bipartiteness of the underlying graph (see Lemma 4.2 of [63]). Additionally, due to the high exponent in the currently best running time, i.e., $O(n^7 \log^4 n)$ [16], the algorithm is unlikely to be practical.

In Chapter 4, we present several variants of a simple algorithm from Rasmussen [88] for counting perfect matchings of a graph. One of them has excellent performance on random graphs and another might be a candidate for good worst case performance (still an open problem). The main idea behind both variants is to repeatedly pick one vertex deterministically and to match it with one of its unmatched neighbors picked at random from some distribution. The algorithm then recurses on the remaining subgraph. The algorithm may terminate before completing a matching if at some point the current vertex might have no unmatched neighbors. But if the algorithm succeeds in finding a matching it outputs the inverse probability of finding this matching.

We provide variants that refine this simple sampling idea. For example, one refinement is to increase the frequency of runs as we go down the computation tree (here run stands for a single application of the estimator algorithm over the subgraph). In other words, the algorithm makes multiple recursive calls to itself on smaller instances. This refinement stems from the observation that our estimator makes most of its "mistakes" towards the end when the graph becomes small. This refined algorithm is an FPRAS for counting perfect matchings in almost all graphs. Furthermore, the running time is almost linear in the size of the adjacency matrix.

Results in this chapter were obtained in a joint work with Martin Fürer. Most of the material is from [41], but a few proofs are from [40].

1.2 Distance Approximation in Geometric Intersection Graphs

Intersection graph of a set of geometric objects is the graph with a vertex for each object and an edge between two vertices if and only if the objects have non-empty intersection. A *disk graph* is an intersection graph of a set of disks with arbitrary radii in the plane. See Fig. 1.1. In higher dimensions, these graphs are referred to as *ball graphs*. Disk graphs have been used widely to model wireless ad-hoc networks [70, 75]. An ad hoc mobile network is a network on a set of autonomous mobile nodes with wireless communication. Usually in an ad hoc network, each node has a fixed transmission range so that only the nodes within the range can receive the

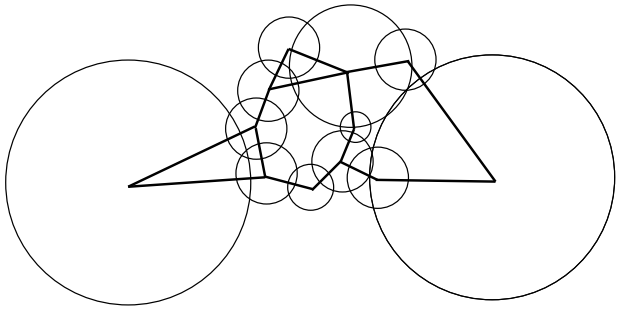


Figure 1.1. Disk Graph: The center of the disks are the vertices in the graphs. An edge is present between two disk centers if the corresponding disks intersect.

messages. Such a range can be modeled by a disk with fixed radius centered at the node [75, 87]. The edge lengths are given by Euclidean distances.

Topology control of wireless ad-hoc networks deployed in the plane is a fundamental problem in the area of wireless computing [87, 92]. Topology control is used to maintain a subnetwork (V, E') of a given network (V, E) and the subnetwork is used for routing and other networking applications. One of the most desirable properties of the resulting topology is the *spanner property* (the shortest path between any pair of nodes in the subnetwork should be longer at most by a constant factor than the shortest path connecting the same pair of nodes in the given network).

Formally, given a real number $t > 1$, we say that a subgraph G' of a graph G is a *t-spanner* for G, if for every pair of vertices u, v in G, there exists a path in G' of length at most t times the distance between u and v in G. The definition of a spanner first appeared implicitly in [11], and then explicitly in [85]. Spanner constructions have also been widely investigated for general graphs [13, 24, 28] (see also the survey by Zwick [111]) and *Euclidean graphs*[3] [8, 19, 29] (see also the book by Narasimhan and Smid [84]).

The problem of computing efficient spanners for ad-hoc wireless networks has been extensively considered [44, 75, 76, 78, 79, 84, 87, 92, 107, 108]. However, all these constructions only work for the restricted case of unit disk graphs (when all the disks have the same radii). In Chapter 5, we consider the problem of efficient construction of sparse spanners of disk and more generally, ball graphs. The *difficulty* in constructing spanners for ball graphs when compared to Euclidean graphs is that two points that are close in space are not necessarily close under the *disk graph metric* (the shortest path metric induced by a connected disk graph on its vertices), In Chapter 5, we present the first algorithm for constructing sparse spanners of ball graphs. It

[3] A Euclidean graph is a graph in which the vertices represent points in the Euclidean space, and the edges are assigned lengths equal to the Euclidean distance between those points.

is based on a new technique for hierarchical decomposition of the space. For a ball graph in $\mathbb{R}^k$ with n vertices, the constructed spanner has only $O(n\epsilon^{-k+1})$ edges. The algorithm runs in $\tilde{O}(n^{2\ell+\delta}\epsilon^{-k})$ expected time[2] where $\ell = 1-1/(\lfloor k/2 \rfloor + 2)$ and δ is any (small) positive constant.

For unit ball graphs, the algorithm has $\tilde{O}(n\epsilon^{-2})$ running time for $k = 2$, $\tilde{O}(n^{4/3}\epsilon^{-3})$ expected running time for $k = 3$, and has $O(n^{2-2/(\lceil k/2 \rceil+1)+\delta}\epsilon^{-k})$ expected running time for $k \geq 4$. The previously best known constructions [75, 78, 79, 84, 87] of spanners of unit ball graphs have time complexity much closer to n^2. Additionally, we show that the construction of $(1+\epsilon)$-spanners for unit ball graphs has the same randomized complexity as the construction of a *Euclidean minimum spanning tree* (problem of computing a minimum spanning tree in the Euclidean space). Therefore, we cannot hope to find a faster algorithm for constructing spanners of unit ball graphs, unless we improve algorithms for some other well-studied problems like Euclidean minimum spanning tree (see Chapter 5 for some additional problems with same complexity as Euclidean minimum spanning tree).

The constructed spanners have a *hereditary*[4] small *vertex separator decomposition*[5]. The results on geometric graph separators might be of independent interest. Since Euclidean graphs are a special case of unit ball graphs, our results also provide a new approach for constructing spanners with small separators in these graphs.

We also consider a related problem of preprocessing a graph so that subsequent distance queries (distance between two vertices) can be answered quickly within a small error. This natural extension to the all pairs shortest path problem captures practical situations, where more often than not, we are interested in estimating the distance between two vertices quickly and accurately [103, 111]. In this framework, the quality of an algorithm is typically measured in terms of the preprocessing time, query time, space complexity and approximation factor. Distance queries are important in disk graphs as they are widely used to determine coverage in wireless sensor networks, and for routing protocols [45, 77, 99]. We present an algorithm that uses the spanners to answer distance queries efficiently. For a disk graph, the algorithm takes $\tilde{O}(n^{3/2})$ pre-processing time and produces a data structure of size $O(n^{3/2})$, such that subsequent distance queries are answered within a multiplicative error of $(1+\epsilon)$ in $O(\sqrt{n})$ time.

Results in this chapter were obtained in a joint work with Martin Fürer. The chapter is primarily based on [43], with some material from [42].

[2]The $\tilde{O}(\cdot)$ notation hides a polylogarithmic factor.

[4]A graph property is hereditary if it is closed under removal of vertices.

[5]A vertex separator is a subset of the vertices that its removal splits the graph into connected components, such that the number of vertices in each component is at most a fixed fraction of the number of vertices in the graph. A separator decomposition is a recursive decomposition of the graph using separators.

1.3 Guide for Reading this Thesis

Chapter 2 is devoted to some basic definitions and general tools. In Chapters 3 and 4, we present the counting results. In Chapter 3, we present the algorithm for approximately counting embeddings and show that it is an FPRAS for a large class of graphs. In Chapter 4, we present an algorithm for approximately counting matchings in general graphs and analyze its performance for the worst and random cases. In Chapter 5, we present algorithms for constructing spanners in ball graphs. We also use spanners to enable fast answering of distance queries. Chapter 5 can be read independently of the previous chapters. In Chapter 6, we make some concluding remarks and discuss directions for further research.

Chapter 2

Preliminaries

2.1 Handy Probabilistic Inequalities

Throughout this thesis we make use of Markov's, Chebyshev's, and Chernoff's inequalities. Here, we present a simplified version of these bounds. Complete proofs of these standard theorems are available in [58, 83].

Theorem 2.1.1 (Markov)**.** *Let X be a random variable assuming only non-negative values. Then for all $t \in \mathbb{R}$,*

$$\Pr[X \geq t] \leq \frac{\mathbb{E}[X]}{t}.$$

Theorem 2.1.2 (Chebyshev)**.** *Let X be a random variable with mean μ and standard deviation σ. Then for all positive t:*

$$\Pr[|X - \mu| > t\sigma] \leq 1/t^2.$$

Theorem 2.1.3 (Chernoff)**.** *Let $X_1, \ldots, X_m$ be independent and identically distributed (i.i.d.) $\{0,1\}$-random variables $\mu = \mathbb{E}[X_i]$ ($1 \leq i \leq n$). For all $\epsilon < 3/2$,*

$$\Pr\left[\left|\sum X_i - \mu n\right| > \epsilon \mu n\right] \leq 2\exp(-\epsilon^2 \mu n/3).$$

2.2 Complexity Classes

We review some basic definitions from complexity theory that are required for our work. For further background, we refer reader to [6, 95].

Definition 1 (Class **NP**)**.** *A language $L \subseteq \{0,1\}^*$ is in **NP** if there exists a polynomial $poly$:*

$\mathbb{N} \to \mathbb{N}$ *and a polynomial-time Turing machine such that for every string* $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists y \in \{0,1\}^{poly(|x|)} \textit{such that} M(x,y) = 1.$$

Definition 2 (**NP**-complete)**.** *We say that a language* $L \subseteq \{0,1\}^*$ *is polynomial-time reducible to* $L' \subseteq \{0,1\}^*$ *(denoted by* $L \leq_p L'$*) if there is a polynomial-time computable function* $f : \{0,1\}^* \to \{0,1\}^*$ *such that for every* $x \in \{0,1\}^*$ *,* $x \in L$ *if and only if* $f(x) \in L'$*. We say that* L' *is* ***NP****-hard if* $L \leq_p L'$ *for every* $L \in$ ***NP****. We say that* L' *is* ***NP****-complete if* L' *is* ***NP****-hard and* L' *is in* ***NP****.*

Definition 3 (Class **#P**, Valiant [105])**.** *A function* $f : \{0,1\}^* \to \mathbb{N}$ *is in the class* ***#P*** *if there exists a polynomial function* $poly : \mathbb{N} \to \mathbb{N}$ *and a polynomial-time Turing machine* M *such that for every string* $x \in \{0,1\}^*$,

$$f(x) = \left| \{ y \in \{0,1\}^{poly(|x|)} \ : \ M(x,y) = 1 \} \right|.$$

Definition 4 (**#P**-complete, Valiant [105])**.** *For a function* $f : \{0,1\}^* \to \{0,1\}^*$*, let* P^f *be the set of functions that are computable by polynomial-time Turing machines that have access to an oracle for* f*. A function* f *is* ***#P****-complete if it is in* ***#P*** *and every* $f \in$ ***#P*** *is in* P^f*.*

Intuitively speaking, a problem Q is complete for a class of problems when Q is in the class, and every other problem in the class can be efficiently expressed as a special case of Q. Therefore, having an efficient algorithm for Q is enough for having an efficient algorithm for every problem in the class.

2.3 Randomized Approximation Schemes

Definition 5 (Fully Polynomial Randomized Approximation Scheme, Karp and Luby [67])**.** *Let* Q *be some function from the set of input strings* Σ^* *to natural numbers. A fully polynomial randomized approximation scheme for* Q *is a randomized algorithm that takes input* $x \in \Sigma^*$ *and an accuracy parameter* $\epsilon \in (0,1)$ *and outputs a number* Z *(a random variable depending on the coin tosses of the algorithm), such that*

$$\Pr\left[(1-\epsilon)Q(x) \leq Z \leq (1+\epsilon)Q(x)\right] \geq 3/4,$$

and runs in time polynomial in $|x|$*,* ϵ^{-1}*.*

Suppose we would like to estimate Q and have a probabilistic algorithm running in time polynomial in $|x|$, whose output is a random variable Z such that $\mathbb{E}[Z] = \mathrm{Q}(x)$ and $\mathbb{E}[Z^2]$ is

finite. Suppose further that we can repeat this experiment as many times as we wish, and the outcomes of the successive trials are independent and identically distributed. A straightforward application of Chebychev's inequality shows that, if we conduct $O\left(\frac{\mathbb{E}[Z^2]}{\mathbb{E}[Z]^2}\epsilon^{-2}\right)$ trials and take the mean, we have an FPRAS for Q.

Theorem 2.3.1. *Let $Z_1, \ldots, Z_t$ be independent and identically distributed with $\mathbb{E}[Z_i] = \mu$ and $Var[Z_i] = \sigma^2$. Let $Z = (1/t)\sum_{i=1}^{t} Z_i$. Then, if $t \geq \frac{4\sigma^2}{\epsilon^2\mu^2}$, we have*

$$\Pr[|Z - \mu| \geq \epsilon\mu] \leq \frac{1}{4}.$$

Proof. By linearity of expectation $\mathbb{E}[Z] = \mu$, and by independence of Z_i's, $Var[Z] = \sigma^2/t$. Applying Chebyshev's inequality (Theorem 2.1.2),

$$\Pr[|Z - \mu| \leq \epsilon\mu] \leq \frac{Var[Z]}{\epsilon^2\mu^2} = \frac{\sigma^2}{t\mu^2\epsilon^2}.$$

Setting $t \geq \frac{4\sigma^2}{\epsilon^2\mu^2}$, gives the required bounds. ❑

Let $\mathcal{A}$ denote the previous FPRAS. The number $3/4$ could be replaced by any number in the open interval $(\frac{1}{2}, 1)$. This is because the success probability can be boosted to $1 - \delta$ by running algorithm $\mathcal{A}$, $O(\log \delta^{-1})$ times and taking the median. The idea is that we can treat each run of $\mathcal{A}$ as a coin flip, where the result is heads if $|Z - \mu| \leq \epsilon\mu$. Let Z' denote the median. If Z' does not lie in the interval $[\mu(1-\epsilon), \mu(1+\epsilon)]$, then at least half of the runs of the algorithm must output a number that lies outside of the interval. The following lemma shows that probability of this happening is small.

Lemma 2.3.2. *If $2s + 1$ independent coin flips are performed with $\Pr[\text{heads}] > 3/4$, then the probability of less than $s + 1$ outcomes being heads is less than $(3/4)^s$.*

Proof. Let E be the event that less than $s + 1$ flips are heads.

$$\begin{aligned}
\Pr[E] &\leq \sum_{i=s+1}^{2s+1} \binom{2s+1}{i} \left(\frac{1}{4}\right)^i \left(\frac{3}{4}\right)^{2s+1-i} \\
&\leq \left(\frac{1}{4}\right)^{s+1} \left(\frac{3}{4}\right)^s \sum_{i=s+1}^{2s+1} \binom{2s+1}{i} \\
&= \frac{1}{2^{2s+2}} \left(\frac{3}{4}\right)^s 2^{2s} \leq \left(\frac{3}{4}\right)^s.
\end{aligned}$$

❑

We can now choose $s = \log_{3/4} \frac{1}{\delta}$ to obtain the desired bound. Together, the complexity of

performing the stochastic experiment, and the ratio of $\frac{\mathbb{E}[Z^2]}{\mathbb{E}[Z]^2}\epsilon^{-2}$ determine the efficiency of the algorithm.

2.4 A Simple Unbiased Counting Algorithm

We would be using the following simple sampling idea (known as importance sampling in statistics [109]): let $\mathcal{S} = \{x_1, \ldots, x_z\}$ be a large set whose cardinality we want to estimate. Assume that we have a randomized algorithm that picks each element x_i with non-zero known probability p_i. Then, the function Count produces an estimate for the cardinality of $\mathcal{S}$. The following proposition shows that the estimate is unbiased, i.e., $\mathbb{E}[Z] = |\mathcal{S}|$.

FUNCTION COUNT

Assume $p_i > 0$ for all i and $\sum_{i=1}^{z} p_i \leq 1$.
If some element x_i is picked (with probability p_i) then output $Z = \frac{1}{p_i}$
Else output $Z = 0$

Proposition 2.4.1. *The function Count is an unbiased estimator for the cardinality of $\mathcal{S}$.*

Proof. It suffices to show that each element x_i has an expected contribution of 1 towards $|\mathcal{S}|$. This holds because on picking x_i (an event that happens with probability p_i), we set Z to the inverse probability of this event happening. Hence, $\mathbb{E}[Z] = \sum_i p_i \cdot \frac{1}{p_i} = |\mathcal{S}|$. ❑

2.5 Background on Random Graphs

The theory of random graphs was initiated by Erdős and Rényi [31]. The most commonly used models of random graphs are $\mathcal{G}(n,p)$ and $\mathcal{G}(n,m)$. Both models specify a distribution on n-vertex graphs. In $\mathcal{G}(n,p)$ each of the $\binom{n}{2}$ edges is added to the graph independently with probability p and $\mathcal{G}(n,m)$ assigns equal probability to all graphs with exactly m edges. See [18] for an extensive treatment. Unless explicitly stated otherwise, the default model addressed in this thesis is $\mathcal{G}(n,p)$.

Let Ω_n be a model of random graphs of order n. In this thesis, $\Omega_n = \mathcal{G}(n,p(n))$ or $\Omega_n = \mathcal{G}(n,m(n))$. We say that almost all graph in Ω_n have a certain property P if $\Pr[P] \to 1$ as $n \to \infty$. Another fact worth noting is that the assertion that almost all graphs $G \in \mathcal{G}(n,1/2)$ have property P is same as the proportion of all labeled graphs of order n that have P tend to 1 as n tends to ∞.

The following theorem will also be used in multiple places in this thesis. It shows that the models $\mathcal{G}(n,p)$ and $\mathcal{G}(n,m)$ are asymptotically equivalent, provided that m is close to pN (where $N = \binom{n}{2}$). The following theorem summarizes this equivalence.

Theorem 2.5.1 (Bollobás [17], Theorem 2, Chapter 2)**.** *If P is any property and $0 < p = m/N < 1$. Let $\Pr_m[Q]$ denote the probability that a graph of $\mathcal{G}(n,m)$ has property Q. Let $\Pr_p[Q]$ denote the probability that a graph of $\mathcal{G}(n,p)$ has property Q. Let $q = 1-p$. Then,*

$$\Pr_m[Q] \leq \Pr_p[Q] \exp(1/(6m))(2\pi pqN)^{1/2} \leq 3m^{1/2}\Pr_p[Q].$$

Proof. The probability $\Pr_p[Q]$,

$$\begin{aligned} \Pr_p[Q] &= \sum_{m'=0}^{N} \binom{N}{m'} p^{m'} q^{N-m'} \Pr_{m'}[Q] \geq \binom{N}{m} p^m q^{N-m} \Pr_m[Q] \\ &\geq \Pr_m[Q] \exp(-1/(6m))(2\pi pqN)^{1/2}. \end{aligned}$$

where the last inequality is based on the following observation:

$$\exp(-1/(6m)) \frac{1}{\sqrt{2\pi}} \left(\frac{n}{m}\right)^m \left(\frac{n}{n-m}\right)^{n-m} \left(\frac{n}{m(n-m)}\right)^{1/2} \leq \binom{n}{m}.$$

Since $(2\pi pqN)^{1/2} \leq 3m^{1/2}$, we obtain the claimed statement. ❑

Chapter 3

Approximately Counting Embeddings into Random Graphs

Let H be a graph, and let $C_H(G)$ be the number of (subgraph isomorphic) copies of H contained in a graph G. In this chapter, we investigate the fundamental problem of estimating $C_H(G)$.

Previous results [21, 34–37, 88, 89] cover only a few specific instances of this general problem, for example, the case when H has degree at most one (monomer-dimer problem). In this chapter, we present the first general subcase of the subgraph isomorphism counting problem which is almost always efficiently approximable. The results rely on a new graph decomposition technique. Informally, an ordered bipartite decomposition is a labeling of vertices such that every edge is between vertices with different labels and for every vertex all neighbors with a higher label have identical labels. The labeling implicitly generates a sequence of bipartite graphs which permits us to break the problem of counting embeddings of large subgraphs into that of counting embeddings of small subgraphs. Using this method, we present a simple randomized algorithm for the counting problem. For all decomposable graphs H and all graphs G, the algorithm is an unbiased estimator (i.e., the expected value equals the true value). Furthermore, for all graphs H having a decomposition where each of the bipartite graphs generated is small and almost all graphs G, the algorithm is a fully polynomial randomized approximation scheme.

We show that the graph classes of H for which we obtain a fully polynomial randomized approximation scheme for almost all G includes graphs of degree at most two, bounded-degree forests, bounded-width grid graphs, subdivision of bounded-degree graphs, and major subclasses of outerplanar graphs, series-parallel graphs, and planar graphs, whereas unbounded-width grid graphs are excluded. As a simple consequence, we also obtain a fully polynomial randomized approximation scheme for counting all cycles in a random graph (solving an open problem of

Frieze and McDiarmid). Additionally, our general technique can easily be applied to proving many more similar results.

We first describe the algorithm and show that it is always an unbiased estimator for $C_H(G)$. Additionally, if the base graph is a random graph from $\mathcal{G}(n,p)$ with constant p and if the template graph has an ordered bipartite decomposition of bounded width, we show the algorithm is an FPRAS. The interesting case of the result is when $p = 1/2$. Since the $\mathcal{G}(n,1/2)$ model assigns a uniform distribution over all graphs on n vertices, an FPRAS (when the base graph is from $\mathcal{G}(n,1/2)$) can be interpreted as an FPRAS for almost all base graphs. This result is quite powerful because now to prove that the number of copies of a template graph can be well-approximated for most graphs G, one just needs to show that the template graph has an ordered bipartite decomposition of bounded width.

The later half of the chapter is devoted to showing many interesting graph classes naturally have an ordered bipartite decomposition of bounded width. Let $\mathcal{C}_k$ denote a cycle of length k. We obtain the following result.

Theorem 3.0.2 (Main Result)**.** *Let H be a simple graph where each connected component is one of the following: graph of degree at most two, bounded-degree tree, bounded-width grid, subdivision of a bounded-degree graph, bounded-degree $\mathcal{C}_3$-free outerplanar graph, bounded-degree $[\mathcal{C}_3, \mathcal{C}_5]$-free series-parallel graph, or planar graphs of girth at least 16. Then, for almost all graphs G, there exists an FPRAS for estimating the number of copies of H in G.*

3.1 Comparision to Previous Results

One of the most well-studied problem is that of counting perfect matchings in graphs. For this problem, Jerrum and Sinclair [60] have presented a simulation of a Markov chain that almost always is an FPRAS (extended to all bipartite graphs in [63]). Similar results using other approaches were obtained later in [21, 34, 41, 88]. Another well-studied problem is that of counting Hamiltonian cycles in random digraphs. For this problem, Frieze and Suen [37] have obtained an FPRAS, and later Rasmussen [88] has presented a simpler FPRAS. Afterwards, Frieze *et al.* [35] have obtained similar results in random regular graphs. Randomized approximation schemes are also available for counting the number of cliques in a random graph [89]. However, there are no general results for counting copies of an arbitrary given graph.

Even when restricted to graphs of degree at most two, Theorem 3.0.2 recovers most of the older results. It also provides simpler, unified proofs for (some of) the results in [21, 34, 37, 88]. For example, to count matchings of cardinality k one could use a template consisting of k disjoint edges. Similarly, to count all cycles of length k the template is a cycle of that length. By

varying k and boosting the success probability, the algorithm can easily be extended to count all matchings or all cycles. This provides the first FPRAS for counting all cycles in a random graph (solving an open problem of Frieze and McDiarmid [36]).

For template graphs coming from the other classes, our result supplies the first efficient randomized approximation scheme for counting copies of them in almost all base graphs. For example, it was not known earlier how to even obtain an FPRAS for counting the number of copies of a given bounded-degree tree in a random graph. For the simpler graph classes the decomposition follows quite straightforwardly, but for graph classes such as subdivision, outerplanar, series-parallel, and planar, constructing the decomposition requires several new combinatorial and algorithmic ideas. Even though our techniques can be extended to other interesting graph classes, we conclude by showing that our techniques can't be used to count the copies of an unbounded-width grid graph in a random graph.

Organization: The rest of the chapter is organized as follows. In Section 3.2, we introduce some terminology. In Section 3.3, we define the ordered bipartite decomposition and present the counting algorithm. In Section 3.4, we show that many graph graph classes have an ordered bipartite decomposition of bounded width. In Section 3.5, we present the negative result for unbounded width grid graphs.

3.2 Definitions and Notation

Automorphisms are edge respecting permutations on the set of vertices. The set of automorphisms form a group under composition. For a graph H, we use $aut(H)$ to denote the size of its automorphism group. For a bounded-degree graph H, $aut(H)$ can be evaluated in polynomial time [81].

Throughout this chapter, we use G to denote a base random graph on n vertices. The graph H is the template whose copies we want to count in G. We can assume without loss of generality that the graph H also contains n vertices, otherwise we just add isolated vertices to H. The number of isomorphic images remains unaffected. Let $\triangle = \triangle(H)$ denote the maximum degree of H.

For a graph F, we use V_F to denote its vertex set and E_F to denote its edge set. Furthermore, we use $v_F = |V_F|$ and $e_F = |E_F|$ for the number of vertices and edges. For a subset S of vertices of F, $N_F(S) = \{v \in V_F - S \mid \exists u \in S \text{ such that } (u, v) \in E_F\}$ denotes the neighborhood of S in F. $F[S]$ denotes the subgraph of F induced by S. Let $L_H(G) = C_H(G) \cdot aut(H)$ denote the number of embeddings (or labeled copies) of H in G. For a random graph G, we will be

interested in quantities $\mathbb{E}[C_H(G)^2]$ and $\mathbb{E}[C_H(G)]^2$.

Our algorithm is randomized. The output of the algorithm is denoted by Z, which is an unbiased estimator of $C_H(G)$, i.e., $C_H(G) = \mathbb{E}_{\mathcal{A}}[Z]$ (expectation over the coin tosses of the algorithm). As the output of our algorithm depends on both the input graph, and the coin tosses of the algorithm, we use expressions such as $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z]]$. Here, the inner expectation is over the coin-tosses of the algorithm, and the outer expectation is over the graphs of $\mathcal{G}(n,p)$. Note that $\mathbb{E}_{\mathcal{A}}[Z]$ is a random variable defined on the set of graphs.

3.3 Approximation Scheme for Counting Copies

We now define a new graph decomposition technique which is used for embedding the template graph into the base graph. As stated earlier our algorithm for embedding works in stages and our notion of decomposition captures this idea.

Ordered bipartite decomposition. An ordered bipartite decomposition of a graph $H = (V_H, E_H)$ is a sequence $V_1, \ldots, V_\ell$ of subsets of V_H such that:

① $V_1, \ldots, V_\ell$ form a partition of V_H.

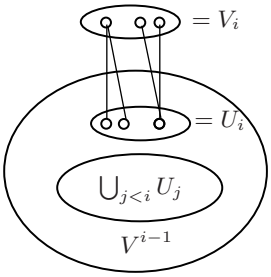
② Each of the V_i (for $i \in [\ell] = \{1, \ldots, \ell\}$) is an independent set in H.

③ $\forall i\, \exists j$ such that $v \in V_i$ implies $N_H(v) \subseteq V_j \cup \left(\bigcup_{k<i} V_k\right)$.

Property ③ just states that if a neighbor of a vertex $v \in V_i$ is in some V_j ($j > i$), then all other neighbors of v which are not in $V_1 \cup \cdots \cup V_{i-1}$, are in V_j. Property ③ will be used in the analysis for random graphs to guarantee that in every stage, the base graph used for embedding is still random with the original edge probability.

Let $V^i = \bigcup_{j \le i} V_j$. Define

$$U_i = N_H(V_i) \cap V^{i-1}.$$



U_i is the set of neighbors of V_i in $V_1 \cup \cdots \cup V_{i-1}$. Define H_i to be the subgraph of H induced by $U_i \cup V_i$. Let E_{H_i} denote the edge set of graph H_i. The *width* of an ordered bipartite decomposition is the size (number of edges) of the largest H_i.

The U_i's will play an important role in our analysis. Note that given a U_j, its corresponding V_j has the property that $V_j \supseteq N_H(U_j) - V^{j-1}$. Hereafter, when the context is clear, we just use *decomposition* to denote an ordered bipartite decomposition. In general, the decomposition of a graph needn't be unique. The following lemma describes some important consequences of the decomposition.

Lemma 3.3.1. *Let $V_1, \ldots, V_\ell$ be a decomposition of a graph $H = (V_H, E_H)$. Then, the following assertions are true. (i) Each of the U_i is an independent set in H (H_i is a bipartite graph). (ii) The edge set E_H is partitioned into $E_{H_1}, \ldots, E_{H_\ell}$.*

Proof. For part (i), assume otherwise. Let (u, v) be an edge in H with both $u, v \in U_i$. Let u appear in some V_j ($j < i$) and v appear in some V_k ($k < i$). Property ② implies that $j \neq k$. Assume without loss of generality that $j < k$. Property ③ implies there exists no vertex $w \in N_H(u)$ such that $w \in V_i$. Therefore, $u \notin U_i$. Contradiction. Additionally, since each of the U_i and V_i is an independent set, each of the graph H_i is bipartite.

For part (ii), first note that due to properties ① and ③, the U_i's are pairwise disjoint (but they do not necessarily form a partition). Therefore, the E_{H_i}'s are also pairwise disjoint. Now since for every edge (u, v) there exist a j, k such that $u \in U_j$ and $v \in V_k$ and without loss of generality $j < k$. Then, $u \in U_k$ and $(u, v) \in E_{H_k}$. Thus, $E_{H_1}, \ldots, E_{H_\ell}$ form a partition of E_H. ❑

Every graph has a trivial decomposition satisfying properties ① and ②, but the situation changes if we add property ③ ($\mathcal{C}_3$ is the simplest graph which has no decomposition). Every bipartite graph though has a simple decomposition, but not necessarily of bounded width. We will primarily be interested in cases where the decomposition is of bounded width. This can only happen if $\triangle$ is a constant. In general, if $\triangle$ grows as a function of n, no decomposition could possibly have a bounded width ($\triangle/2$ is always a trivial lower-bound for the width). For us the parameter ℓ plays no role.

ALGORITHM EMBEDDINGS(G,H)

Initialize $X \leftarrow 1$, $Mark(0) \leftarrow \emptyset$, $\varphi(\emptyset) \leftarrow \emptyset$
let $V_1, \ldots, V_\ell$ denote an ordered bipartite decomposition of H
for $i \leftarrow 1$ to ℓ do
  let $G_f \leftarrow G[V_G - Mark(i-1) \cup \varphi(U_i)]$
  compute X_i (the number of embeddings of H_i in G_f with U_i mapped by φ)
  pick an embedding uniformly at random (if one exists) and use it to update φ
  if no embedding exists, then set Z to 0 and terminate
  $X \leftarrow X \cdot X_i$
  $Mark(i) \leftarrow Mark(i-1) \cup \varphi(V_i)$
$Z \leftarrow X/aut(H)$
output Z

The input to the algorithm EMBEDDINGS is the template graph H together with its decomposition and the base graph G. The algorithm tries to construct a bijection φ between the vertices

of H and G. V_i represents the set of vertices of H which get embedded into G during the i^{th}-stage, and the already constructed mapping of U_i is used to achieve this. For a subset of vertices $S \subseteq V_H$, $\varphi(S)$ denotes the image of S under φ. If $X > 0$, then the function φ represents an embedding of H in G (consequence of property ① and ②), and the output X represents the inverse probability of this event happening. Since every embedding has a positive probability of being found, X is an unbiased estimator for the number of embeddings of H in G (Proposition 2.4.1, Chapter 2), and Z is an unbiased estimator for the number of copies of H in G.

The actual procedure for computing the X_i's is not very relevant for our results, but note that the X_i's can be computed in polynomial time if H has a decomposition of bounded width. In this case the algorithm EMBEDDINGS runs in polynomial time.

3.3.1 FPRAS for Counting in Random Graphs

Since the algorithm EMBEDDINGS is an unbiased estimator by using Chebychev's inequality (Theorem 2.3.1, Chapter 2) implies that repeating the algorithm $O(\epsilon^{-2}\mathbb{E}_{\mathcal{A}}[Z^2]/\mathbb{E}_{\mathcal{A}}[Z]^2)$ times and taking the mean of the outputs results in a randomized approximation scheme for estimating $C_H(G)$. From here on, we abbreviate $C_H(G)$ as C. The ratio $\mathbb{E}_{\mathcal{A}}[Z^2]/\mathbb{E}_{\mathcal{A}}[Z]^2$ is commonly referred to as the *critical ratio*.

We now concentrate on showing that for random graphs the algorithm is an FPRAS. A few of the technical details of our proof are somewhat similar to previous applications of this sampling idea, such as that for counting perfect matchings [41, 88]. The simpler techniques in these previous results, however, are limited to handling one edge per stage (therefore, working only when H is a matching). Our algorithm embeds a small sized subgraph at every stage. The key for obtaining an FPRAS is to guarantee that the factor contributed to the critical ratio at every stage is very small (which is now involved because it is no longer a simple ratio of binomial moments as in [41, 88]). Adding this to the fact that we can do a stage-by-stage analysis of the critical ratio (thanks to the decomposition property which ensures the graph stays essentially random), provides the ingredients for the FPRAS.

The analysis will be done for a worst-case graph H under the assumption that the sizes of the bipartite graphs H_i's are bounded by a universal constant w, and a random graph G. Here, instead of investigating the critical ratio, we investigate the much simpler ratio

$$\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z^2]]/\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z]]^2$$

, which we call the *critical ratio of averages*. We use the second moment method to show that these two ratios are closely related. For this purpose, we take a detour through the $\mathcal{G}(n, m)$

model. The ratio $\mathbb{E}[C^2]/\mathbb{E}[C]^2$ plays an important role here and for bounding it we use a recent result of Riordan [90]. The result (stated below) studies the related question of when a random graph G is likely to have a spanning subgraph isomorphic to H.

Theorem 3.3.2. *(Riordan [90]) Let H be a graph on n vertices. Let $e_H = \alpha N = \alpha(n)N$, and let $p = p(n) \in (0,1)$ with pN an integer. Suppose that the following conditions hold: $\alpha N \geq n$, and $pN, (1-p)\sqrt{n}, np^{\gamma}/\triangle^4 \to \infty$, where*

$$\gamma = \gamma(H) = \max_{3 \leq s \leq n} \{\max\{e_F \mid F \subseteq H, v_F = s\}/(s-2)\}.$$

Then, w.h.p. a random graph $G \in \mathcal{G}(n, pN)$ has a spanning subgraph isomorphic to H.

The quantity γ is closely related to twice the maximum average degree of a subgraph of H. The idea behind the proof is to use Markov's inequality (Theorem 2.1.1, Chapter 2) to bound $\Pr[C = 0]$ in terms of $\mathbb{E}[C]$ and $Var[C]$. The main thrust lies in proving that $\mathbb{E}[C^2]/\mathbb{E}[C]^2 = 1 + o(1)$. Now by following Riordan's proof, we obtain the following result.

Proposition 3.3.3. *Let H be a graph on n vertices. Let $e_H = \alpha N = \alpha(n)N$, and let $p = p(n) \in (0,1)$ with pN an integer. Let $\nu = \max\{2, \gamma\}$. Suppose that the following conditions hold: $pN, np^{\nu}/\triangle^4 \to \infty$ and $\alpha^3 N p^{-2} \to 0$. Then, w.h.p. a random graph $G \in \mathcal{G}(n, pN)$ satisfies $\mathbb{E}[C^2]/\mathbb{E}[C]^2 = 1 + o(1)$. In particular, if H is a bounded-degree graph on n vertices. Then, w.h.p. a random graph $G \in \mathcal{G}(n, \Omega(n^2))$ satisfies $\mathbb{E}[C^2]/\mathbb{E}[C]^2 = 1 + o(1)$.*

Note that some of the conditions in Proposition 3.3.3 are rephrased from Theorem 3.3.2. These are the conditions in the proof of Theorem 3.3.2 that are needed for bounding $\mathbb{E}[C^2]/\mathbb{E}[C]^2$. We will be interested in bounded-degree graphs H. For a bounded-degree graph H, both $\triangle$ and γ are constants. Additionally, we will be interested in dense random graphs (where the conditions of Proposition 3.3.3 are satisfied). We immediately obtain the following corollary.

Corollary 3.3.4. *Let H be a bounded-degree graph on n vertices. Then, w.h.p. a random graph $G \in \mathcal{G}(n, \Omega(n^2))$ satisfies $\mathbb{E}[C^2]/\mathbb{E}[C]^2 = 1 + o(1)$.*

Converting this result back to the $\mathcal{G}(n,p)$ model by using known results for asymptotic equivalence for $\mathcal{G}(n,m)$ and $\mathcal{G}(n,p)$ models (Theorem 2.5.1, Chapter 2) yields

Corollary 3.3.5. *Let H be a bounded-degree graph on n vertices. Let $\omega = \omega(n)$ be any function tending to ∞ as $n \to \infty$, and let p be a constant. Then, w.h.p. a random graph $G \in \mathcal{G}(n,p)$ satisfies $C \geq \mathbb{E}[C]/\omega$.*

Remark: Since C is fairly tightly concentrated around its mean, a rudimentary approximation for C is just $\mathbb{E}[C] = \frac{n!p^{e_H}}{aut(H)}$ (as $v_H = n$). However, this naive approach doesn't produce for *any* $\epsilon > 0$, an $(1 \pm \epsilon)$-approximation for C. If we just output $\mathbb{E}[C]$ it would for any fixed $\epsilon > 0$ produce an ϵ-approximation for random graphs in time independent of ϵ. What we actually want is that for random graphs, the approximation scheme must be able to produce, for any $\epsilon > 0$ an ϵ-approximation in time $poly(n, \epsilon^{-1})$. The order of quantifiers are different. See, e.g., the discussions in [34, 37, 88, 89].

Using the above result we investigate the performance of algorithm EMBEDDINGS when G is a random graph. Here, we don't try to optimize the order of the polynomial arising in the running time analysis, though for simple template instances such as matchings or cycles, one could easily determine the exact order.

Theorem 3.3.6. *Let H be a n-vertex graph with a decomposition of width w (a constant). Let Z be the output of algorithm* EMBEDDINGS*, and let p be a constant. Then, w.h.p. for a random graph $G \in \mathcal{G}(n,p)$ the critical ratio $\mathbb{E}_{\mathcal{A}}[Z^2]/\mathbb{E}_{\mathcal{A}}[Z]^2$ is polynomially bounded.*

Proof. We first relate the critical ratio to the critical ratio of averages. As the estimator is unbiased $\mathbb{E}_{\mathcal{A}}[Z] = C$. Therefore, from Lemma 3.3.5,

$$C = \mathbb{E}_{\mathcal{A}}[Z] = \frac{\mathbb{E}_{\mathcal{A}}[X]}{aut(H)} \geq \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]}{\omega \cdot aut(H)}.$$

Squaring both sides,

$$\mathbb{E}_{\mathcal{A}}[Z]^2 = \frac{\mathbb{E}_{\mathcal{A}}[X]^2}{aut(H)^2} \geq \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2}{\omega^2\, aut(H)^2}.$$

Note that $\mathbb{E}_{\mathcal{A}}[X]/aut(H)$ refers to the expected output for fixed graph G, and the inequalities hold for almost all such graphs G, while $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]/aut(H)$ is the expected output for a random graph $G \in \mathcal{G}(n,p)$.

The numerator of critical ratio of averages, $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z^2]] = \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]/aut(H)^2$. Using Markov's inequality (Theorem 2.1.1, Chapter 2)

$$\Pr[\mathbb{E}_{\mathcal{A}}[Z^2] \geq \omega \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z^2]]] \leq \frac{1}{\omega} \overset{n\to\infty}{\longrightarrow} 0$$

Using the above inequalities yields

$$\frac{\mathbb{E}_{\mathcal{A}}[Z^2]}{\mathbb{E}_{\mathcal{A}}[Z]^2} = \frac{\mathbb{E}_{\mathcal{A}}[X^2]}{\mathbb{E}_{\mathcal{A}}[X]^2} \leq \omega^3 \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[Z]]^2} = \omega^3 \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2}.$$

Now, we just concentrate on bounding the critical ratio of averages. Let $V_1, \ldots, V_\ell$ denote

a decomposition of H of width w. In the bipartite graph $H_i = (U_i, V_i, E_{H_i})$, let $e_i = |E_{H_i}|$, $v_i = |V_i|$, and $u_i = |U_i|$. Let $n_i = n - \sum_{j<i} v_j$. We will rely on the fact that all the H_i's are of bounded width.

Let $n'_i = n_i + u_i$. Let G_i be a random graph from $\mathcal{G}(n'_i, p)$ with u_i distinguished vertices. Let $L_{H_i|U_i}(G_i)$ denote the number of embeddings of H_i in G_i where the mapping of the vertices in U_i to the distinguished vertices in G_i is fixed (given). The results don't depend on the mapping used for U_i. We abbreviate $L_{H_i|U_i}(G_i)$ by L_i.

First we investigate the numerator of the critical ratio of averages. Here we use the fact that

$$\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]] = \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_1^2]] \cdot \cdots \cdot \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_\ell^2]].$$

The previous equality arises, because at the i^{th}-stage the graph used for embedding H_i is from $\mathcal{G}(n'_i, p)$ irrespective of the choices made over the first $(i-1)$-stages. This is guaranteed by property ③ of the decomposition and in turn it allows us to perform a stage-by-stage analysis of the critical ratio.

Furthermore, $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_i^2]] = \mathbb{E}[L_i^2]$ (as the graph is random, it doesn't matter which vertices U_i gets mapped to). Next we investigate the denominator of the critical ratio of averages. Here we use the fact that

$$\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2 = (n! p^{e_H})^2 = \mathbb{E}[L_1]^2 \cdot \cdots \cdot \mathbb{E}[L_\ell]^2.$$

Therefore, the ratio

$$\begin{aligned} \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2} &= \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_1^2]] \cdot \cdots \cdot \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_\ell^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_1]]^2 \cdot \cdots \cdot \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_\ell]]^2} \\ &= \frac{\mathbb{E}[L_1^2] \cdot \cdots \cdot \mathbb{E}[L_\ell^2]}{\mathbb{E}[L_1]^2 \cdot \cdots \cdot \mathbb{E}[L_\ell]^2} = \prod_{i=1}^{\ell} \frac{\mathbb{E}[L_i^2]}{\mathbb{E}[L_i]^2}. \end{aligned}$$

To bound this expression we investigate the parameter $Var[L_i]$.

Now consider a complete bipartite graph K_{u_i,n_i} with one side being the u_i distinguished vertices of G_i and the other side being the remaining (non-distinguished) vertices of G_i. Let $\mathcal{F}_{H_i|U_i}(K_{u_i,n_i})$ be the set of embeddings of H_i in K_{u_i,n_i} where the mapping of the vertices in U_i to the vertices in K_{u_i,n_i} is fixed as in G_i. For each embedding f from $\mathcal{F}_{H_i|U_i}(K_{u_i,n_i})$ define the indicator random variable $I_{f(H_i)} = \mathbf{1}[f(H_i) \subseteq G_i]$. For each $F \subseteq H_i$, let e_F be the number of edges in F, and let r_F be the number of vertices in F which belong to V_i. Now there are $\Theta(n_i^{2v_i - r_F})$ pairs (f, g) of embeddings of H_i in $\mathcal{F}_{H_i|U_i}(K_{u_i,n_i})$ with $f(H_i) \cap g(H_i)$ isomorphic

($\simeq$) to F. In the following, we use $A \asymp B$ for $A = \Theta(B)$.

$$\begin{aligned}
Var[L_i] &= \sum_{f,g} Cov[I_{f(H_i)}, I_{g(H_i)}] = \sum_{\substack{f,g \\ E_{f(H_i)} \cap E_{g(H_i)} \neq \emptyset}} \mathbb{E}[I_{f(H_i)} I_{g(H_i)}] - \mathbb{E}[I_{f(H_i)}]\mathbb{E}[I_{g(H_i)}] \\
&= \sum_{F \subseteq H_i, e_F > 0} \sum_{\substack{f,g \\ f(H_i) \cap g(H_i) \simeq F}} \mathbb{E}[I_{f(H_i)} I_{g(H_i)}] - \mathbb{E}[I_{f(H_i)}]\mathbb{E}[I_{g(H_i)}] \\
&= \sum_{F \subseteq H_i, e_F > 0} \sum_{\substack{f,g \\ f(H_i) \cap g(H_i) \simeq F}} p^{2e_i - e_F} - p^{2e_i} \asymp \sum_{F \subseteq H_i, e_F > 0} n_i^{2v_i - r_F}(p^{2e_i - e_F} - p^{2e_i}) \\
&= \sum_{F \subseteq H_i, e_F > 0} \frac{n_i^{2v_i} p^{2e_i}}{n_i^{r_F} p^{e_F}}(1 - p^{e_F}) \asymp \sum_{F \subseteq H_i, e_F > 0} \frac{\mathbb{E}[L_i]^2}{n_i^{r_F} p^{e_F}}(1 - p^{e_F}) \\
&\asymp \max_{F \subseteq H_i, e_F > 0} \frac{\mathbb{E}[L_i]^2}{n_i^{r_F} p^{e_F}}(1 - p^{e_F}).
\end{aligned}$$

The second equality used the fact that random variables $I_{f(H_i)}$ and $I_{g(H_i)}$ are independent if $E_{f(H_i)} \cap E_{g(H_i)} = \emptyset$. The implicit constants in the above equivalences depend on the width of H_i (a constant), but are independent of n_i. The quantity

$$\max_{F \subseteq H_i, e_F > 0} \frac{(1 - p^{e_F})}{n_i^{r_F} p^{e_F}} = O(1/n_i) \text{ ($r_F = 1$, provides the maximum).}$$

Therefore, $Var[L_i]/\mathbb{E}[L_i]^2 = O(1/n_i)$, implying $\mathbb{E}[L_i^2]/\mathbb{E}[L_i]^2 = 1 + O(1/n_i)$. If $e_i = 0$, then $Var[L_i] = 0$, and $\mathbb{E}[L_i^2] = \mathbb{E}[L_i]^2$. Putting everything together, we obtain

$$\frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2} = \prod_{i=1}^{\ell} \frac{\mathbb{E}[L_i^2]}{\mathbb{E}[L_i]^2} \leq \prod_{i=1}^{\ell} \left(1 + \frac{c}{n_i}\right),$$

for constant c depending only on w and p. Since $n = n_1 > n_2 > \cdots > n_\ell$, $\prod_{i=1}^{\ell}(1 + \frac{c}{n_i})$ can be polynomially bounded (to $O(n^c)$) by a telescoping argument. ❑

Summarizing, we have the following result: if H has a decomposition of bounded width w, then for almost all graphs G, running the algorithm EMBEDDINGS poly$(n)\epsilon^{-2}$ times and taking the mean, results in an $(1 \pm \epsilon)$-approximation for C. Here, poly(n) is a polynomial in n depending on w and p. Since each run of the algorithm also takes polynomial time (as H has bounded width), this is, an FPRAS.

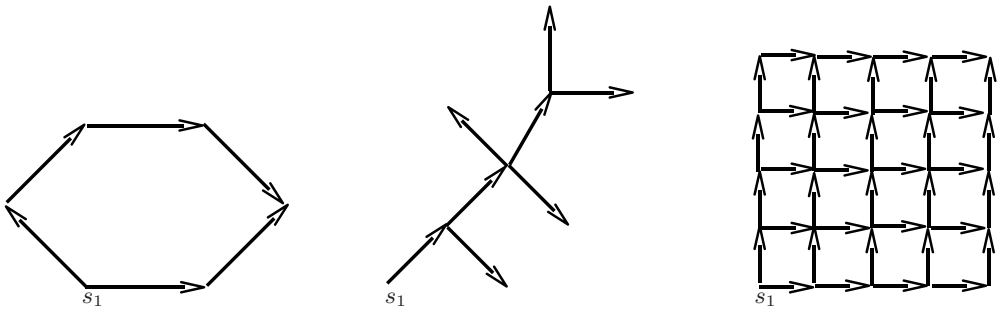


Figure 3.1. Decomposition for a cycle, tree, and grid. The graphs are actually undirected. The arrows just connect the vertices of U_i to their neighbors in V_i. Let $V_1 = \{s_1\}$. All out-neighbors of a vertex are in the same V_i, and all in-neighbors of a vertex are in the same U_i.

3.4 Graphs with Ordered Bipartite Decomposition

We divide this section into subsections based on the increasing complexity of the graph classes. Some of the later graph classes include the ones that will be covered earlier. We will prove the following result in the remainder of this section.

Theorem 3.4.1. *Let H be a graph where each connected component is one of the following: graph of degree at most two, tree, bounded-width grid, subdivision graph, $\mathcal{C}_3$-free outerplanar graph, $[\mathcal{C}_3, \mathcal{C}_5]$-free series-parallel graph, or planar graph of girth at least 16. Then, there exists an ordered bipartite decomposition of H. Furthermore, if H has bounded degree, then the decomposition has bounded width.*

From now onwards, we concentrate on connected components of the graph H. If H is disconnected a decomposition is obtained by concatenating the decomposition of all the connected components (in any order). We will abuse notation and let H stand for both the graph and a connected component in it. $\triangle$ is the maximum degree in H. For constructing the decomposition, the following definitions are useful,

$$U^i = \bigcup_{j \leq i} U_j, \quad V^i = \bigcup_{j \leq i} V_j, \quad \text{and } D^i = V^i - U^i.$$

3.4.1 Some Simple Graph Classes

We first consider simple graph classes such as graphs of degree at most two (paths and cycles), trees, and grid graphs. The Fig. 3.1 illustrates some examples.

Paths: Let H represent a path $(s_1, \ldots, s_{k+1})$ of length $k = k(n)$. Then, the decomposition is, $V_i = \{s_i\}$ for $1 \leq i \leq k+1$.

Cycles: First consider the cycles of length four or greater. Let $s_1, \ldots, s_k$ be the vertices of a cycle H of length $k = k(n)$ enumerated in cyclic order. In the decomposition, $V_1 = \{s_1\}$, $V_2 = \{s_2, s_k\}$, and $V_i = \{s_i\}$ for $3 \leq i \leq k-1$.

Cycles of length three (triangles) don't have a decomposition, but can easily be handled separately. Actually, if $H = H_1 \cup H_2$, where graphs H_1 and H_2 are disjoint, H_1 has a decomposition of bounded width, and H_2 consists of a vertex disjoint union of triangles, then again, under the conditions of Theorem 3.3.6 there exists an FPRAS for estimating C (see Section 3.4.2). This also completes the claim for graphs of degree at most two in Theorem 3.4.1.

Trees: For a tree H, $V_1 = \{s_1\}$, where s_1 is any vertex in H. For $i \geq 2$, let U_i be any vertex from D^{i-1}, then V_i is the set of neighbors of this vertex which are not in V^{i-1}. Intuitively, V_i is the set of children of the vertex in U_i, if one thinks of H as a tree rooted at s_1. The width of the decomposition is at most $\triangle$.

Grids: Let w_0 be the width of the grid graph H. Set $V_1 = \{s_1\}$, where s_1 is any corner vertex in H. Later on, V_i is the set of all vertices which are at a lattice (Manhattan) distance i from s_1. Since for each i, there are at most w_0 vertices at distance i from from s_1, the sizes of the V_i's are bounded if w_0 is bounded. Consequently, the width of the decomposition is bounded if w_0 is bounded. This construction also extends to higher dimensional grid graphs.

3.4.2 Counting Benign Triangles

For simplicity, we will discuss only the case where H is a union of $n/3$ vertex disjoint triangles. Even though H doesn't have a decomposition, there is a simple FPRAS for counting copies of H in random graphs. Extension to other possible cases of template graphs with triangles in them as permissible under the conditions of Theorem 3.4.1 is straightforward. Let $s_1, \ldots, s_n$ be the vertices in H, with every triplet $s_{3i+1}, s_{3i+2}, s_{3i+3}$ forming a triangle in H.

Let $Z = X/aut(H)$ be the output of the algorithm EMBEDDINGS for inputs H and $G \in \mathcal{G}(n, p = \text{constant})$, but where each $V_i = \{s_i\}$ and $\ell = n$ (even though $V_1, \ldots, V_\ell$ is not an ordered bipartite decomposition). As in Theorem 3.3.6, we will again investigate the ratio $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]/\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2$ which equals the critical ratio of averages.

The numerator is,

$$\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]] \;=\; \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_1^2 X_2^2 \cdot \cdots \cdot X_n^2]] = \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_1^2 X_2^2 X_3^2]] \cdot \cdots \cdot \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_{n-2}^2 X_{n-1}^2 X_n^2]].$$

The last equality follows because after embedding each triangle the subgraph of G into which nothing has been embedded yet is random with the original edge probability p. Consider a

representative term from this product,

$$\begin{aligned}\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2_{3i+1}X^2_{3i+2}X^2_{3i+3}]] &= \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2_{3i+1}]]\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2_{3i+2}X^2_{3i+3}]] \\ &= (n-3i)^2\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2_{3i+2}X^2_{3i+3}]].\end{aligned}$$

Here, as earlier, we relied on the fact the graph into which we embed s_{3i+2} is random. Let $m = X_{3i+2}$ and $m' = X_{3i+3}$. Therefore, m denotes the number of ways of embedding s_{3i+2} and m' denotes the number of ways of embedding s_{3i+3}. Since the number of edges incident on the vertices in G is binomially distributed, $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2_{3i+1}X^2_{3i+2}X^2_{3i+3}]]$ equals

$$(n-3i)^2\sum_{m=0}^{n-3i-1} m^2\left(\sum_{m'=0}^{m-1} m'^2\binom{m-1}{m'}p^{m'}(1-p)^{m-1-m'}\right)\binom{n-3i-1}{m}p^m(1-p)^{n-3i-1-m}.$$

Let L_i denote the number of embeddings of a triangle in a random graph from $\mathcal{G}(n-3i,p)$. Then, the denominator is

$$\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2 = \mathbb{E}[L_0]^2 \cdot \cdots \cdot \mathbb{E}[L_{n/3-1}]^2.$$

Note that $\mathbb{E}[L_i] = \binom{n-3i}{3}3!p^3$. Therefore, the critical ratio of averages is

$$\frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2} = \prod_{i=0}^{n/3-1}\frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2_{3i+1}X^2_{3i+2}X^2_{3i+3}]]}{\mathbb{E}[L_i]^2} \leq \prod_{i=0}^{n/3-1}\left(1+\frac{c}{n-3i}\right),$$

for a constant c. Again, we obtain a polynomial bound on the critical ratio of averages, which translates to an FPRAS.

3.4.3 Subdivision Graphs

A *k-subdivision graph* of a graph is obtained by inserting $k = k(n)$ new vertices in every edge, that is by replacing each original edge by a path of length $k+1$. We relax this definition and say that a k-subdivision graph is the graph obtained by inserting at least one and at most k vertices in every edge. Let H be a k-subdivision graph of a graph F. We now show that H has a decomposition of width at most $\triangle$.

The main idea behind the decomposition is that as soon as a vertex v of F appears in some V_j, all vertices in $N_H(v)$ not in V^j are selected in V_{j+1}, i.e., $v \in U_{j+1}$. The decomposition of

H can be formally defined as,

$$V_i = \begin{cases} \{s_1\} \text{ where } s_1 \text{ any vertex in } V_F, \text{ if } i = 1, \\ N_H(a_i) - V^{i-1}, \text{ if } i \geq 2 \text{ and } \{a_i\} = V_F \cap D^{i-1} \neq \emptyset, \\ N_H(b_i) - V^{i-1} \text{ where } b_i \in D^{i-1}, \text{ otherwise.} \end{cases}$$

We now argue correctness of the decomposition for which the following lemma is useful.

Lemma 3.4.2. *There exists at most one vertex in $V_F \cap D^i$ for all stages i in the decomposition.*

Proof. Proof by induction. True by construction for $i = 1$. Now consider some j^{th}-stage. By the inductive hypothesis, $V_F \cap D^{j-1}$ has at most one vertex. If there is a vertex, then a_j is this vertex. In this case, $N_H(a_j)$ doesn't contain any vertex from V_F (subdivision property). Otherwise, $b_j \notin V_F$, therefore, there is at most one vertex of V_F in V_j (subdivision property). Therefore, in both cases, $|V_F \cap D^j| \leq 1$. ❑

It can easily be verified that all properties of the decomposition are satisfied and the width of the decomposition is at most $\triangle$. A direct consequence of this result is that there are some special types of expanders which have a decomposition of bounded width. This is because any graph resulting from the constant subdivision of a constant-degree expander would still be a constant-degree expander. For example, consider the 1-subdivision graph[1] $S(H)$ of constant degree expander H. $S(H)$ has an ordered bipartite decomposition of bounded width. So the only fact that remains to be verified is that vertex expansion ratio of $S(H)$ is a constant.

Lemma 3.4.3. *A 1-subdivision graph of a constant-degree expander is an expander.*

Proof. Let A be a set of vertices in H. Let α (= constant) denote the vertex expansion ratio of H and $\triangle$ denote the maximum degree in H. Let $S(H)$ denote the 1-subdivision graph of H. Let B be a subset of vertices from $N_{S(H)}(A)$. We consider the vertex expansion ratios for two different scenarios of B.

- **Case B** $= \emptyset$. In this case, $|N_{S(H)}(A)| \geq |N_H(U)| \geq \alpha|A|$.
- **Case B** $\neq \emptyset$. First assume that, $B = N_{S(H)}(A)$. Under this assumption, $N_{S(H)}(A \cup B) = N_H(A)$. Say $|N_H(A)| = \lambda$. Now even if $B \subset N_{S(H)}(A)$, $|N_{S(H)}(A \cup B)|$ is at least λ. Therefore,

$$|N_{S(H)}(A \cup B)| = \lambda \geq \alpha|A| \geq \frac{\alpha}{\triangle + 1}(|A| + |B|).$$

[1]The results also extends to any constant (not necessarily 1) subdivision graph of a constant-degree expander.

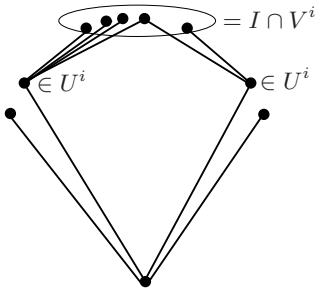
The final case to consider involves a set of vertices C in $S(H)$, which are not in H. In this case, $|N_{S(H)}(C)| \geq |C|/\triangle$.

From the above case analysis it is clear that the vertex expansion ratio of $S(H)$ is a constant, and the proof follows. ❑

Remark: Most of the graph classes we considered appear to have small treewidth. So a natural question would be to relate these two decomposition schemes. However, treewidth and the width of ordered bipartite decomposition are incomparable. In one direction, consider a star graph. Treewidth is 1, but no ordered bipartite decomposition of width less than $n/2$ exists. For the other direction, consider the 1-subdivision graph of Lemma 3.4.3. $S(H)$ has constant expansion which implies a treewidth $\Theta(n)$ [53].

3.4.4 Outerplanar Graphs

A graph$H$ is *planar* if it can be drawn in the plane with its edges intersecting at their vertices only. One such drawing is called an *embedding* of the graph in the plane. A *face* of the graph is a region bounded by a set of edges and vertices in the embedding. A graph is *outerplanar* if it has a planar embedding such that all vertices are on the same face. Let H be a $\mathcal{C}_3$-free outerplanar graph. The idea behind the decomposition is that vertices in U_i partitions the outer face into smaller intervals, each of which can then be handled separately.



Before we formally describe the decomposition, we need some terminology. Let $s_1, \ldots, s_k$ be the vertices around the outer face with $k = k(n)$ (ordering defined by the outerplanar embedding). For symmetry we add two dummy vertices s_0, s_{k+1} without neighbors and define $U_1 = \{s_0, s_{k+1}\}$, and $V_1 = \{s_1\}$.

For $i \geq 1$, two vertices s_{j_0}, s_{j_1} with $j_0 < j_1$, define a stage i interval if $s_{j_0}, s_{j_1} \in U^i$, but for $j_0 < l < j_1, s_l \notin U^i$. If the interval is defined it is the sequence of vertices between s_{j_0}, s_{j_1} (including the endpoints). Let a_i be a median vertex of $I \cap V^i$ (median based on the ordering), where I is a stage i interval. Define U_{i+1} as the smallest subset of V^i containing $\{a_i\}$ and also $N_H(N_H(U_{i+1}) - V^i) \cap V^i$. Define V_{i+1} as $N_H(U_{i+1}) - V^i$. We now argue that this is indeed a decomposition. Consider a stage i interval I, with s_{j_0}, s_{j_1} as the defining end points, and a_i as the median of $I \cap V^i$.

Lemma 3.4.4. *For every $i \geq 1$, there is a stage i interval I with $U_{i+1} \subseteq I$.*

Proof. U_{i+1} can only contain vertices that have a path to a_i not containing any vertex in U^i. Since the graph is outerplanar, any path from a_i to any vertex $w \notin I$ passes through either of the endpoints (s_{j_0}, s_{j_1}), both of which are in U^i. Hence, $U_{i+1} \subseteq I$. ❑

Lemma 3.4.5. *Let I be a stage i interval with $U_{i+1} \subseteq I$. Then, $|U_{i+1}| \leq |I \cap V^i| \leq 2\triangle$.*

Proof. The first inequality follows as $U_{i+1} \subseteq V^i$ (definition) and $U_{i+1} \subseteq I$ (Lemma 3.4.4). For the second one we use induction over i. The interval I is split into several new intervals (at least two as $a_i \in U_{i+1}$) by the vertices of U_{i+1}, which define the stage $i+1$ intervals. The newly created interval are of two types: (a) both its end points are from U_{i+1}, (b) one end point is from U_i and other is from U_{i+1}. In the intervals of the first type there are at most $2\triangle$ vertices from V^{i+1} (at most $\triangle$ from each of the endpoint) and no vertex from V^i. In the intervals of the second type, there are at most $\triangle$ new vertices adjacent to the endpoint in U_{i+1} and at most $\triangle$ old vertices from V^i (from the inductive hypothesis and the fact that a_i is the median). Therefore, each of the newly created stage $i+1$ intervals have at most $2\triangle$ vertices from V^{i+1}. ❑

The properties ① and ③ are guaranteed by the construction. Lemma 3.4.5 implies that the width of the decomposition is most $2\triangle^2$. Property ② holds because there are no triangles in H. See Fig. 3.2 for an example.

3.4.5 Series-Parallel Graphs

A *series-parallel* graph (also called a two-terminal series-parallel graph) is a graph with two distinguished vertices s and t that is obtained as follows. A single edge (s,t) is a series-parallel graph (base case). Let H_a and H_b be two series-parallel graphs with terminals s_a, t_a and s_b, t_b respectively. The graph formed by identifying t_a with s_b is a series-parallel graph with terminals s_a, t_b (series operation is denoted by $\oplus$). The graph formed by identifying s_a with s_b and t_a with t_b is a series-parallel graph with terminals $s_a = s_b$ and $t_a = t_b$ (parallel operation is denoted by $||$).

In the following, the process of adding a vertex to some V_j is referred by the term *selecting*. We say a vertex is *finished* once it is added to some U_i, i.e., all its neighbors are selected. The algorithm is recursive. The basic idea is to first finish the terminals, so that the parallel components separate (for the decomposition purposes). Then, the algorithm finishes a vertex joining two serial components. In both these steps the algorithm might be forced to finish some other vertices too.

To define the decomposition we need some more terminology. Let $H = (V_H, E_H)$ be a $[\mathcal{C}_3, \mathcal{C}_5]$-free series-parallel graph with (distinguished) terminals s, t. Let $\mathcal{V}_H$ denote a decomposition of H with the i^{th}-pair being $V_{i,H}$. Let $V_H^i = \bigcup_{j \leq i} V_{j,H}$. For a set of vertices S in H define

$$D_H(i,S) = \{u \in V_H^{i-1} \mid \text{ there exists } v \in S \text{ such that } (u,v) \in E_H\}.$$

$D_H(i,S)$ represents the set of neighbors of S in H selected in the first $(i-1)$-stages. The

algorithm starts by finishing s, t as follows.

$$V_{1,H} = \{s\}, \quad \text{and} \quad V_{2,H} = N_H(s).$$
$$V_{3,H} = \begin{cases} \{t\} \cup N_H(D_H(3, \{t\})) - V_H^2, \text{ if } t \notin V_H^2, \\ \varnothing, \text{ otherwise.} \end{cases}$$
$$V_{4,H} = N_H(t) \cup N_H(D_H(4, N_H(t))) - V_H^3.$$

In words, the above four stages achieve: (i) select s, (ii) finish s, (iii) select t unless already selected, (iv) finish t. Let $\circ$ denote the concatenation operator. Define

$$\mathcal{V}_H = V_{1,H} \circ V_{2,H} \circ V_{3,H} \circ V_{4,H} \circ \mathcal{V}_{H|s,t},$$

where $\mathcal{V}_{H|s,t}$ is defined recursively as:

1. Base case: If all the vertices in H are selected, $\mathcal{V}_{H|s,t} = \varnothing$.
2. Parallel case: If $H = H_a || H_b$, find recursively $\mathcal{V}_{H_a|s,t}$ and $\mathcal{V}_{H_b|s,t}$. Define $\mathcal{V}_{H|s,t} = \mathcal{V}_{H_a|s,t} \circ \mathcal{V}_{H_b|s,t}$.
3. Serial case: If $H = H_a \oplus H_b$, with x as the vertex joining H_a and H_b. Let $s \in V_{H_a}$ and $t \in V_{H_b}$.
   (a) If x is finished, define $\mathcal{V}_{H|s,t} = \mathcal{V}_{H_a|s,x} \circ \mathcal{V}_{H_b|x,t}$.
   (b) If $x \in V_H^4$ (x has already been selected), but not yet finished, finish x. This produces the set $V_{5,H} = N_H(x) \cup N_H(D_H(5, N_H(x))) - V_H^4$. Define $\mathcal{V}_{H|s,t} = V_{5,H} \circ \mathcal{V}_{H_a|s,x} \circ \mathcal{V}_{H_b|t,x}$.
   (c) Otherwise, first select x which produces the set $V_{5,H} = \{x\} \cup N_H(D_H(5, \{x\})) - V_H^4$. Then, finish x. This produces the set $V_{6,H} = N_H(x) \cup N_H(D_H(6, N_H(x))) - V_H^5$. Define $\mathcal{V}_{H|s,t} = V_{5,H} \circ V_{6,H} \circ \mathcal{V}_{H_a|s,x} \circ \mathcal{V}_{H_b|t,x}$.

The properties ① and ③ are guaranteed by the construction. The following lemma proves that the V_i's form an independent set and provides bounds on the sizes of U_i's. The proof looks at two possible situations, conditioning on presence or absence of paths of length 2 or 3 between s and t. Since both $\mathcal{C}_3$ and $\mathcal{C}_5$ are forbidden, there can either be a path of length 2 or 3 between any two vertices, but not both. This fact will be crucial for implying property ②. See Fig. 3.2 for an example.

Lemma 3.4.6. *Let H be a $[\mathcal{C}_3, \mathcal{C}_5]$-free series-parallel graph with terminals s, t. If there exists no path of length 2 or 3 between s and t, and s, t are the only vertices finished, then the above*

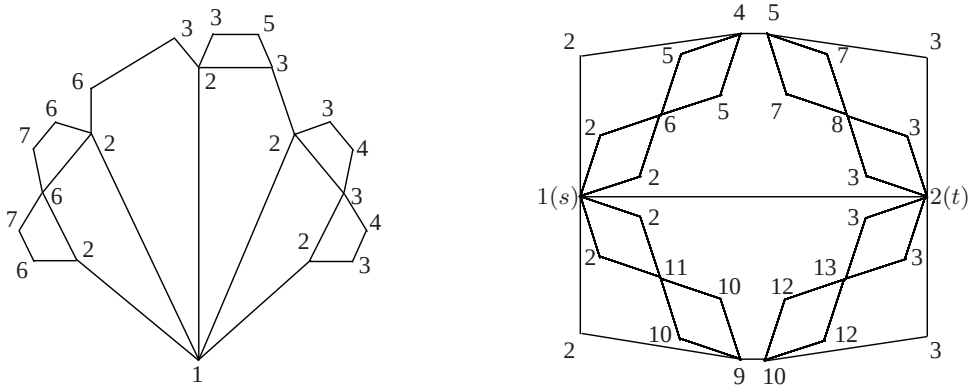


Figure 3.2. The first graph is outerplanar and the second is series-parallel. Vertices with label i constitute V_i. Neighbors of V_i with lower labels constitute U_i.

algorithm finds a decomposition of H by finishing $O(\triangle^2)$ vertices in every stage. Else if there exists a path of length 2 or 3, and s, t and possibly some vertices in $N_H(s) \cup N_H(N_H(s))$ are the only vertices finished, then the above algorithm finds a decomposition of H by finishing $O(\triangle^2)$ vertices in every stage.

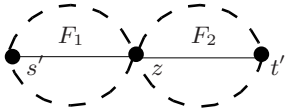
Proof. The proof is inductive. Let F denote a graph with terminals s', t' appearing in some stage of the recursive algorithm. The algorithm always first finishes s' and then t', and once s' and t' are finished the parallel components can be handled independently for constructing the decomposition. In the process of finishing t', the algorithm could possibly finish some vertices in $N_F(s') \cup N_F(N_F(s'))$. Hence, in each of the parallel components F', terminals s', t' and possibly some vertices in $N_{F'}(s') \cup N_{F'}(N_{F'}(s'))$ are finished. Therefore, inductively a decomposition can be obtained. So the challenging case is when F has just one parallel component. Let $F = F_1 \oplus F_2$ with z as the vertex joining F_1 and F_2. There are three different cases. In each of them the interesting event occurs after s', t', and z are finished, which splits F into F_1 and F_2. Afterwards, decomposition on F_1 and F_2 could be constructed independently.

In the following, we describe the cases under the assumption that there exists no edge between s' and t'. If there exists such an edge, then the description would remain the same except that the stage where t' is selected would no longer exist (t' is now selected when s' is finished). Also if there is an edge between s' and t', then there exists no path of length 2 between s' and t', as, otherwise there would be a triangle.

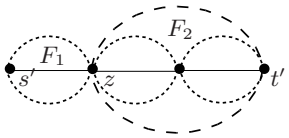
No path of length 2 or 3 between s' and t': Note that at the stage when s' is finished no other vertex in F is finished. Later, when t' is selected the only vertices in $N_F(s')$ that finish at that stage are those which are neighbors of t' (this set is $\varnothing$ as, otherwise, there would be a path of length 2 between s', t'). Similarly, at the stage when t' is finished the only vertices in $N_F(s')$ that finish are those which share a common neighbor with t' (this set is also $\varnothing$ as,

otherwise, there would be a path of length 3 between s', t'). Now at the stage when z is selected some vertices in $N_F(s')$ and $N_F(t')$ could possibly be finished, and at the stage when z is finished some vertices in $N_F(s') \cup N_F(t') \cup N_F(N_F(s')) \cup N_F(N_F(t'))$ could possibly be finished (this supplies the $O(\triangle^2)$ bound). However, as soon as z is finished, the graphs F_1 and F_2 can be handled independently. Now F_1 is a smaller series-parallel graph with terminals s', z, where s', z and possibly some vertices in $N_{F_1}(s') \cup N_{F_1}(N_{F_1}(s'))$ are finished. Therefore, inductively a decomposition of F_1 can be completed. Similarly, F_2 can be viewed as a series-parallel graph with terminals t', z. In F_2, terminals t', z and possibly some vertices in $N_{F_2}(t') \cup N_{F_2}(N_{F_2}(t'))$ are finished. Therefore, inductively a decomposition of F_2 can also be completed.

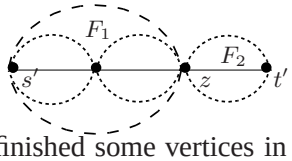
Paths of length 2 *between* s' *and* t': So there is no path of length 3 between s' and t'. If t' has been selected before s' is finished, t' is finished together with s' (at which stage z is also selected). Note that s' and t' can be finished in the same stage because there is no path of length 3. At the stage when z is finished some vertices in $N_F(s') \cup N_F(t')$ could possibly be finished. Afterwards, we can invoke induction on both F_1 and F_2. If s' is finished before selecting t', then z is finished while selecting t'. At the stage when z is finished some vertices in $N_F(s')$ could possibly be finished. Later, at the stage when t' is finished some vertices in $N_F(z)$ could possibly be finished. But again after t' is finished, we can invoke induction on both F_1 and F_2.



Paths of length 3 *between* s' *and* t': So there is no path of length 2 between s' and t'. There are two sub-cases based on the distance from s' to z. First assume that this distance is one. At the stage when s' is finished z is selected. At the stage when t' is selected no vertex in F is finished (absence of path of length 2). At the stage when t' is finished, z is finished and also some other vertices in $N_F(s')$ could possibly be finished. Hereafter, induction can be invoked over F_1 and F_2.



Now if the distance between s' and z is two. Then, the distance between t' and z is one. At the stage when s' is finished no other vertex in F is finished. At the stage when t' is selected no vertex in F is finished. At the stage when t' is finished, z gets selected and some vertices in $N_F(s')$ would be finished. Finally, at the stage when z is finished some vertices in $N_F(s') \cup N_F(N_F(s')) \cup N_F(t')$ could possibly be finished. Hereafter, induction can be invoked over F_1 and F_2.



Therefore, a decomposition of H can be obtained with no more than $O(\triangle^2)$ finishing at each stage, i.e., every $|U_i| = O(\triangle^2)$. A more precise upper bound of $2\triangle^2$ can be obtained by

a more careful analysis. Also as revealed by the proof (unlike triangles) not all pentagons are malevolent for the decomposition. ❑

3.4.6 Planar Graphs

Define a *thread* as an induced path in H whose vertices are all of degree 2 in H. A k-thread is a thread with k vertices. Let H be a planar graph of girth at least 16. We first prove a structural result on planar graphs.

Lemma 3.4.7. *Let H be a planar graph of minimum degree 2 and girth at least 16, then H always contains a 3-thread.*

Proof. Assume without loss of generality that the graph H is connected, otherwise it is sufficient to prove the statement for each of the components. Let $\widehat{H}$ be the graph obtained from H by contracting all degree 2 vertices. Then, $\widehat{H}$ is a planar graph of minimum degree 3.

We first show that $\widehat{H}$ contains a face of degree 5 or less. For contradiction, suppose that all the faces have degree at least 6. Let n be the number of vertices, m be the number of edges, and k be the number of faces of $\widehat{H}$. Moreover, let F be the set of faces and V the set of vertices of $\widehat{H}$. Since the degree of each face is at least 6 (remember that the degree of a face f is the number of edges going around f), $2m = \sum_{f \in F} \deg(f) \geq 6k$. Moreover, $2m = \sum_{v \in V} \deg(v) \geq 3n$, since the minimum degree in $\widehat{H}$ is at least 3. By Euler's formula[2] and the previous inequalities: $m + 2 = n + k \leq (2m)/3 + m/3 = m$. A contradiction.

Let $\hat{f}$ be a face of H that corresponds to a face of the degree 5 or less in $\widehat{H}$. Since the degree of $\hat{f}$ is at least 16 (the girth is 16), it is easy to see that $\hat{f}$ contains a 3-thread in H. ❑

In order to define a decomposition, we define a 3-thread partition $X_1, \ldots, X_c$ of a planar graph H as a partition of V_H such that each X_i satisfies

$$X_i = \begin{cases} \{a_i\}, \text{where } a_i \text{ is a degree 0 or 1 vertex in } H[V_H - \bigcup_{j<i} X_j], \textit{or} \\ \{a_i, b_i, c_i\}, \text{where } a_i, b_i, c_i \text{ form a 3-thread in } H[V_H - \bigcup_{j<i} X_j]. \end{cases}$$

Remember that for a subset of vertices S of H, $H[S]$ denotes the subgraph of H induced by S. By Lemma 3.4.7 every planar graph with girth at least 16 has a 3-thread partition. As earlier, we say, a vertex is selected if we add it to some V_k. Using the 3-thread partition (which can be constructed using Lemma 3.4.7), a decomposition of a planar graph of girth at least 16 can be constructed by repeating this following simple procedure,

[2] Euler's formula states that if a finite, connected, planar graph is drawn in the plane without any edge intersections, and v is the number of vertices, e is the number of edges, and f is the number of faces, then $v - e + f = 2$.

i. Find the largest index l such that X_l contains a vertex z_l which has not yet been selected, but is adjacent to an already selected vertex.

ii. Define $U_i = N_H(z_l) \cap D^{i-1}$ and $V_i = N_H(U_i) - V^{i-1}$.

iii. Increment i.

Lemma 3.4.8. *Let H be a planar graph of girth at least 16. Then, the above procedure finds a decomposition of H of width at most $2\triangle$.*

Proof. The properties ① and ③ of the decomposition are satisfied trivially, whereas the property ② is satisfied because of the girth restriction on H.

Let $X_1, \ldots, X_c$ be a 3-thread partition of H. Let $\bar{H}_i$ be the graph induced by $V_H - \bigcup_{j<i} X_j$ on H. The first observation is that a vertex in any X_j ($1 \leq j \leq c-1$) has at most one edge connecting it to the vertices in $X_{j+1} \cup \cdots \cup X_c$. Consider some stage i of the decomposition. Let l be the largest index with an unselected vertex z_l. From the previous observation it follows that vertices in $N(z_l)$ that are in $X_1 \cup \cdots \cup X_{l-1}$ are not selected, in stages 1 to $i-1$. Assume otherwise. Let u be a vertex belonging to $N(z_l) \cap X_{l'} (l' < l)$ that is selected in the first $i-1$ stages. Then, u needs to have a neighbor in $X_l \cup \cdots \cup X_c - \{z_l\}$, a contradiction since it would imply that u (which is in $X_{l'}$) has two neighbors in $X_l \cup \cdots \cup X_c$. Therefore, till stage i none of the neighbors of z_l in $X_1 \cup \cdots \cup X_{l-1}$ have been selected. By definition of threads z_l could have at most two neighbors in $\bar{H}_i$. The cases where it has two neighbors are one of the following: (a) z_l has one neighbor from $X_{l+1} \cup \cdots \cup X_c$ and another from X_l, or (b) z_l has both its neighbors from X_l. This implies that $|N_H(z_l) \cap D^{i-1}| = |U_i| \leq 2$, and $|V_i| \leq 2\triangle$. ❑

3.5 Negative Result for Ordered Bipartite Decomposition

As mentioned earlier only graphs of bounded degree have a chance of having a decomposition of bounded width. So a natural question to ask is whether all bounded-degree graphs with a decomposition have one of bounded width. In this section, we answer this question negatively by showing that every unbounded-width grid graph fails to satisfy this condition. For simplicity, we will only consider $\sqrt{n} \times \sqrt{n}$ grid graphs, but our proof techniques extend to other cases as well.

Let $H = (V_H, E_H)$ be a $\sqrt{n} \times \sqrt{n}$ grid graph with $V_H = \{(i,j) \,|\, 0 \leq i,j \leq \sqrt{n}-1\}$ and $E_H = \{((i,j),(i',j')) \,|\, i = i' \text{ and } |j-j'| = 1 \text{ or } |i-i'| = 1 \text{ and } j = j'\}$. We now show that any decomposition of H has a width of at least $\sqrt{n}$. Let $V_1, \ldots, V_\ell$ be any decomposition of H. Consider any 2×2 square of H defined by vertices a, b, c, d. The two neighbors a, b of

the vertex c with the smallest label l always have the same label $l' > l$. The fourth vertex d has any label l'' with $l'' \geq l$ and $l'' \neq l'$. We define a new graph $H' = (V_H, E_{H'})$ on the same set of vertices by putting the edge (a, b) into $E_{H'}$. Note that all vertices in a connected component have the same label thus are chosen together (appear in the same V_k).

Let $\mathcal{H}_D$ be a class of graphs on vertex set V_H with exactly one diagonal in every 2×2 square (and no other edges). That is any graph $H_D = (V_H, E_D)$ from $\mathcal{H}_D$ has for every (i, j) with $0 \leq i, j \leq \sqrt{n} - 2$ exactly one of the edges $((i, j), (i + 1, j + 1)), ((i, j + 1), (i + 1, j))$ in E_D and no other edges are in E_D. Note that $H' \in \mathcal{H}_D$. The following theorem shows that any graph $H_D \in \mathcal{H}_D$ has the property that there is a connected component touching top and bottom or left and right.

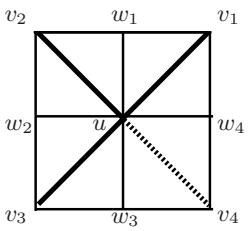
Theorem 3.5.1. *There exists a connected component of H_D that contains at least one vertex from every row or at least one vertex from every column.*

Proof. Assume H_D does not have a connected component that contains a vertex of every row. Let $H_U = (V_U, E_U)$ be the subgraph of H_D generated by all the vertices connected to the top row, i.e., H_U is a collection of those connected components in H_D that have at least one vertex from the top row. By assumption, H_U does not contain any vertices from the bottom row.

For every 2×2 sub-grid of H with vertices a, b, c, d and edge $(a, b) \in E'$, we call (a, b) a *boundary edge* if exactly one of c, d is in V_U and neither of a or b are in V_U. Let $H_B = (V - V_U, E_B)$ be the subgraph of H_D where E_B is the set of boundary edges. We assign the color red to all the vertices in V_U and color black to all the vertices in $V - V_U$. Over the following two claims we make some observations about the structure of H_B. For a vertex v, let $c(v)$ indicate whether the vertex is colored red (r) or black (b).

Claim 3.5.2. *There are no degree 3 vertices in H_B, i.e., all vertices in H_B have degree $0, 1, 2,$ or 4.*

Proof. Assume to the contrary. Let u be a degree 3 black vertex. Let $(u, v_1), (u, v_2), (u, v_3)$ be the only edges incident on u.



Trivially $c(u) = b, c(v_1) = b, c(v_2) = b, c(v_3) = b, c(v_4) = b$. For the other vertices, there are only two possibilities: (i) $c(w_1) = b, c(w_2) = r, c(w_3) = b, c(w_4) = r$ and (ii) $c(w_1) = r, c(w_2) = b, c(w_3) = r, c(w_4) = b$. As all the edges in H_D are all either between

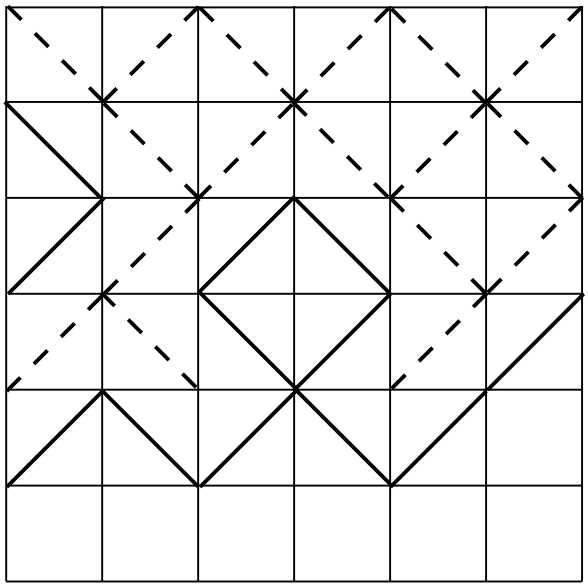


Figure 3.3. The third figure illustrates the negative result. The dotted diagonal lines are the edges in H_U, and the solid diagonal lines are the edges in H_B. There exists a component in H_B than spans from the left to right boundary.

two red vertices or two black vertices and every 2×2 sub-grid has exactly one edge, v_4 is black and there exists an edge between (u, v_4) in H_B. Therefore, every vertex in H_B has degree either $0, 1, 2,$ or 4. ❑

As in the previous claim, by considering all possibilities for the neighbors of u being in V_U or not, one can conclude immediately that all vertices of degree 1 are on the left or right border and there are odd numbers of degree 1 vertices on each border.

Claim 3.5.3. *All the degree 1 vertices of H_B are either on the left or the right border of the grid H. Additionally, there is an odd number of degree 1 vertices of H_B on the left and on the right border.*

Every connected component in H_B has an even number of degree 1 vertices. From Claim 3.5.3, we know that degree 1 vertices only occur at the left and right boundary of H_B and there are odd number of them on both boundaries. Putting these two statements together implies that there exists a component in H_B (therefore, in H_D) that connects the left and the right border. See also Fig. 3.3. ❑

Corollary 3.5.4 (Negative Result)**.** *There exists no decomposition of a $\sqrt{n} \times \sqrt{n}$-grid graph H of width $\sqrt{n} - 1$.*

Chapter 4

Approximately Counting Perfect Matchings in General Graphs

In this chapter, we describe a new method for counting the number of perfect matchings $M(G)$ for any graph $G = (V, E)$ with $|V|$ = even. Counting perfect matchings for the bipartite graph with bipartite adjacency matrix A, is equivalent to computing the permanent of the bipartite adjacency matrix A. A number of approaches have been designed to approximately count the number of perfect matchings for bipartite graphs.

4.1 Related Work on Approximating Permanents

Current research in the area of approximating permanents can been divided into four major categories [22]. They are: elementary recursive algorithms; reductions to determinants; iterative balancing; and Markov chain Monte-Carlo methods. One of the simplest estimators of the permanent using elementary recursive algorithms was proposed by Rasmussen [88]. This estimator has a running time of $O(n^3\omega)$ for almost all $(0\text{-}1)$ matrices. In this chapter, we will extend Rasmussen's idea to get a running time of $O(n^2\omega)$ for almost all $(0\text{-}1)$ matrices. The famous Karmarkar, Karp, Lipton, Lovász, Luby: ($\mathrm{K^2L^3}$) [66] estimator uses reductions to determinants. This estimator, which is based on the Godsil/Gutman estimator [46], has a running time of $poly(n)2^{n/2}$ for all (0-1) matrices. In 1995, Frieze and Jerrum [34] proved that the $\mathrm{K^2L^3}$ estimator runs in time $O(n \cdot M(n)\omega)$ for almost all non-negative matrices, where $M(n)$ is the time required to perform matrix multiplications. Recently, an fpras for computing the permanent of an arbitrary matrix with non-negative entries was proposed by Jerrum, Sinclair, Vigoda [63]. This is based on the Markov chain Monte-Carlo approach. However, due to their high exponent

Authors	Running Time
Jerrum and Sinclair [65]	$O(n^{O(1)}\epsilon^{-2})$
Chien [21]	$O(nM(n)\omega\epsilon^{-2})$
Ours	$O(n^2\omega\epsilon^{-2})$

Table 4.1. Comparision of various estimators for counting perfect matchings in random graphs.

Authors	Running Time
Jerrum and Sinclair [65]	$O(n^{O(1)}\epsilon^{-2})$
K^2L^3 [66], Frieze and Jerrum [34]	$O(n\mathrm{M}(n)\omega\epsilon^{-2})$
Rasmussen [88]	$O(n^3\omega\epsilon^{-2})$
Ours	$O(n^2\omega\epsilon^{-2})$

Table 4.2. Comparision of various estimators for computing permanent of random Boolean matrices.

in the running time, i.e., $O(n^{10}\log^3 n)$ (reduced to $O(n^7\log^4 n)$ in [16]), the algorithm is unlikely to be practical [22]. For this reason, it is still worth investigating alternative approaches. Also, their proof relies crucially on the bipartiteness of the underlying graph and their algorithm doesn't generalize to arbitrary graphs (see Lemma 4.2 of [63]).

4.1.1 Approximately Counting Perfect Matchings in Random Graphs

One of the major contribution of this chapter is to show that there exists a randomized approximation scheme for counting perfect matchings in random graphs (drawn from $\mathcal{G}(n,1/2)$) that takes only $\mathrm{O}(n^2\omega)$ time, where $\omega=\omega(n)$ is any function satisfying $\omega(n)\to\infty$ as $n\to\infty$. We then generalize our technique to obtain a running time which is polynomial in the size of the input graph for the case of $\mathcal{G}(n,p)$. We also show how the same estimator when applied to another problem, that of estimating the size of a tree, could result in better running times for some kinds of random trees. Table 4.1 summarizes the running times of various estimators for counting perfect matchings of a random graph from $\mathcal{G}(n,1/2)$. Table 4.2 summarizes the running times of various estimators for counting permanent of random Boolean matrices (where each entry is 1 independently with probability $1/2$).

Organization: The rest of the chapter is organized as follows. In Section 4.2, we present a new estimator. In Section 4.3, we present a variant of this estimator and show that it has excellent performance on random graphs. In Sections 4.4 and 4.5 we mention some applications of our estimators.

4.2 New Estimators for Counting Perfect Matchings

We start this section with a simple estimator (which is a special case of the algorithm EMBEDDINGS of Chapter 3) and build on it as we go to get better but more complex estimators. The idea behind algorithm SIMPLE is to repeatedly pick one vertex deterministically and match it uniformly at random with one of its neighbors. Remove both the vertex and the matched vertex and all edges incident on them to get $G_{\ell k}$, recurse on the remaining $G_{\ell k}$. This approach may stop early, because at some point the current vertex might have no neighbors.

ALGORITHM SIMPLE

If $n = 0$ then $X_G \leftarrow 1$
Else
  Choose vertex ℓ (lowest numbered vertex remaining)
  $W \leftarrow \{j \,|\, (\ell, j) \in E\}$
  If $W = \varnothing$ then $X_G \leftarrow 0$
  Else
    Choose k uniformly at random from W
    Compute $X_{G_{\ell k}}$ and set $X_G \leftarrow |W| X_{G_{\ell k}}$

An obvious modification to this estimator, which could lead to an improved performance, is to introduce a systematic bias so that the choice of a neighbor is not uniform at random. One could assign different probabilities to the neighbors of ℓ and pick a neighbor k with its probability. Knuth [69] analyzed a variant of such an estimator under some tight conditions on probabilities.

Nodes are finite sequences $(x_1, \ldots, x_k)$ satisfying property $P_k(x_1, \ldots, x_k)$. The root is the empty sequence. If $P_{k+1}(x_1, \ldots, x_k, x_{k+1})$ holds, then also $P_k(x_1, \ldots, x_k)$ holds and $(x_1, \ldots, x_k, x_{k+1})$ is a child of $\vec{x} = (x_1, \ldots, x_k)$. Nodes have arbitrary given costs. We want to estimate the cost of a tree defined as the sum of the costs of its vertices. Knuth's recursive procedure starts at the root. When it is in a leaf $(x_1, \ldots, x_k)$ then it returns with $C = c(x_1, \ldots, x_k)$. When it is in another node $\vec{x} = (x_1, \ldots, x_k)$, it selects a child $(x_1, \ldots, x_k, x_{k+1}(j))$ with probability $\Pr[\vec{x}, j]$ and makes a recursive call to it. When it returns with a cost estimate C it divides it by $\Pr[\vec{x}, j]$, adds $c(\vec{x})$ to it and returns.

Theorem 4.2.1. ***Knuth [69]*** *Let tree*

$$T = \{(x_1, x_2, \ldots, x_n) \,|\, n \geq 0 \text{ and } P_n(x_1, x_2, \ldots, x_n) \text{ holds}\}$$

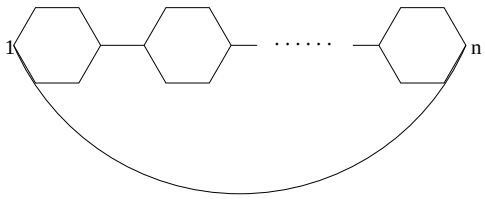


Figure 4.1. Chain of hexagons.

and $cost(T)$ be the function to be estimated. Let $T(x_1, x_2, \ldots, x_k)$ be the subtree rooted at $(x_1, \ldots, x_k)$ and let $(x_1, \ldots, x_k, x_{k+1}(j))$ be the j^{th} child of $(x_1, \ldots, x_k)$. If the probabilities $\Pr[\vec{x}, j]$ (probability of going from k to its j^{th} successor) satisfy

$$\frac{cost(T((x_1, \ldots, x_k, x_{k+1}(j))))}{\Pr[\vec{x}, j]} \leq \alpha \frac{cost(T((x_1, \ldots, x_k, x_{k+1}(i))))}{\Pr[\vec{x}, i]}.$$

for all $i, j, \vec{x}$ and some constant $\alpha \geq 1$, then the variance of the output C computed by Knuth's algorithm is at most

$$\left(\left(\frac{\alpha^2 + 2\alpha + 1}{4\alpha} \right)^n - 1 \right) cost(T)^2.$$

The above theorem is applicable only under very restrictive conditions. We note that we needn't always require that the probabilities be good approximations to the relative subtree costs. The major harm to the variance is done by choosing probabilities too low, higher probabilities tend to have lesser influence on the variance. To illustrate this fact let us consider a variant of the graph example introduced by Jerrum, Sinclair & Vigoda [63] (Fig. 4.1) which has exponentially many perfect matchings. If we use a procedure to eliminate useless edges at every step (can be done in polynomial time with Edmond's algorithm) we see that the perfect matching consisting of all horizontal edges is chosen with probability $\Theta(1)$ (i.e., huge probability) and we still can bound variance by a small constant. Taking this fact into account we modify the above theorem to make it more widely applicable.

Theorem 4.2.2. *Let d be the number of children of $\vec{x} = (x_1, \ldots, x_k)$. If the probabilities $\Pr[\vec{x}, j]$ satisfy*

$$\frac{cost(T((x_1, \ldots, x_k, x_{k+1}(j))))}{cost(T(x_1, \ldots, x_k))} \leq \alpha \Pr[\vec{x}, j].$$

for all j, and some constant $\alpha \geq 1$, then the variance of C as computed by Knuth's algorithm is at most $(\alpha^n - 1)cost(T)^2$

Proof. The idea is similar to that of Knuth. We use T_i to denote $T((x_1, \ldots, x_k, x_{k+1}(i))$ and $\Pr[j]$ to denote $\Pr[\vec{x}, j]$. We note that

$$\sum_{1 \le i < j \le d} \Pr[i] \Pr[j] \left(\frac{cost(T_i)}{\Pr[i]} - \frac{cost(T_j)}{\Pr[j]} \right)^2 = \sum_{1 \le j \le d} \frac{cost(T_j)^2}{\Pr[j]} - \left(\sum_{1 \le j \le d} cost(T_j) \right)^2 .$$

From our assumption on probabilities we get

$$\sum_{1 \le j \le d} \frac{cost(T)^2}{\Pr[j]} \le \alpha \left(\sum_{1 \le j \le d} cost(T_j) \right)^2 .$$

Now let C denote the random variable at node $(x_1, \ldots, x_k)$ and let C_j denote the random variable at $(x_1, \ldots, x_k(j))$. From Knuth we know that the variance of the estimator C is

$$\begin{aligned} Var[C] &= \sum_{1 \le j \le d} \frac{Var[C_j]}{\Pr[j]} + \sum_{1 \le i < j \le d} \Pr[i] \Pr[j] \left(\frac{cost(T_i)}{\Pr[i]} - \frac{cost(T_j)}{\Pr[j]} \right)^2 \\ &\le \sum_{1 \le j \le d} \frac{Var[C_j]}{\Pr[j]} + (\alpha - 1) cost(T)^2. \end{aligned}$$

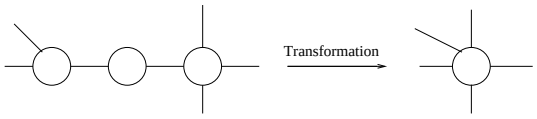
By using induction we complete the proof. ❑

The $cost$ function corresponds to $M(G)$ in our case. One way to assign probabilities is to bias the sampling in favor of low degree during the picking of a neighbor k. We also pick as ℓ, the vertex with lowest degree. We call this estimator GREEDY. The optimal choice of the probabilities is proportional to the unknown number of perfect matchings containing the selected partial matching. In a random bipartite graph, the expected number of such perfect matchings is indirectly proportional to the number of additional neighbors of the matched vertex k. This motivates our choices of probabilities. There is no reason to believe that the above choice of probabilities couldn't be beneficial for general graphs.

ALGORITHM GREEDY

If $n = 0$ then $X_G \leftarrow 1$
Else
  Eliminate edges not contributing towards any perfect matching
  Choose vertex ℓ (a vertex with the lowest degree)
  $W \leftarrow \{j \mid (\ell, j) \in E\}$
  If $W = \varnothing$ then $X_G \leftarrow 0$
  Else
    Choose k from W with probability $p(k) \propto \frac{1}{deg(k)-1}$
    Compute $X_{G_{\ell k}}$ and set $X_G \leftarrow \frac{X_{G_{\ell k}}}{p(k)}$

Another enhancement would be handling vertices with degree 2 in a special way. For vertices with degree 2, one could condense the graph and recurse over the remaining graph. All such algorithms are unbiased estimators. We show that for some general scheme (GEN).



ALGORITHM GEN

1. Use any function to alter G without changing the number of perfect matchings
2. Select any vertex ℓ from the graph deterministically (any criterion)
3. Select a probability distribution on the neighbors of ℓ such that each neighbor k of ℓ has a positive probability $p(k)$
4. Choose a neighbor k of ℓ with probability $p(k)$
5. Remove both the vertices and all edges incident on them to get $G_{\ell k}$
6. Recursively compute the estimator for remaining graph $G_{\ell k}$
7. Output the estimator $X_G = X_{G_{\ell k}}/p(k)$

Theorem 4.2.3. *Let $G = (V, E)$ be a graph, and let X_G be the estimator as given by* GEN. *Then,* $\mathbb{E}[X_G] = M(G)$.

Proof. It is sufficient to show that every fixed perfect matching has an expected contribution of one towards X_G. This implies, the expected value of X_G is the number of perfect matchings. The proof is by induction on the number of recursive calls. The case $n = 0$ is trivial. Each induction step involves two parts:

Part 1: Assume we alter the graph G in Step 1 to get G'. By induction hypothesis we know that every perfect matching has an expected contribution of one towards $X_{G'}$. Since the number of perfect matchings doesn't change from G' to G ($X_G = X_{G'}$), the same contribution is also assigned to G.

Part 2: Let G denote the graph with ℓ as the vertex chosen in Step 2. Assume that $W = \{k \,|\, (\ell, k) \in E\} \neq \emptyset$. For all $k \in W$, we know by induction hypothesis that every perfect matching in $G_{\ell k}$ has an expected contribution of one to $X_{G_{\ell k}}$. Let $\Pr[\ell, k]$ denote the probability that edge (ℓ, k) is chosen. Now, the definition of X_G is

$$\forall k \in W,\ X_G = X_{G_{\ell k}} / \Pr[\ell, k] \text{ with probability } \Pr[\ell, k].$$

Thus, every edge (ℓ, k) contributes a factor of $1/\Pr[\ell, k]$ with probability $\Pr[\ell, k]$ to X_G. Hence, any fixed perfect matching in G which contains the edge (ℓ, k) also has an expected contribution of one to X_G. ❑

However, with all this intelligence, the worst case of GREEDY turned out to be difficult to analyze, so we turned to experimental simulation to test its performance. We investigate the performance of GREEDY against Chien's estimator on some instances of commonly used graphs where counting the number of perfect matchings is interesting and some random graphs. We restricted our test cases to inputs (mostly bipartite) where it is possible to accurately count the number of perfect matchings. Not surprisingly, GREEDY not only runs faster, but also produces more accurate results every time. Random graphs were generated as in DIMACS implementation challenge. The final test case was a complete graph with a perfect matching removed (i.e., deranged matchings of n people with partners (of either sex) other than their spouse [96]). The results (in scientific notation) are summarized in the table below. They are based on 1000 runs of both estimators.

Graph Type	#Matchings	GREEDY	Chien's Estimator
6×6 Square Grid Graph	6.728e+03	6.533e+03	7.310e+03
8×8 Square Grid Graph	1.298e+07	1.270e+07	1.738e+07
10×10 Square Grid Graph	2.586e+11	2.659e+11	1.009e+11
20×3 Rectangular Grid Graph	4.134e+05	4.136e+05	4.222e+05
5D-Hypercube	5.891e+05	5.845e+05	6.659e+05
20+20 Random Bipartite Graph with 100 Edges	6.95e+05	7.191e+05	6.557e+05
20+20 Random Bipartite Graph with 200 Edges	1.871e+12	1.862e+12	1.606e+12
20+20 Random Bipartite Graph with 300 Edges	5.967e+15	5.736e+15	7.693e+15
Complete graph(n=100) with 1 Matching Removed	1.644e+78	1.644e+78	1.530e+78

4.3 A Better Estimator for Random Graphs

A closer look at random graphs tells us that the estimator SIMPLE often makes most of its mistakes towards the end when the graph becomes small. This motivates us to increase the precision as the size of the graph decreases. At every level designated as branching point, we do multiple recursive calls (branching factor) on the subgraph. For simplicity we assume that both branching points and branching factor are pre-computed before the algorithm starts. We call this estimator REP. The resulting computation has a tree structure. We use computation trees with branching points at heights $h = 2^j (j \geq 1)$ and a branching factor of 3 as shown in Fig. 4.2. Also it can easily be verified that such an estimator is unbiased. The idea is similar to one used by Karger and Stein [101] to obtain a faster Min-Cut algorithm.

ALGORITHM REP

If $h = 0$ then $X_G \leftarrow 1$
Else
  Choose vertex ℓ (lowest numbered vertex remaining)
  $W \leftarrow \{j \mid (\ell, j) \in E\}$
  If $W = \emptyset$ then $X_G \leftarrow 0$
  Else
    If h is a branching point then $K \leftarrow$ branching factor
    Else $K \leftarrow 1$
    For $i = 1$ to K do
      Choose $k(i)$ uniformly at random from W
      Compute $X_{G_{\ell k(i)}}$
    $X_G \leftarrow |W|(\frac{1}{K}\sum_{i=1}^{K} X_{G_{\ell k(i)}})$

We now look into the performance of the REP estimator. We bound its worst case performance and analyze its behavior on random graphs. We show that this estimator performs well on random graphs, and improves the previous bound of $O(n\mathcal{M}(n)\omega)$, where $\mathcal{M}(n)$ is time required to perform matrix multiplications for this case and $\omega = \omega(n)$ is any function satisfying $\omega(n) \rightarrow \infty$ as $n \rightarrow \infty$. Even though we conjecture that adding some intelligent vertex selection mechanism as in GREEDY may actually result in better performance of REP in the worst case, it also makes the problem difficult to analyze.

Algorithm	$\mathbb{E}[X_G^2]$	$\mathbb{E}[X_G]^2$	Critical ratio
Chien [21]	$6(2^{n/6}) + (2^{n/2})$	$(2^{n/6}+1)^2$	$\Theta(2^{n/6})$
SIMPLE + Useless Edge Removal	$3 + (3/2)2^{n/3}$	$(2^{n/6}+1)^2$	Constant ($< 3/2$)

Table 4.3. Performance of SIMPLE on Fig. 4.1.

4.3.1 Worst Case Performance of SIMPLE/REP

Let $(n)!!$ (called semi-factorial [56]) denote $n(n-2)(n-4)\ldots 1 = (2m)!/(2^m)m!$ when $n = 2m-1$ is odd.

Theorem 4.3.1. *Let $G = (V, E)$ be a graph with $|V| = n = 2h$, and let X_G be the estimator defined by* REP. *Then, $\mathbb{E}[X_G^2] \leq M(G)(n-1)!!$.*

Proof. Assume that $W = \{j \,|\, (\ell, j) \in E\} \neq \emptyset$. Let $\hat{X}_G(i)$ (for $i = 1, \ldots, K$) be the auxiliary estimator defined by the ith branch, and let $\hat{X}_G = \hat{X}_G(1)$.

$$\begin{aligned}
\mathbb{E}[X_G^2] &= \mathbb{E}[\left(\frac{1}{K}\sum_{i=1}^{K}\hat{X}_G(i)\right)^2] = \frac{1}{K^2}\left(\sum_{i=1}^{K}\mathbb{E}[\hat{X}_G^2(i)] + \sum_{i=1}^{K}\sum_{j=1,j\neq i}^{K}\mathbb{E}[\hat{X}_G(i)]\mathbb{E}[\hat{X}_G(j)]\right) \\
&= \frac{1}{K^2}(K\mathbb{E}[\hat{X}_G^2] + K(K-1)\mathbb{E}[\hat{X}_G]^2) \\
&= \frac{1}{K}\sum_{j\in W}\mathbb{E}[\hat{X}_G^2 | k(1) = j]\Pr[k(1) = j] + (1-1/K)M(G)^2 \\
&\leq \frac{|W|}{K}\sum_{j\in W}\mathbb{E}[(X_{G_{\ell j}})^2] + (1-1/K)M(G)(n-1)!! \\
&\leq \frac{1}{K}\sum_{j\in W}M(G_{\ell j})|W|(n-3)!! + (1-1/K)M(G)(n-1)!! \leq M(G)(n-1)!!.
\end{aligned}$$

where the last step follows because $|W| \leq (2n-1)$ and $\sum_{j\in W} M(G_{\ell j}) = M(G)$. ❑

Thus, the bound on the critical ratio is $\frac{\mathbb{E}[X_G^2]}{\mathbb{E}[X_G]^2} \leq \frac{(n-1)!!}{M(G)}$. The analysis shows a devastatingly bad worst case bound on the performance of the REP (or SIMPLE) estimator, but one might expect that combining these estimators with one or more of the previous ideas could lead to a better critical ratio. To illustrate this possibility, let us consider the performance of the estimators on the graph from Fig. 4.1. Table 4.3 summarizes the result.

4.3.2 Performance of REP on Random Graphs

We use the same notations as in Chapter 3. We use $\mathcal{G}(n)$ to represent the set of all graphs with vertices $\{1, \ldots, n\}$, $\mathcal{G}(n, m)$ to represent the subset of those graphs $\mathcal{G}(n)$ with exactly m edges.

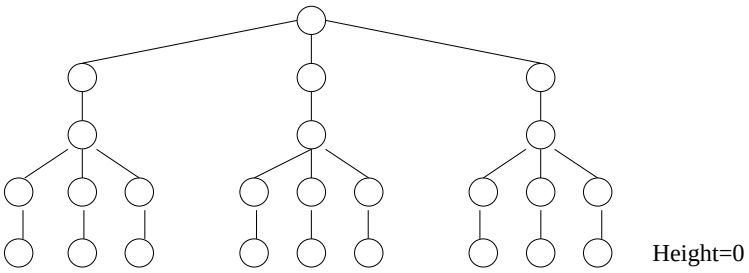


Figure 4.2. Computation tree of the algorithm REP.

We use $\mathbb{E}_{\mathcal{A}}$ to represent the mean over the coin-tosses of the estimator. For a random graph we will also be interested in quantities $\mathbb{E}[M(G)]$ and $\mathbb{E}[M(G)^2]$. As the output (i.e., the random variable X) depends on both, the input graph and the coin tosses of the algorithm, we can use expressions like $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]$. Here, $\mathbb{E}_{\mathcal{A}}[X]$ is a random variable defined on the set of graphs. We investigate the performance of the REP estimator for the most commonly used model $\mathcal{G}(n, 1/2)$.

The idea behind the algorithm REP is the following. As we work with smaller matrices the time decreases drastically. Without increasing the asymptotic complexity, we can branch at powers of s by any number less than s^2. It turns out that for $s = 2$, branching by 3 is actually just sufficient to keep the critical ratio bounded. The situation becomes simple, if we investigate the critical ratio of averages

$$R(X) = \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X]]^2}.$$

For a single run, the critical ratio of averages doubles when the dimension is increased from n to $2n$. Doing K parallel runs decreases the variance of X by a factor of K. Thus, the critical ratio of averages changes from $R(X)$ to $(R(X) - 1)/K + 1$. Therefore, with branching by 3 at powers of 2 the critical ratio of averages grows from ≤ 2 to ≤ 4 and is reduced again to ≤ 2.

Theorem 4.3.2. *The running time of the algorithm* REP *with branching factor of 3 and branching points of* 2^i *for* $(i \geq 1)$ *is* $O(n^2)$.

Proof. Let $2^{i-1} < n \leq 2^i$. Then, the running time: Between top and 1^{st} branching level is $< n^2 \leq 2^{2i}$. Between 1^{st} and 2^{nd} branching level is $< 3(2^{i-1})^2$. Between 2^{nd} and 3^{rd} branching level is $< 9(2^{i-2})^2$. As this forms a geometric series, the total running time is $O(n^2)$. ❑

Our result rests on the following weak version of a result of Janson [57], which we state here.

Theorem 4.3.3. ***(Janson [57], Chien [21])*** *Let* $G \in \mathcal{G}(n, m)$ *where* $\frac{m^2}{n^3} \to \infty$. *Let* $p = m/\binom{n}{2}$,

then

$$\mathbb{E}[M(G)] = (n-1)!!\, p^{n/2} \exp\left(-\frac{1-p}{4p} + O\left((1-p)\frac{n^3}{m^2}\right)\right).$$

and $\frac{\mathbb{E}[M(G)^2]}{\mathbb{E}[M(G)]^2} = 1 + O\left(\frac{n^3}{m^2}\right)$.

Lemma 4.3.4. *Let $\omega = \omega(n)$ be any function satisfying $\omega \to \infty$ as $n \to \infty$. Then, for almost all graphs G, with X_G being any unbiased estimator of $M(G)$,*

$$\frac{\mathbb{E}_{\mathcal{A}}[X_G^2]}{\mathbb{E}_{\mathcal{A}}[X_G]^2} \leq \omega \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G]]^2}$$

Proof. This proof for the case when the graph is bipartite has already been done by Rasmussen [88] and is based on a result from Frieze and Jerrum [34]. The proof for the general graph case proceeds in the same manner. Let M denote a binomial random variable with parameters $\binom{n}{2}$ and $1/2$. Choose a graph G from $\mathcal{G}(n, M)$.

Let $\omega' = \Theta(\ln(\sqrt{\omega}))$. Since $\omega' \leftarrow \infty$ as $n \leftarrow \infty$, using Chernoff (Theorem 2.1.3, Chapter 2) bounds implies:

$$\Pr\left[M < \frac{\binom{n}{2}}{2} - \omega' n\right] \stackrel{n\to\infty}{\longrightarrow} 0.$$

Let $\mu(n, m) = \mathbb{E}_{[M(G)]}$, the mean of the number of matchings in a random graph with m edges as computed in Lemma 4.3.3. From Lemma 4.3.3, we get that,

$$\Pr\left[M(G) < \frac{1}{2}\mu(n, m) \,|\, M = m\right] \stackrel{n\to\infty}{\longrightarrow} 0.$$

We now assume that $M \geq (\binom{n}{2}/2) - \omega' n$ and that $M(G) \geq \mathbb{E}[M(G)]/2$. Let $\mu(n) = \mathbb{E}[M(G)]$, where the expectation is over set of all graphs in $\mathcal{G}(n, 1/2)$. Let $q = (\binom{n}{2}/2) - \omega' n$ and $p = q/\binom{n}{2}$. We get,

$$\begin{aligned}
\frac{M(G)}{\mu(n)} &\geq \frac{\mu(n,m)}{2\mu(n)} \geq \frac{\mu(n,q)}{2\mu(n)} = \frac{1}{2\mu(n)}(n-1)!!p^{n/2}\exp\left(\frac{p-1}{4p} + O\left((1-p)\frac{n^3}{q^2}\right)\right) \\
&\geq \frac{1}{2}\left(1 - \frac{2\omega'}{n-1}\right)^{n/2} \exp\left(-1 + O\left(\frac{1}{n}\right)\right) \\
&= \Theta(\omega^{-1}).
\end{aligned}$$

This implies that $M(G) \geq \mathbb{E}[M(G)]/\omega$, or $\mathbb{E}_{\mathcal{A}}[X_G] \geq \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G]]/\omega$. Squaring both sides we

get $\mathbb{E}_{\mathcal{A}}[X_G]^2 \geq \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G]]^2/\omega^2$. Now by Markov's inequality (Theorem 2.1.1, Chapter 2),

$$\Pr[\mathbb{E}_{\mathcal{A}}[X_G^2] \geq \omega \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G^2]]] \leq \frac{1}{\omega} \stackrel{n\to\infty}{\longrightarrow} 0.$$

Therefore, for almost all graphs G,

$$\frac{\mathbb{E}_{\mathcal{A}}[X_G^2]}{\mathbb{E}_{\mathcal{A}}[X_G]^2} \leq \omega^3 \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_G]]^2}.$$

Since the above inequality holds for all ω, the claimed result follows. ❑

Let $\widehat{\text{REP}}$ be an auxiliary random approximator. Its only difference from REP is that $K = 1$ at the start, i.e., there is no branching in the root of the computation tree even if $n = 2h = 2^j$. The random variables $\hat{X}_h$ and X_h are the outputs of $\widehat{\text{REP}}$ and REP respectively when the input G_{2h} is a random graph from $\mathcal{G}(2h)$. To model the quality of $\widehat{\text{REP}}$ and REP, we introduce two terms $\hat{R}(h)$ and $R(h)$. $\hat{R}(h)$ models the critical ratio of averages of the auxiliary approximator $\widehat{\text{REP}}$, while $R(h)$ models the critical ratio of averages of REP until height h.

$$\hat{R}(h) = \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h^{\,2}]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h]]^2} \quad \text{and} \quad R(h) = \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_h^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_h]]^2}.$$

The proofs are organized as follows: We establish the recursive relationship between $R(h)$ and $\hat{R}(h)$ in Theorems 4.3.5 and 4.3.8. With Theorems 4.3.9 and 4.3.10, we establish the claimed performance bounds. The following theorem shows how $\hat{R}(h)$ varies as a function of $R(2^{\lfloor \log(h-1) \rfloor})$, i.e., R at the previous branching point.

Theorem 4.3.5. *Let G_{2h} denote a random graph from $\mathcal{G}(2h)$, and let $R(h)$ and $\hat{R}(h)$ be the functions defined above. Then,*

$$\hat{R}(h) \leq \begin{cases} 2 & \text{for } h = 1 \\ \dfrac{2h}{2^{\lfloor \log(h-1) \rfloor + 1} + 1} R(2^{\lfloor \log(h-1) \rfloor}) & \text{for } h > 1. \end{cases}$$

Proof. Let M_i denote a binomial variable with parameters i and $p = \frac{1}{2}$. Let M be the degree of the first selected vertex ℓ. Thus, $M = M_{2h-1}$.

$$\begin{aligned} \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h^2]] &= \sum_{m=0}^{2h-1} \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h^2] \mid M = m] \Pr[M = m] \\ &= \sum_{m=0}^{2h-1} \mathbb{E}_{\mathcal{G}}[m^2 \mathbb{E}_{\mathcal{A}}[(X_{h-1})^2]] \Pr[M = m] = \mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[(X_{h-1})^2]] \, \mathbb{E}[M_{2h-1}^2] \end{aligned}$$

$$= \quad \mathbb{E}[M_{2h-1}^2]\mathbb{E}[M_{2h-3}^2]\ldots\,\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[(X_{2^{\lfloor\log(h-1)\rfloor}})^2]].$$

The denominator is: $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h]]^2 = \left(\frac{(2h-1)!!}{2^h}\right)^2 = \prod_{i=1}^{h}\mathbb{E}[M_{2i-1}]^2$.

$$\begin{aligned}\hat{R}(h) &= \frac{\mathbb{E}[M_{2h-1}^2]}{\mathbb{E}[M_{2h-1}]^2}\frac{\mathbb{E}[M_{2h-3}^2]}{\mathbb{E}[M_{2h-3}]^2}\cdots\frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[(X_{2^{\lfloor\log(h-1)\rfloor}})^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_{2^{\lfloor\log(h-1)\rfloor}}]]^2}\\ &\leq \frac{2h}{2^{\lfloor\log(h-1)\rfloor+1}+1}\frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[(X_{2^{\lfloor\log(h-1)\rfloor}})^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_{2^{\lfloor\log(h-1)\rfloor}}]]^2}\\ &= \frac{2h}{2^{\lfloor\log(h-1)\rfloor+1}+1}R(2^{\lfloor\log(h-1)\rfloor}).\end{aligned}$$

❑

Before venturing into showing the dependence of $R(h)$ on $\hat{R}(h)$ we establish a few important technical lemmas. The following lemma shows a bound of a higher moment of a binomial distribution. A lot of similar results have appeared in literature (see [73] for more details).

Lemma 4.3.6. *For $n \geq 0$ we have*

$$\sum_{j=0}^{\binom{n}{2}}\binom{\binom{n}{2}}{j}j^n = O\left(\left(\frac{n^2-n}{4}\right)^n 2^{\binom{n}{2}}\right).$$

Proof. Let N denote $\binom{n}{2}$. The term $\binom{N}{j}j^n$ has its maximum value around $j = N/2 + n$. The idea is to split the summation into three parts, from 0 to $\frac{1}{2}N$, from $\frac{1}{2}N+1$ to $\frac{1}{2}N+n$, and from $\frac{1}{2}N+n+1$ to N.

Each of these three parts can be upper bounded by a constant multiple of $\left(\left(\frac{n^2-n}{4}\right)^n 2^N\right)$. For j small:

$$\sum_{j=0}^{\frac{N}{2}}\binom{N}{j}j^n < \left(\frac{N}{2}\right)^n\sum_{j=0}^{\frac{N}{2}}\binom{N}{j} < \left(\frac{N}{2}\right)^n 2^N.$$

For j moderate:

$$\sum_{j=\frac{N}{2}+1}^{\frac{N}{2}+n}\binom{N}{j}j^n < n\binom{N}{\frac{N}{2}}\left(\frac{N}{2}+n\right)^n.$$

Using Stirling's approximation for $\binom{N}{\frac{N}{2}}$

$$\sum_{j=\frac{N}{2}+1}^{\frac{N}{2}+n}\binom{N}{j}j^n \quad\leq\quad n(\frac{c'}{\sqrt{n^2-n}}2^N)\frac{N^n}{2^n}\left(1+\frac{2n}{N}\right)^n \leq c''\left(\frac{N}{2}\right)^n(2^N).$$

For j large we will show:

$$\sum_{j=\frac{N}{2}+n+1}^{N} \binom{N}{j} j^n \leq c''' \left(\frac{N}{2}\right)^n (2^N).$$

We claim that the terms $\binom{N}{j} j^n$ for $j = \frac{N}{2}+i, \frac{N}{2}+i+n, \frac{N}{2}+i+n, \ldots$ with $i \leq n$ are bounded by a decreasing geometric sequence. We investigate both factors separately and introduce bounds for their corresponding quotients Q_1 and Q_2.

$$\begin{aligned} Q_1 &= \frac{\binom{N}{\frac{N}{2}+i+n}}{\binom{N}{\frac{N}{2}+i}} = \frac{(\frac{N}{2}+i)!}{(\frac{N}{2}+i+n)!}\frac{(\frac{N}{2}-i)!}{(\frac{N}{2}-i-n)!} \\ &= \frac{\prod_{j=1}^{n}(\frac{N}{2}-i-n+j)}{\prod_{j=1}^{n}(\frac{N}{2}+i+j)} = \prod_{j=1}^{n}\frac{\frac{N}{2}-i-n+j}{\frac{N}{2}+i+j} \leq \left(\frac{\frac{N}{2}-i}{\frac{N}{2}+i+n}\right)^n \\ &= \left(1-\frac{2i+n}{\frac{N}{2}+i+n}\right)^n \leq \exp\left(-\frac{(2i+n)n}{\frac{N}{2}+i+n}\right) \leq \exp\left(-2n\frac{2i+n}{N+2i+2n}\right). \end{aligned}$$

For $i \leq n$ and substituting for N, we obtain $Q_1 \leq \exp\left(\frac{-12n^2}{n^2+7n}\right)$. Furthermore, if $n \geq 7$, then $7n \leq n^2$ and thus $Q_1 \leq \exp(-6)$. We now define Q_2 as $Q_2 = \frac{(\frac{N}{2}+i+n)^n}{(\frac{N}{2}+i)^n} = \left(1+\frac{n}{\frac{N}{2}+i}\right)^n \leq \exp\left(\frac{2n}{N+2i}n\right)$. Hence, for $i \geq 0$ and substituting for N, we obtain

$$Q_2 \leq \exp\left(\frac{4n}{n-1}\right) < \exp(5).$$

For $n \geq 7$, we obtain $Q_1 Q_2 < \exp(-11)\exp(5) = \mathrm{e}^{-1}$. As an immediate consequence, we obtain

$$\begin{aligned} \sum_{j=\frac{N}{2}+n}^{N} \binom{N}{j} j^n &< \frac{1}{(1-\mathrm{e}^{-1})} \sum_{j=\frac{N}{2}+n}^{\frac{N}{2}+2n-1} \binom{N}{j} j^n < \frac{n}{(1-\mathrm{e}^{-1})}\frac{2^N}{\sqrt{n^2-n}}\left(\frac{N}{2}+2n\right)^n \\ &< c'''\frac{1}{(1-\mathrm{e}^{-1})}2^N\left(\frac{N}{2}\right)^n\left(1+\frac{4n}{N}\right)^n < c''''\left(\frac{N}{2}\right)^n(2^N). \end{aligned}$$

Substituting these results back we finish the proof of the lemma. ❑

We now prove that the number of perfect matchings in a random graph is fairly tightly clustered.

Lemma 4.3.7. *Let G_n be a random graph from $\mathcal{G}(n)$. Then, for some constant c independent of*

$n \frac{\mathbb{E}[(M(G_n))^2]}{\mathbb{E}[M(G_n)]^2} \leq c$.

Proof. Conditioning the numerator on the number of edges m.

$$\begin{aligned}\mathbb{E}[(M(G_n))^2] &= \Pr\left[m < \frac{1}{4}\binom{n}{2}\right] \mathbb{E}\left[(M(G_n))^2 \,|\, m < \frac{1}{4}\binom{n}{2}\right] + \\ &\quad \Pr\left[m \geq \frac{1}{4}\binom{n}{2}\right] \mathbb{E}\left[(M(G_n))^2 \,|\, m \geq \frac{1}{4}\binom{n}{2}\right].\end{aligned}$$

By Chernoff's bound (Theorem 2.1.3, Chapter 2), we have

$$\Pr\left[m < \frac{1}{4}\binom{n}{2}\right] < \exp\left(-\frac{1}{16}\binom{n}{2}\right).$$

So for the numerator we have

$$\begin{aligned}\mathbb{E}[M(G_n)^2] &< 2\Pr\left[m \geq \frac{1}{4}\binom{n}{2}\right] \mathbb{E}\left[(M(G_n))^2 \,|\, m \geq \frac{1}{4}\binom{n}{2}\right] \\ &\leq 2\,\mathbb{E}\left[(M(G_n))^2 \,|\, m \geq \frac{1}{4}\binom{n}{2}\right] = 2 \sum_{j=\frac{1}{4}\binom{n}{2}}^{\binom{n}{2}} \mathbb{E}[(M(G_n))^2 \,|\, m = j] \Pr[m = j].\end{aligned}$$

Substituting for the probability of having j 1's and using Theorem 4.3.3 for the values of $\mathbb{E}[(M(G_n))^2]$ and $\mathbb{E}[M(G_n)]^2$. Let p_j denote $j/\binom{n}{2} = 2j/(n^2-n)$, we obtain

$$\begin{aligned}&\frac{\mathbb{E}[(M(G_n))^2]}{\mathbb{E}[M(G_n)]^2} \\ &\leq \frac{2^{n+1}((n-1)!!)^2}{((n-1)!!)^2} \sum_{j=\frac{n^2}{8}}^{\binom{n}{2}} \left(\frac{2j}{n^2-n}\right)^n \exp\left(\underbrace{\frac{2(p_j-1)}{4p_j} + O\left(\frac{(1-p_j)n^3}{m^2}\right)}_{\leq c'}\right) \binom{\binom{n}{2}}{j} 2^{-\binom{n}{2}} \\ &< \frac{2^{2n+1}\exp(c')}{(n^2-n)^n} \sum_{j=0}^{\binom{n}{2}} j^n \binom{\binom{n}{2}}{j} 2^{-\binom{n}{2}}.\end{aligned}$$

However, from Lemma 4.3.6, we know that $\sum_{j=0}^{\binom{n}{2}} \binom{\binom{n}{2}}{j} j^n = O\left(\left(\frac{n^2-n}{4}\right)^n 2^{\binom{n}{2}}\right)$. Substituting this result we finish the proof of the lemma. ❑

We are now prepared to establish the dependence of $R(h)$ on $\hat{R}(h)$. As mentioned earlier $R(h)$ and $\hat{R}(h)$ differ only at the branching points.

Theorem 4.3.8. *Let $R(h)$ and $\hat{R}(h)$ be the functions defined above with c an upper bound on*

$\frac{\mathbb{E}_{\mathcal{G}}[M(G)^2]}{\mathbb{E}_{\mathcal{G}}[M(G)]^2}$. *Then,*

$$R(h) \leq \begin{cases} \frac{\hat{R}(h)}{K} + \frac{(K-1)c}{K} & \textit{if } h = \textit{branching point} \\ \hat{R}(h) & \text{otherwise.} \end{cases}.$$

Proof. At all levels other than the branching levels, we have $K = 1$ implying $R(h)$=$\hat{R}(h)$. However, at the branching levels we have:

$$R(h) = \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_h^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_h]]^2} = \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[(\frac{1}{K}\sum_{i=1}^{K}\hat{X}_h(i))^2]]}{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\frac{1}{K}\sum_{i=1}^{K}\hat{X}_h(i)]]^2}.$$

Furthermore, since the outcomes of the successive trials $\hat{X}_h(i)$ are independent and identically distributed

$$\mathbb{E}_{\mathcal{A}}\left[\left(\frac{1}{K}\sum_{i=1}^{K}\hat{X}_h(i)\right)^2\right] = \frac{\mathbb{E}_{\mathcal{A}}[\hat{X}_h^2] + (K-1)\mathbb{E}_{\mathcal{A}}[\hat{X}_h]^2}{K}.$$

Using this for the numerator and noting that the denominator is just $\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[X_h]]^2$ we get

$$R(h) = \frac{\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h^2]]}{K\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h]]^2} + \frac{(K-1)\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h]^2]}{K\mathbb{E}_{\mathcal{G}}[\mathbb{E}_{\mathcal{A}}[\hat{X}_h]]^2} = \frac{\hat{R}(h)}{K} + \frac{(K-1)\mathbb{E}_{\mathcal{G}}[\mathrm{M}(G_h)^2]}{K\mathbb{E}_{\mathcal{G}}[\mathrm{M}(G_h)]^2}.$$

❑

In the following theorem we show that both $R(h)$ and $\hat{R}(h)$ are bound by a constant implying that the critical ratio is $O(\omega)$ from Lemma 4.3.4.

Theorem 4.3.9. *Let $\hat{R}(h)$ and $R(h)$ be the functions defined above. Then, for* REP *with a branching factor of 3 and branching points at powers of 2, there exists a constant $c \geq 1$ such that*

$$\hat{R}(1) \leq 2c,\ \hat{R}(h) \leq \frac{4\,c\,h}{2^{\lfloor \log(h-1) \rfloor + 1} + 1} \textit{ for } h > 1,$$

and

$$R(h) \leq \begin{cases} 2c & h = \textit{branching point or } h = 1 \\ \dfrac{4\,c\,h}{2^{\lfloor \log(h-1) \rfloor + 1} + 1} & \textit{otherwise} \end{cases}$$

Proof. We use induction on h. We know that $c \geq 1$. For $h = 1$, $\hat{R}(1) = 2 \leq 2c$ and $R(1) = 2 \leq 2c$. Assuming the statement is true for h, we prove it for $h + 1$. From Theorem 4.3.5, we

get

$$\hat{R}(h+1) \quad \leq \quad \frac{2h+2}{2^{\lfloor \log h \rfloor + 1}+1} R(2^{\lfloor \log h \rfloor}) \leq \frac{4c(h+1)}{2^{\lfloor \log h \rfloor + 1}+1}.$$

(where $R(2^{\lfloor \log h \rfloor}) \leq 2c$ is by induction hypothesis). For $R(h+1)$, there are two cases:
Case 1: $h+1$ is a branching point. From Theorem 4.3.8, $R(h+1) \leq 2c$.
Case 2: $h+1$ is not a branching point. From Theorem 4.3.8, $R(h+1) = \hat{R}(h+1)$. ❑

Theorem 4.3.10. *Let $\omega = \omega(n)$ be any function satisfying $\omega \to \infty$ as $n \to \infty$. Then, for almost all graphs G, we have,*

$$\frac{\mathbb{E}_{\mathcal{G}}[X_G^2]}{\mathbb{E}_{\mathcal{G}}[X_G]^2} \leq O(\omega).$$

Proof. The factor of $\frac{2h}{2^{\lfloor \log(h-1) \rfloor + 1}+1} \leq 2$. Hence, both $R(h)$ and $\hat{R}(h)$ are $O(1)$ (Theorem 4.3.9). Using Lemma 4.3.4, we bound the critical factor by $O(\omega)$. ❑

Each call to REP with a branching factor of 3 and branching points at powers of 2 can be performed using $O(n^2)$ operations. Furthermore, to obtain an FPRAS for random graphs, it is sufficient to repeat REP $O(\omega)$ times. Thus, we obtain a total running time of $O(n^2\omega\epsilon^{-2})$ for almost all graphs. This is the fastest known algorithm for approximating $M(G)$ for random graphs. Using Theorem 2.3.1 from Chapter 2 we get,

Theorem 4.3.11. *Let $\omega = \omega(n)$ be any function satisfying $\omega \to \infty$ as $n \to \infty$. There exists an $O(n^2\omega\epsilon^{-2})$ time randomized apporximation scheme for counting perfect matchings in almost all graphs. In particular, there exists an $O(n^2\omega\epsilon^{-2})$ time randomized apporximation scheme for computing permanent of almost all $(0\text{-}1)$ matrices.*

4.3.3 Extension of the Estimator to Random Graphs with Arbitrary Probabilities

Till now we have been dealing with the interesting random graph model $\mathcal{G}(n, \frac{1}{2})$. In this section we investigate the performance of the REP for the case $\mathcal{G}(n,p)$ for arbitrary edge probability p. We propose three different choices of the branching point parameter s (Table 4.4) to REP depending on p. We always branch by a factor of 2 at powers of s (branching point) rounded up to the next integer. One can pretend the powers of s are integers because

i. With rounding up, one gets a result that is not worse (compared to rounding down).

ii. The extra cost (of rounding up compared to rounding down) is negligible. It can trivially be bound by a factor of 2.

Probability	B.P. Selector s	Single Run	Critical Ratio	Total Time
p $> \frac{1}{5}$	$\sqrt{2} < s < 2^{\frac{1}{(p^{-1}-1)}}$	$O(n^2)$	$O(\omega(n))$	$O(n^2\omega(n))$
p $= \frac{1}{5}$	$\sqrt{2}$	$O(n^2 \log n)$	$O(\log(n)\omega(n))$	$O(n^2 \log^2(n)\omega(n))$
p $< \frac{1}{5}$	$2^{\frac{1}{(p^{-1}-1)}} < s < \sqrt{2}$	$O(n^{\frac{1}{\log s}})$	$O(n^{\frac{1}{p}-1-\frac{1}{\log s}}\omega(n))$	$O(n^{\frac{1}{p}-1}\omega(n))$

Table 4.4. Performance of REP for different probabilities.

As before, together the complexity of performing the stochastic experiment, and the ratio of $\frac{\mathbb{E}[X^2]}{\mathbb{E}[X]^2}$ will determine the efficiency of the algorithm.

4.3.4 Extension to Probability $p > \frac{1}{5}$

Here, the major contribution to the complexity is from the work we do at the top of the computation tree, handling the first $n/2$ rows costs $O(n^2)$. In the remaining part, the time spent between two consecutive branching levels decrease as geometric series, giving us total running time of $O(n^2)$. By applying Theorem 4.3.10, we obtain that $R(n) = O(1)$.

4.3.5 Extension to Probability $p = \frac{1}{5}$

Here we do $O(n^2)$ work between any two branching points, since we have $O(\log n)$ such branching levels, the complexity of a single run of the experiment is $O(n^2 \log n)$.

Theorem 4.3.12. *Let $h = s^k$ and $h' = s^{k+1}$ be two consecutive branching points. Let 2 be the branching factor. Then,*

$$R(h') = \frac{R(h)s^{(\frac{1}{p}-1)} + c}{2}$$

where c is the constant from Theorem 4.3.5.

Proof. If each entry is chosen to be one with probability p, $\frac{\mathbb{E}[M_h^2]}{\mathbb{E}[M_h]^2} = \left(1 + \frac{1-p}{hp}\right)$. Hence, between h and h' the ratio $\hat{R}$ grows by

$$\begin{aligned}\prod_{j=s^k+1}^{s^{k+1}} \left(1 + \frac{1-p}{jp}\right) &= \exp\left(\ln \prod_{j=s^k+1}^{s^{k+1}} \left(1 + \frac{1-p}{jp}\right)\right) \leq \exp\left(\sum_{j=s^k+1}^{s^{k+1}} \frac{1-p}{jp}\right) \\ &\leq \exp\left(\frac{1-p}{p}\int_{s^k}^{s^{k+1}} \frac{dx}{x}\right) = s^{(\frac{1}{p}-1)}.\end{aligned}$$

As in Theorem 4.3.5, for this probability space $\hat{R}(h') = s^{(\frac{1}{p}-1)}R(h)$. By using these values in Theorem 4.3.8 we complete the proof. ❑

From Theorem 4.3.12, we can see that for $p = 1/5$ the $R(h)$ increases by $\frac{c}{2}$ between two consecutive branching points. Since there are $O(\log n)$ such branching points, $R(n) = O(\log n)$.

4.3.6 Extension to Probability $p < \frac{1}{5}$

Here the major contribution to the complexity of the experiment is from the work we do at the leaves which is of order $O(2^{\frac{\log n}{\log s}})$ =$O(n^{\frac{1}{\log s}})$. Again by application of Theorem 4.3.12, $R(n)$ is $O\left(n^{\frac{1}{p}-1-\log s}\right)$.

4.4 Permanent of Matrices with Arbitrary Entries

Remember that computing permanent of a binary matrix can be interpreted as counting perfect matchings of some bipartite graph (Chapter 1). We can also extend our general scheme GEN to deal with an arbitrary matrix A with non-negative entries. When working with some row r containing some vector v of entries. Choose an entry a_{rj} of A with probability $p_v(j)$ and output $a_{rj}/p_v(j)$, where one reasonable choice of probabilities is $p_v(j) = (\sum_i a_{ri})^{-1} a_{rj}$.

4.5 Estimating the Size of a Tree

One of the chief difficulties involved with the backtracking technique for combinatorial problems has been the inability to predict efficiency of the algorithm. Knuth [69] was the first to present a reasonable estimator for this problem and it was later enhanced by Purdom [86]. Knuth's idea is to estimate the size of a backtrack tree by repeatedly following random paths from the root. We could also apply our method to construct an unbiased estimator for determining the size of backtrack trees.

We conjecture that for certain classes of trees our estimator performs better than Knuth's estimator. One example where we perform better is a random tree model where for every node at depth d we toss $h - d$ coins to generate at most $h - d$ children. This results in height bounded random tree with degree of nodes strictly decreasing as we go down. One could easily see that such a restricted random tree model is essentially what we encounter with permanents, and in previous sections we have shown that our estimator outperforms Knuth's estimator.

Chapter 5

Spanners for Ball Graphs with Applications

Let $G = (V, E)$ be a weighted graph, and let $d_G(u, v)$ be the length of a shortest path between vertices u and v in G.

Definition 6 ($1+\epsilon$-spanner)**.** *For any fixed $\epsilon > 0$, a $(1+\epsilon)$-spanner of G is a subgraph G' such that for every pair of vertices u and v, $d_{G'}(u, v)/d_G(u, v) \leq (1+\epsilon)$.*

In this chapter, we present a new method for producing spanners of geometric graphs with support for fast distance queries. Spanners were formally introduced by Peleg and Schäffer [85].

Spanner constructions have been widely investigated for Euclidean graphs [8–10, 19, 29]. For example, a famous result of Arya *et al.* [8] provides an $O(n \log n)$ time[1] algorithm for constructing a $(1+\epsilon)$-spanner for Euclidean graphs. The spanner constructed has bounded degree and sum of weights of its edges is only a constant times the weight of a minimum spanning tree of the Euclidean graph. For more results on this topic, the readers are referred to the recent book by Narasimhan and Smid [84] on geometric graph spanners.

Our constructions are also more general, as they are not restricted to Euclidean graphs, but extend to geometric disk graphs, as well as their higher dimensional versions, the ball graphs. Edge lengths are given by Euclidean distances, but not all edges have to be present in our graphs.

Disk graphs have been used widely to model the communication between objects in VLSI [82] and to model the communication between objects in the context of wireless networks [75, 87]. For wireless networks they represent the fact that two wireless nodes can directly communicate with each other only if they are within a certain distance. Some restricted versions

[1] Ignoring $1/\epsilon$ factors which grow exponentially with dimension.

of disk graphs like unit disk graphs, quasi unit disk graphs, λ-precision unit disk graphs have also been proposed as good models for wireless networks [55, 72].

Spanners are important for disk graphs because restricting the size of a network reduces the amount of routing information. Spanners are used in topology control for maintaining network connectivity, improving throughput, and optimizing network lifetime [75, 87]. Distance queries are important in disk graphs as they are widely used to determine coverage in wireless sensor networks, and for routing protocols [45, 77, 99]. In most potential applications one would not only desire high accuracy of these estimates but also the actual path producing this estimate. Our approximation algorithms are designed to achieve this guarantee also.

Geometric spanner constructions for disk-like graphs have been widely investigated in both theory and networking communities. Many constructions, both centralized and distributed, also with additional properties like planarity and power saving have been proposed [44, 75, 76, 78, 79, 87, 107, 108]. However, all these constructions only work for some restricted cases of disk graphs (e.g., unit disk graphs).

5.1 Our Contributions

We present the first algorithm for constructing spanners of general ball (disk) graphs. Let S denote the ratio between largest and smallest ball radius in the graph G. For ball graphs in $\mathbb{R}^k$, we construct $(1+\epsilon)$-spanners with $O(n\epsilon^{-k+1})$ edges. For the interesting case when S is polynomially bounded the algorithm runs in $\tilde{O}(n^{2\ell+\delta}\epsilon^{-k})$ expected time[2] where $\ell = 1 - 1/(\lfloor k/2 \rfloor + 2)$ and δ is any positive constant. In general, there is a logarithmic dependence on S in the running time.

In the case when all the balls have the same radius (unit ball graphs), the algorithm has $\tilde{O}(n\epsilon^{-2})$ running time for $k = 2$, has $\tilde{O}(n^{4/3}\epsilon^{-3})$ expected running time for $k = 3$, and has $O(n^{2-2/(\lceil k/2 \rceil+1)+\delta}\epsilon^{-k})$ expected running time for $k \geq 4$. The previously best known constructions of spanners of unit ball graphs were primarily based on Yao graph construction [110] and have $O(n^{2-a(k)})$ running time for $a(k) = 2^{-k+1}$ in dimensions k greater than 3. Additionally, for unit ball graphs, we show that constructing $(1+\epsilon)$-spanners has the same randomized complexity as the construction of a Euclidean minimum spanning tree. Therefore, we cannot hope to find a faster algorithm for constructing spanners of unit ball graphs, unless we improve algorithms for some other well-studied problems [32].

Our spanners also have a small vertex separator decomposition (defined in Section 5.3). Note that the input graph might have no small separator, it could even be complete. The spanners

[2]We use the notation $\tilde{O}(f) \equiv O(f\text{polylog}(f))$ (i.e., $\tilde{O}$ ignores logarithmic factors, not merely constants).

constructed have a vertex separator of size $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S)$, which can be found in $O(n \log n)$ time. Since Euclidean graphs are just a special case of (unit) ball graphs, our results also provide a new approach for constructing spanners with small separators in these graphs (a previous solution [47] turned out to be incorrect [48]). Our result can be seen as an extension of a result by Smith and Wormald [98], who show the existence of separators of size $O(n^{1-1/k}\epsilon^{-O(1)})$ in the Arya *et al.* [8] spanners of the Euclidean graphs. Separators for disk graphs with bounded S were also investigated in [3, 38].

Using this, we obtain fast algorithms for approximately answering distance queries in ball graphs.

Definition 7 (c-stretch)**.** *An estimate $\delta(u, v)$ of the path length $d_G(u, v)$ is said to be a c-stretch if it satisfies $d_G(u, v) \leq \delta(u, v) \leq cd_G(u, v)$.*

We show that the spanner can be preprocessed in $\tilde{O}(nf(n, S, \epsilon))$ time and space, such that subsequent distance queries under the d_G metric can be answered with $(1 + \epsilon)$-stretch in $O(f(n, S, \epsilon))$ time, where $f(n, S, \epsilon) = n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S$ is the size of the separator.

At the expense of a slightly higher preprocessing time we also obtain a better error guarantee on the queries. This better error guarantee is formalized by the notion of *strong* $(1 + \epsilon)$*-approximation*. A $(1 + \epsilon)$-approximate estimate $\delta(u, v)$ is said to be strong if

$$d_G(u, v) \leq \delta(u, v) \leq d_G(u, v) + \epsilon\tau(u, v), \text{ where}$$

$$\tau(u, v) = max\{\ell \mid \exists \text{ a shortest path in } G \text{ between } u \text{ and } v \text{ with maximal edge length } \ell\}.$$

Let G be a ball graph with $m = \Omega(n \log n)$ edges. We show that after $O(mf(n, S, \epsilon))$ time and $O(nf(n, S, \epsilon))$ space preprocessing, a strong $(1 + \epsilon)$-approximate estimate for the distance between any two vertices can be obtained in $O(f(n, S, \epsilon))$ time. In all our cases, $\tau(u, v)$ is strictly less than $d_G(u, v)$. Because, if edge between u and v is present then $d_G(u, v) = d(u, v)$ and we can set $\tau(u, v) = 0$ and $\delta(u, v) = d(u, v)$. If no edge exists between u and v then $\tau(u, v) < d_G(u, v)$. Therefore, this approximation is strictly better than the standard $(1 + \epsilon)$-approximation. In both the above results, we can also output a corresponding short path between the query vertices in $O(L)$ time, where L is the number of edges of the reported path.

5.2 Related Work on Distance Queries

For general undirected graphs, Thorup and Zwick [103] show that for any $c \geq 1$, a graph with n vertices and m edges can be preprocessed in $O(cmn^{1/c})$ expected time, constructing a data

structure of size $O(cn^{1+1/c})$, such that a $(2c-1)$-stretch answer to any distance query can be produced in $O(c)$ time. Many other time-space trade-off results are also known (See Zwick's [111] survey on this subject). Better results are available for some special classes of graphs. For example, if the graph G is a geometric $O(1)$-spanner with m edges, then Gudmundsson *et al.* [49, 50] show that G can be preprocessed in $O(m \log n)$ time, constructing a data structure of size $O(n \log n)$, such that a $(1+\epsilon)$-stretch answer to any distance query can be produced in $O(1)$ time. For unit disk graphs, Gao and Zhang [45] gave a construction that produces a data structure of size $O(n \log n)$ in $O(n\sqrt{n \log n}\epsilon^{-3})$ time, such that $(1+\epsilon)$-stretch answer to any distance query can be produced in $O(1)$ time.

Organization: The rest of the chapter is organized as follows. In Section 5.3, we introduce some terminology. In Section 5.4, we describe the basic ideas behind our construction. In Section 5.5, we present the spanner construction algorithm. In Section 5.6, we present and analyze the algorithm for finding a separator decomposition of the spanner. In Section 5.7, we present two algorithms for fast answering of distance queries.

5.3 Preliminaries

Let $\mathcal{P} = \{v_1, \ldots, v_n\}$ be a set of points in $\mathbb{R}^k$ for any fixed dimension k. Let $\mathcal{D} = \{D_{v_1}, \ldots, D_{v_n}\}$ be a set of n balls such that (1) D_u is centered at $u \in \mathcal{P}$, and (2) D_u has radius of r_u. Balls D_u and D_v intersect if $d(u,v) \leq r_u + r_v$, where $d(\cdot,\cdot)$ denotes the Euclidean metric. The ball graph $G = (V = \mathcal{P}, E)$ is a weighted graph where an edge between u and v with weight $d(u,v)$ exists iff D_u and D_v intersect. Let d_G denote the shortest path metric induced by the connected graph G on its vertices.

We use m to denote the number of edges in graph G. We require that the input to the algorithms is the set of balls, not only the corresponding intersection graph. Without loss of generality we assume that for every ball $D_u \in \mathcal{D}$ there exists at least one ball center outside D_u. We then re-scale the balls such that the largest radius equals one. The global scale factor (ratio between largest and smallest radius) of $\mathcal{D}$ is then defined as

$$S(\mathcal{D}) = 1/\min\{r_u \mid D_u \in \mathcal{D}\}.$$

In the plane, our algorithms use a variant of quadtrees. For a node t, denote by $p(t)$ the parent of t in the tree. We use d_t to denote the level (depth) of node t in the tree. A point (x,y) is *contained* in the node t representing a square with center (x_t, y_t) and length l_t in the quadtree

iff

$$x_t - l_t/2 \leq x < x_t + l_t/2 \quad \text{and} \quad y_t - l_t/2 \leq y < y_t + l_t/2.$$

For a set of squares T in the quadtree a point is contained in T iff there exists $t \in T$, such that point is contained in t. The distance between two squares is the Euclidean distance between their centers. In higher dimensions, the algorithm uses a variant of 2^k-trees.

Throughout the chapter we refer to the vertices of a graph as *vertices* and vertices of a tree as *nodes*. We assume without loss of generality that ϵ^{-1} is a power of 2 to avoid many floors and ceilings. Note that for a given ϵ it is sufficient to construct a $(1 + c\epsilon)$-spanner for a fixed constant c as we could always start be scaling down ϵ by a factor of c. All logarithms $(\log)$ in this chapter are base 2.

A subset of vertices T of a graph G with n vertices is an $f(n)$-separator that α-splits ($\alpha < 1$) if $|T| \leq f(n)$ and the vertices of $G \setminus T$ can be partitioned into two sets V_1 and V_2 such that there are no edges from V_1 to V_2, $\max\{|V_1|, |V_2|\} \leq \alpha n$, where f is a function. An $f(s)$-separator decomposition of G is a recursive decomposition of G using separators, where subgraphs of size s have separators of size $O(f(s))$.

5.4 Yao Graphs and Well-Separated Pair Decompositions

In this section, we present an overview of some of the basic concepts that we use to obtain the spanner results. Both the constructions of Yao graphs and well-separated pair decompositions have been widely used to construct geometric spanners (see for example [84]).

5.4.1 Modified Yao Graph

A Yao graph [110] construction involves partitioning the space around each point into cones with a fixed opening angle and connecting the point to its nearest neighbor in each cone. Even though constructing the original Yao graph is costly (see [110]), a variant of it called the Θ-graph can be constructed in $\tilde{O}(n\epsilon^{-k})$ time [91]. In a Θ-graph points inside a wedge are projected onto the angle bisector of a cone and the closest point in this distorted metric is used to add an edge. Even though Θ-graphs are spanners of the Euclidean graphs, they fail to be spanners even for unit ball graphs due to the distortion. See the example in Fig. 5.1.

It is known that, for unit disk (ball) graphs the Yao-graph with long edges (edges of length greater than 2) removed is a spanner (see, e.g., [84], Chapter 4). But no prior algorithm was known for constructing spanners for general disk graphs. We now define a *modified* Yao graph, which forms a spanner of the general disk (ball) graph. We later show how a variant of this

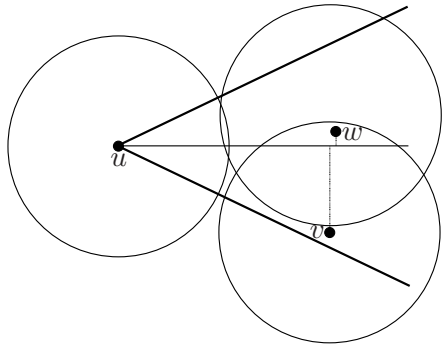


Figure 5.1. Figure illustrating the fact that Θ-graphs fail to be spanner for unit disk graphs. u, v, and w are disk centers. The point w is closer to u than v. There is an edge (u, w), but no edge (u, v) in the unit disk graph. The distance between u and the projection of v (onto the angle bisector) is less than the distance between u and the projection of w (onto the angle bisector). If we add an edge (u, v) to the Θ-graph then the path between u and v in the Θ-graph is shorter than the path between u and v in the original unit disk graph. If we don't add the edge then there may not be any path between u and w in the Θ-graph.

modified Yao graph can be constructed efficiently. Let $\mathcal{C}(p) = \{co_1(p), \ldots, co_{\epsilon^{-1}}(p)\}$ be a collection of ϵ^{-1} cones such that: (1) each cone has its apex at $p \in \mathcal{P}$, (2) each cone has an opening angle of $2\pi\epsilon$, and (3) the union of these cones cover $\mathbb{R}^2$.

Definition 8 (Modified Yao Graph)**.** *The vertices of a modified Yao graph Y are the points of $\mathcal{P}$. For each $p \in \mathcal{P}$ and integer i $(1 \leq i \leq \epsilon^{-1})$, add an edge from p to the point q contained in $co_i(p)$ if*

i. $r_q \geq r_p$,

ii. the edge (p, q) exists in G, and

iii. q is the closest point in $co_i(p)$ to p satisfying the previous two conditions.

Lemma 5.4.1. *Let (u, v) be an edge in the disk graph G. Then there exists a path in the spanner Y such that $d_Y(u, v) \leq (1 + c\epsilon)d(u, v)$ for some constant c.*

Proof. Assume $r_u \leq r_v$. Consider the cone $co_i(u)$ containing v. Let (u, w) be the edge added in Y from u to the point w which is also contained in $co_i(u)$. Let x be the projection of the point w on the line connecting u and v. See Fig. 5.2. Then $d(u, w) \leq d(u, x) + d(x, w)$ and $d(w, v) \leq d(x, v) + d(x, w)$. Adding these inequalities we get,

$$\begin{aligned} d(u, w) + d(w, v) &\leq d(u, x) + d(x, v) + 2d(x, w) \\ &= d(u, v) + 2d(x, w) \leq d(u, v) + 4\pi\epsilon d(u, w). \end{aligned}$$

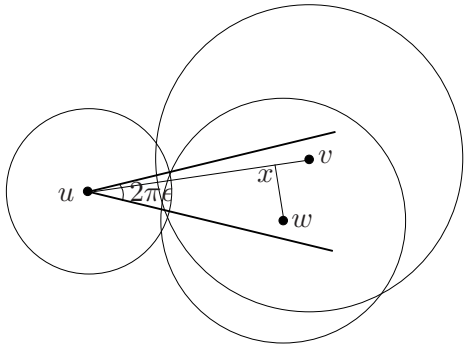


Figure 5.2. Figure illustrating the proof of Lemma 5.4.1.

The last inequality holds because $\frac{d(x,w)}{d(u,w)} \leq \sin(2\pi\epsilon) < 2\pi\epsilon$. Now for $\epsilon \leq 1/8$ (cones have opening angle ≤ 45), $d(w,v) < d(u,v)$. Since $r_u \leq r_w$ and there exists an edge (u,v) implies that there also exists an edge (w,v) in G. We complete the proof by induction. We inductively assume that Lemma 5.4.1 holds for all the finitely many pairs of (u',v') of vertices of G with $d(u',v') < d(u,v)$, in particular for $u' = w$ and $v' = v$. Therefore,

$$\begin{aligned} d_Y(u,v) &= d(u,w) + d_Y(w,v) \leq d(u,w) + (1+c\epsilon)d(w,v) \\ &\leq d(u,w) + (1+c\epsilon)(d(u,v) - (1-4\pi\epsilon)d(u,w) \leq (1+c\epsilon)d(u,v). \end{aligned}$$

The last inequality is satisfied if $c \geq 78$ and $\epsilon \leq 1/15$. If $r_u \geq r_v$, then we swap u and v in the above proof. □

From the above lemma by summing over all edges of a path in G we also get that graph Y is a $(1+\epsilon)$-spanner of the disk graph G. For ball graphs in $\mathbb{R}^k$, the number of edges in Y can be bounded by $O(n\epsilon^{-k+1})$ using the result of Lukovszki ([80], Theorem 2.22). We use the following ideas to *efficiently* construct a *variant* of the modified Yao graph which is still a spanner of the disk graph.

5.4.2 Modified Well-Separated Pair Decomposition

Let $(\mathcal{S},\rho)$ be a metric space, where $\mathcal{S}$ is a set of elements and ρ the distance function defined on $\mathcal{S}\times\mathcal{S}$. For any subset $T \subseteq \mathcal{S}$, the diameter $Dia_\rho(T)$ is defined to be $\max_{p,q\in T}\rho(p,q)$. The distance $\rho(T,T')$ between two sets $T,T' \subseteq \mathcal{S}$ is define to be $\min_{p\in T, q\in T_1}\rho(p,q)$.

Definition 9 (Well-Separated Pair Decomposition)**.** *A s-well-separated pair decomposition for $(\mathcal{S},\rho)$ for a real number $s > 0$, is a sequence $\{A_1,B_1\},\ldots,\{A_m,B_m\}$ of pairs of nonempty subsets of $\mathcal{S}$, such that:*

① *the sets A_i and B_i are disjoint, for every i,*

② *for any two distinct points p and q of $\mathcal{S}$, there is an unique pair $\{A_i, B_i\}$ such that $p \in A_i$ and $q \in B_i$, or vice-versa,*

③ *for each pair $\{A_i, B_i\}$, $\rho(A_i, B_i) \geq s \cdot \max\{Dia(A_i), Dia(B_i)\}$.*

For a point set in k-dimensional Euclidean space it is known that a ϵ^{-1}-well-separated pair decomposition with $O(\epsilon^{-k}n)$ many pairs exists [20]. Given such a pair decomposition it can be easily converted into a $(1+\epsilon)$-spanner by picking a representative edge (an actual edge of the graph) from each pair into the spanner (the conversion process is explained in detail in [19, 97]).

However, the disk graph metric being more general doesn't have the same nice properties as the Euclidean graph metric. In a recent result, Gao and Zhang [45] gave a construction of a ϵ^{-1}-well-separated pair decomposition with $O(\epsilon^{-4}n \log n)$ pairs, for unit disk graphs. They also show that for unit ball graphs in $\mathbb{R}^k$ at least $\Omega(n^{2-2/k})$ pairs are needed. However, one cannot hope to extend these results to general disks graphs, as general disk graphs do not have a sub-quadratic well-separated pair-decomposition. One such example is the star graph, formed by a big disk and $n-1$ pairwise disjoint small disks intersecting the big disk.

Loosely speaking, in our algorithm we construct set pairs that are disjoint (condition ①) and well-separated (condition ③). However (unlike Definition 9), we require that the vertices in each individual set form a clique in G. We show that for constructing the spanner when combined with ideas of the modified Yao graph, one requires only a linear number of such set pairs. The final challenge lies in minimizing the time for finding the representative edges between set pairs.

5.5 Spanners of Disk Graphs

We first describe a high-level idea of our algorithm. For simplicity, we describe all the algorithms for the dimension $k = 2$ and then state the generalizations to higher k. In order to construct a spanner G' of a disk graph G, the disks are classified by their approximate radii. A disk D_u with radius r_u is said to be of order $\lceil -\log r_u \rceil$. Now a first idea is that every disk is responsible for maintaining the connections to disks of lower or equal order only (i.e., to approximately larger disks). This requirement is relaxed when several disks of equal order are located close together. Then all disks except one representative disk are treated like disks of smaller radii, and only the representative disk is responsible for establishing longer connections.

Occasionally, a small disk D_u is isolated in the sense that there is no disk of lower order nearby. Far away large disks can still produce edges in G. Some of these long edges have to be retained in the spanner G' to establish connections with these large disks. In order for G' to have

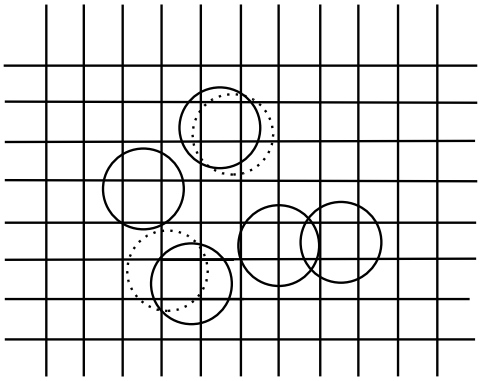


Figure 5.3. The ϵ-grid with $\epsilon = 1$. Assume all disks have the same radius. The centers of solid disks are the representative vertices used in G'.

the spanner properties, we notice that it is sufficient to retain in each approximate direction just one edge to a (relatively near) far neighbor D_v.

In order to determine clusters of disks with a common approximate radius, the plane is partitioned by a hierarchy of grids (like quadtrees). The largest disks of order 0 are classified according to the locations of their centers in the squares of a fixed grid of width ϵ. This grid is refined to a grid of width $\epsilon/2$ for the classification of disks of order 0 and 1, and so on. From each class (cluster of large disks centered in a grid box) an arbitrary representative is selected (except that any representative selected is also chosen as the representative of all smaller grid boxes containing the representative's disk center).

5.5.1 Spanner Construction Algorithm

As explained earlier, each disk is associated with an order, a disk D_u is of order l iff $2^{-l} \leq r_u < 2^{-l+1}$. Let l_{max} denote the largest order among disks in $\mathcal{D}$, i.e., $l_{max} = \lceil \log S(\mathcal{D}) \rceil$. The spanner G' is constructed in the following manner.

Quad-dissection: Our spanner construction involves recursively partitioning the plane using a simple variant of quadtrees. The input to quad-dissection is a set of disks in $\mathbb{R}^2$. Let $\mathcal{P}$ be the set of their centers. Define the *bounding box* to be the smallest axis-parallel square enclosing $\mathcal{P}$. The left bottom corner of the bounding box is assumed to be the origin. An ϵ-grid is defined by horizontal and vertical line segments drawn at $y \in \epsilon\mathbb{Z}$ and $x \in \epsilon\mathbb{Z}$ within the bounding box. A quad-dissection of the ϵ-grid is a recursive partition into smaller squares. We view the resulting structure as a 4-ary forest with the root nodes as the non-empty squares in the ϵ-grid. Each square is partitioned into four equal squares, which form its children. We continue partitioning all the non-empty squares until each disk center is contained in a separate square of size $\epsilon 2^{-l_{max}}$

or smaller. See Fig. 5.3 and Fig. 5.4.

Constructing the forest: Let $\Gamma = (V_\Gamma, E_\Gamma)$ denote the forest from the quad-dissection of the ϵ-grid. Γ is a collection of disjoint trees. We define the lth level of a forest Γ as the set of all vertices that have distance l from the roots of the trees in Γ. The set of nodes at level l in Γ corresponds to the set of non-empty squares defined by the $\epsilon 2^{-l}$-grid. For a node $t \in V_\Gamma$ of level d_t, define $D(t)$ as the set of disks whose orders are $\leq d_t$ and their centers are contained in t. Let $C(t)$ be the set of disk centers of disks in $D(t)$.

We define $Roots$ as the set of nodes in Γ where a node $t \in Roots$ if (1) t is a root of a tree in Γ, or (2) $t \in V_\Gamma$ with $C(t) \neq \emptyset$, whereas $C(p(t)) = \emptyset$ (where $p(t)$ is the parent of t in Γ).

A node t of the forest is called *interesting* if $C(t) \neq C(p(t))$. It immediately follows that all nodes in $Roots$ are interesting and that Γ has at most $2n$ interesting nodes. For efficiency, instead of Γ, we construct a compressed forest Γ' in which we only introduce the interesting nodes of Γ and shortcut the degree one internal nodes. The construction of the compressed forest follows from simple modifications of known algorithms (in [14, 20, 52]) for generating compressed quadtrees. We now describe a construction which is very similar to that of Bern *et al.* [14]. Readers familiar with the construction of compressed quadtrees can skim this construction.

Compressed Forest: We denote the compressed forest as $\Gamma' = (V_{\Gamma'}, E_{\Gamma'})$. We start by introducing some terminology. Given two fixed-point real numbers $X = \sum x_i 2^i$ and $Y = \sum y_i 2^i$, define a bitwise shuffle operation of X and Y as: $X|Y = \sum_i (2x_i + y_i)4^i$. Define $agree(X, Y)$ with $X \neq Y$ as the smallest $l \geq 0$ such that we have $\lfloor X 2^{l+1} \rfloor \neq \lfloor Y 2^{l+1} \rfloor$, i.e., agree defines the first location after the binary point in which X and Y differ. Define $round(X, l) = \lfloor X 2^l \rfloor / 2^l$.

For a point p, let $x(p)$ and $y(p)$ denote its x- and y-coordinates. Sort the points in $\mathcal{P}$ in increasing order of their shuffled representations. Let $\mathcal{P}_s = (p_1, p_2, \ldots, p_n)$ denote this sorted sequence (i.e., $x(p_{i-1})|y(p_{i-1}) \leq x(p_i)|y(p_i)$ for $2 \leq i \leq n$). Note that the (bit-fiddling) shuffle operation needn't be performed explicitly, as $agree$ can be used to compare shuffled numbers. Because if $X|Y < X'|Y'$, then either $agree(X, X') \leq agree(Y, Y')$ and $X < X'$ or $agree(X, X') > agree(Y, Y')$ and $Y < Y'$. Construct a list A_s whose ith entry a_i is the minimum of $agree(x(p_i)\epsilon^{-1}, x(p_{i-1})\epsilon^{-1})$ and $agree(y(p_i)\epsilon^{-1}, y(p_{i-1})\epsilon^{-1})$. Set $a_1 = 0$. Start the forest with a different leaf node t_j for every disk ($1 \leq j \leq n$). We maintain a stack with top representing the last inserted entry and a special bottom of stack entry $(-\infty, Null)$. When we scan a new element a_i in the list A_s we do:

i. If $a_i \geq a_{top}$, push (a_i, t_i) onto the top of the stack and increment i.

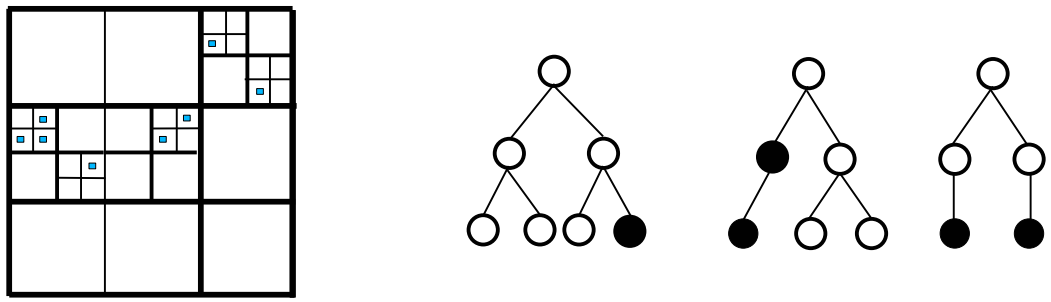


Figure 5.4. The quad-dissection procedure for a set of disk centers and the corresponding forest Γ. Assume all the disks have the same radius. The unshaded nodes are the interesting nodes in Γ.

ii. Else, let $a_{top} = a_{top-1} = \cdots = a_{top-r+2} > a_{top-r+1}$. Note that $2 \leq r \leq 4$. Pop $(a_{top}, t_{top}), \ldots, (a_{top-r+1}, t_{top-r+1})$. Create a new node t in Γ', and let its children be $t_{top}, \ldots, t_{top-r+2}, t_{top-r+1}$. Label t with a_{top}, and push $(a_{top-r+1}, t)$ on the stack.

The level of a node t (d_t) is the label of t. At every node in Γ', we also maintain the disk of the largest radius in it. The representative R_t of $t \in V_{\Gamma'}$ is chosen to be the disk of the largest radius, which was also the representative in one of the children of t. Then we visit each node t and add the children of t to $Roots$ if among the disks with centers contained in t, the largest radius is less than 2^{-d_t}. The bottom left hand corner of t has coordinates $(round(x(R_t), d_t + \log \epsilon^{-1}),$ $round(y(R_t), d_t + \log \epsilon^{-1}))$.

The time for constructing the forest is $O(n \log n)$, assuming the $agree$[3] operation can be implemented in constant time.

For convenience, we will use Γ to describe the high-level ideas and to prove the spanner property (Section 5.5.2). However, the actual algorithm uses Γ' and in the running time analysis we will be using the fact that Γ' can be constructed in $O(n \log n)$ time.

Choosing the representatives: For every leaf node $t \in V_\Gamma$ we choose the disk center in $C(t)$ as its representative R_t. Note that by the construction of Γ, each $C(t)$ is a singleton set when t is a leaf. For an internal node t, pick a child s satisfying $R_s \in C(t)$ and set $R_t = R_s$.

Neighborhood of nodes: For every interesting node t, define its *close neighborhood* ($N_c(t)$) as the set of all nodes at level d_t which are within a distance of 2^{-d_t} from t.

We also define a *far neighborhood* ($N_f(t)$) for the nodes in $Roots$. To do so, we introduce some new definitions. For a node $t \in V_\Gamma$ and integers α, β define its (α, β)-shift, $Sh((\alpha, \beta), t)$

[3]Our model of computation is the real RAM model augmented with the floor operation. The authors believe that the same running time can be obtained by just using floor of binary logarithm (a simple operation). Also the floor operation used here is not dangerous as we don't multiply variables.

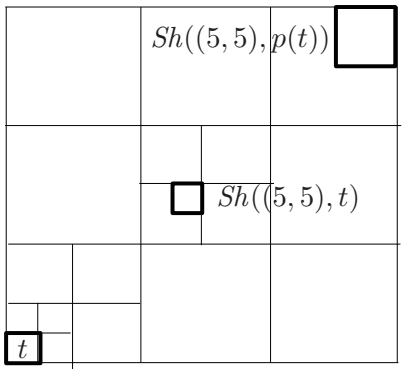


Figure 5.5. The result of $(\alpha = 5, \beta = 5)$-shift of node t and $p(t)$.

with $-4\epsilon^{-1} \leq \alpha, \beta \leq 4\epsilon^{-1}$ and $\max\{|\alpha|, |\beta|\} \geq (2\epsilon)^{-1}$, as the square t' obtained by shifting t by $\epsilon\alpha 2^{-d_t}$ in the x-direction and by $\epsilon\beta 2^{-d_t}$ in the y-direction, respectively (see Fig. 5.5). For $k > 1$, define $p^k(t) = p(p^{k-1}(t))$ (so $p^k(t)$ is the kth ancestor of t). For every $t \in Roots$, construct $O(\epsilon^{-2})$ ordered buckets, where $bucket((\alpha, \beta), t)$ is defined as

$$\begin{aligned} bucket((\alpha,\beta),t) &= \{Sh((\alpha,\beta),t), Sh((\alpha,\beta),p(t)), Sh((\alpha,\beta),p^2(t)), \dots\}, \\ Bucket(t) &= \bigcup_{\alpha,\beta} bucket((\alpha,\beta),t). \end{aligned}$$

Empty squares can be ignored in $Bucket(t)$. All nodes in a bucket are also nodes in Γ. Now for every $t \in Roots$ and every (α, β)-pair do the following: scan through $bucket((\alpha, \beta), t)$ to find the first[4] node $t' \in bucket((\alpha, \beta), t)$ such that there exists $u \in C(t)$ and $v \in C(t')$ with edge (u, v) in G. If no such t' exists, then set $t' = \emptyset$. Add t' to $N_f(t)$.

The idea behind creating the $bucket$ is (1) to ensure that for every (α, β)-pair there exist lines passing through all the nodes of $bucket((\alpha, \beta), t)$ and t, and (2) to ensure that disks that intersect any disk centered in t, have their centers either close to t or inside a node of $Bucket(t)$. The first fact easily follows from the construction. The second fact is proved by the following lemma. Remember that a point is contained in a set of squares if there exists a square in the set which contains the point.

Lemma 5.5.1. *Let u be a disk center contained in a node $t \in V_\Gamma$. Then for every edge (u, v) in G, v is contained in $Bucket(t) \cup N_c(t)$.*

Proof. Let $d(\cdot, \cdot)$ denote the Euclidean metric. If $d(u, v) \leq 2^{-d_t}$ then v is contained in $N_c(t)$.

[4]We view the $bucket((\alpha, \beta), t)$ as an ordered set, therefore the first node is the smallest square in $bucket((\alpha, \beta), t)$ satisfying the condition.

If $2^{-d_t} < d(u,v) \leq 2^{-d_t+1}$ then v is contained in $\bigcup_{\alpha,\beta} Sh((\alpha,\beta),t)$. If $2^{-d_t+1} < d(u,v) \leq 2^{-d_t+2}$ then v is contained in $\bigcup_{\alpha,\beta} Sh((\alpha,\beta),p(t))$. In general, if $2^{-d_t+k} < d(u,v) \leq 2^{-d_t+k+1}$ (for $k = 0, 1, \ldots, d_t$) then v is contained in $\bigcup_{\alpha,\beta} Sh((\alpha,\beta),p^k(t))$. □

Finding the close neighborhood: Finding the close neighborhood for nodes at level 0 is straightforward. Because we work with Γ' we construct a *close pseudo-neighborhood* ($N'_c(t)$) for a node t. A node $r \in V_{\Gamma'}$ is in $N'_c(t)$ if (1) r belongs to $N_c(t)$, or (2) in Γ the node r is the deepest level ancestor of a node $r' \in N_c(t)$ with $r' \notin V_{\Gamma'}$.

Now assuming we have found $N'_c(t)$, we describe the construction of close pseudo-neighborhood for a child s of t in Γ'. We access all the nodes in $N'_c(t)$. We do a level-order traversal from these nodes with a modification that the subtree of any node b is accessed only if $d_b \leq d_s$ and R_b is at most $(1+\sqrt{2}\epsilon)2^{-d_b}$ distance away from R_s. The nodes at which the traversal ends and whose representatives are at most $(1+\sqrt{2}\epsilon)2^{-d_s}$ away from R_s define $N'_c(s)$.

Any node $g \in V_{\Gamma'}$ accessed during the traversal has a node $f \in V_\Gamma$ such that (1) f is an ancestor of s in Γ, (2) $d_g = d_f$, and (3) the distance between f and g is $O(2^{-d_g})$. We charge the access of node g to the node f (f needn't be in Γ'). For this entire procedure we can charge all the accesses of node g to different nodes which are at the same level as g and only $O(2^{-d_g})$ away (note there are only $O(\epsilon^{-2})$ such nodes). This implies that the time for finding close pseudo-neighborhoods for all nodes is $O(n\epsilon^{-2})$.

Finding the far neighborhood for small global scale: If $S(\mathcal{D})$ is small (polynomially bounded), we use the observation from Gupta *et al.* ([51], Section 2.1) that maps the problem of reporting the intersection of a collection of disks with a query disk into halfspace range searching in two dimensions higher. Agarwal and Matoušek ([2], Theorem 1.1) construct a data structure for $\mathbb{R}^k$ which for any parameter $n \leq m \leq n^{\lfloor k/2 \rfloor}$ and any positive constant δ, after $O(m^{1+\delta})$ space and time preprocessing answers halfspace queries in $\tilde{O}(n/m^{1/\lfloor k/2 \rfloor})$ time, and has $O(m^{1+\delta}/n)$ amortized update time. We construct this data structure bottom-up. The data structure for any node $t \in V_{\Gamma'}$ stores the points in $C(t)$. Among the children of t, let s be the node having the most number of points in its data structure. We update the data structure of s to construct data structure for t. This is done by first deleting points from the data structure that are not in $C(t)$ and then by using siblings of s to insert the remaining points of $C(t)$. We then query the data structure with every disk in $C(t')$ with $t \in Bucket(t')$ to check for intersection.

If $S(\mathcal{D})$ is the global scale factor, then each $bucket((\alpha,\beta),t)$ is of size $O(\log S(\mathcal{D}))$. Therefore, $|Bucket(t)| = O(\epsilon^{-2} \log S(\mathcal{D}))$ and the representation of $Bucket(t)$ in Γ' can be found as achieved for the close neighborhood. Each disk acts as a query disk $O(\epsilon^{-2} \log S(\mathcal{D}))$ many times, so total number of queries is at most $O(n\epsilon^{-2} \log S(\mathcal{D}))$.

To balance the total time for setting up the data structure at every node and the total query time, we assume the parameter m of [2] to be n^c for some $c \geq 1$. As we work in two dimensions higher, each query can be answered in $\tilde{O}(n^{1-c/2})$ time. The total time for answering queries is $\tilde{O}(n^{2-c/2}\epsilon^{-2}\log S(\mathcal{D}))$. The total time for setting up all the data structures of [2] by the procedure described above is $O(n^{c+\delta})$, where δ is any positive constant. On eliminating c, by balancing the query and construction times, we get a space and time bound of $O(n^{4/3+\delta}\epsilon^{-2}\log^{2/3} S(\mathcal{D}))$ for finding the far neighborhood of all nodes.

Remark: The use of dynamic halfspace range query algorithm in the above construction is just for simplicity. Similar results could also be obtained by using static halfspace range query algorithms.

Finding the far neighborhood for large global scale: In this case we use the adjacency list to place the edges into the right $bucket$. For a node $t \in Roots$, we only consider an edge (u, v) if it satisfies (1) $u \in C(t)$, $v \notin C(t)$, and (2) $r_u \leq r_v$. If this is the case, then we find points v' and v'' on the line segment connecting u to v such that $2d(u, v') = d(u, v'')$, v' is contained in $\bigcup_{\alpha,\beta} Sh((\alpha, \beta), t)$, and v'' is not contained in $\bigcup_{\alpha,\beta} Sh((\alpha, \beta), t)$. Let α', β' be such that v' is contained in $Sh((\alpha', \beta'), t)$. We put the edge (u, v) in $bucket((\alpha', \beta'), t)$. Once all edges have been put into their respective $bucket$, we pick the shortest edge in each $bucket$ and add it to the spanner. The entire procedure can be implemented in $O(|E|)$ given the adjacency list of G.

Edges in spanner G′: For every interesting node t and every $t' \in N_c(t)$, we add an edge between R_t and $R_{t'}$. Additionally, for every node $t \in Roots$, we pick an edge of G between a vertex in $C(t)$ and another in $C(t')$ into the spanner, where $t' \in N_f(t)$. The following lemma proves that G' is indeed a sparse subgraph of G.

Lemma 5.5.2. *The graph G' has $O(n\epsilon^{-2})$ edges and is a subgraph of the disk graph G.*

Proof. The nodes at level d_t in Γ are defined by the set of non-empty squares of the $\epsilon 2^{-d_t}$-grid. Thus, the set $N_c(t)$ is of size $O(\epsilon^{-2})$ and for every interesting node we add $O(\epsilon^{-2})$ edges from its representative R_t. There is an edge $(R_t, R_{t'})$ with $t' \in N_c(t)$ in G because the Euclidean distance between these two points is at most $(1 + \sqrt{2}\epsilon)2^{-d_t}$ and the radii of disks centered at R_t and $R_{t'}$ are at least 2^{-d_t}.

Also for every $t \in Roots$, we add $O(\epsilon^{-2})$ additional edges and these edges are added only if they are present in G. Since there are at most $2n$ interesting nodes in Γ, the result follows. □

5.5.2 Proof of the Spanner Property

Over the next three lemmas we prove that G' is a $(1 + \epsilon)$-spanner of the disk graph G.

Lemma 5.5.3. *Let u and v be disk centers such that there exists a node $t \in V_\Gamma$ with $u, v \in C(t)$. Then there exists a path in the spanner G' such that $d_{G'}(u,v) \leq (1 + c_0\epsilon)d(u,v)$ for some constant c_0.*

Proof. Since $r_u, r_v \geq 2^{-d_t}$ and length of t is $\epsilon 2^{-d_t}$, we have $d_G(u,v) = d(u,v)$. The proof is by induction over the size of $C(t)$. The base case of $|C(t)| = 1$ is trivial as here $u = v$. Now in the subtree rooted at t, consider the deepest level such that there exist nodes a and b with $u \in C(a),\ v \in C(b)$ and an edge between R_a and R_b in G'. There exists such a and b because interesting nodes with the same parent have an edge between their representatives.

In the subtree rooted at a, find the closest descendant a' of a which is interesting with $u \in C(a')$. Let b' be the node at same level as a' with $v \in C(b')$. There exists no edge between representatives of a' and b' (because a and b are the deepest level nodes with this property). This implies that $d(u,v) \geq 2^{-d_{a'}}$. Also if not a', at least $p(a')$ contains both u and R_a (this is because a' is closest descendant of a which is interesting). Therefore for some constant c_2, we get $d(u, R_a) \leq c_2\epsilon 2^{-d_{a'}}$ and thus $d(u, R_a) \leq c_2\epsilon d(u,v)$. Using similar arguments one can show that $d(R_b, v) \leq c_2\epsilon d(u,v)$.

By the inductive hypothesis we know, $d_{G'}(u, R_a) \leq (1 + c_0\epsilon)d(u, R_a)$ and $d_{G'}(R_b, v) \leq (1+c_0\epsilon)d(R_b, v)$. In G' there exists a path from u to v of length $(1+c_0\epsilon)d(u, R_a)+d(R_a, R_b)+(1 + c_0\epsilon)d(R_b, v)$. Finally putting everything together we get

$$\begin{aligned} d_{G'}(u,v) &\leq (1 + c_0\epsilon)d(u, R_a) + d(R_a, R_b) + (1 + c_0\epsilon)d(R_b, v) \\ &\leq (2 + c_0\epsilon)d(u, R_a) + d(u,v) + (2 + c_0\epsilon)d(R_b, v) \\ &\leq (1 + c_0\epsilon)d(u,v). \end{aligned}$$

The second inequality comes from applying the triangle inequality and the final step involves substituting $d(u, R_a)$ and $d(R_b, v)$ in terms of $d(u,v)$. The constants satisfy $c_2 \leq 2\sqrt{2}$ and $c_0 \geq 4c_2/(1 - 2c_2\epsilon)$. □

Lemma 5.5.4. *Let (u,v) be an edge in the disk graph such that there exist nodes $t, t' \in V_\Gamma$ with $u \in C(t)$, $v \in C(t')$, and $t' \in N_c(t)$. Then there exists a path in the spanner G' such that $d_{G'}(u,v) \leq (1 + c_0\epsilon)d(u,v)$ for some constant c_0.*

Proof. If there exists a node t such that $u, v \in C(t)$, then by Lemma 5.5.3 the claim is true. Otherwise, consider the deepest level nodes a, b in Γ, such that (1) $u \in C(a)$, $v \in C(b)$, and (2) edge between R_a and R_b is in G'. As in Lemma 5.5.3, one can argue that $d(u, R_a) \leq c_2\epsilon d(u,v)$

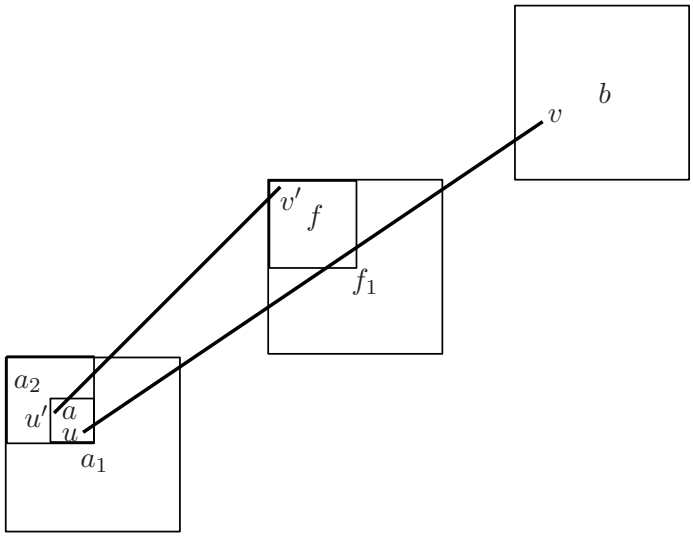


Figure 5.6. Figure illustrating the proof of Lemma 5.5.5.

and $d(R_b, v) \leq c_2 \epsilon d(u, v)$ for some constant $c_2 \leq 2\sqrt{2}$. From Lemma 5.5.3, we also know that

$$d_{G'}(u, R_a) \leq (1 + c_0\epsilon) d(u, R_a) \text{ and } d_{G'}(R_b, v) \leq (1 + c_0\epsilon) d(R_b, v).$$

The proof follows by applying the triangle inequality. □

Lemma 5.5.5. *Let (u, v) be an edge in the disk graph G. Then there exists a path in the spanner G' such that $d_{G'}(u, v) \leq (1 + c_1\epsilon) d(u, v)$ for some constant c_1.*

Proof. Assume without loss of generality that $r_u \leq r_v$. Let a be the smallest square in $Roots$ containing u. Let t be the node containing v with $d_a = d_t$. We divide the analysis into two parts based on the position of t. For the case when $t \in N_c(a)$ using similar arguments as in Lemma 5.5.4, we can show that $d_{G'}(u, v) \leq (1 + c_0\epsilon) d(u, v)$.

If $t \notin N_c(a)$. From Lemma 5.5.1, we know there exists at least one node in $Bucket(a)$ containing v (as we are in the case where v is not contained in any node of $N_c(a)$). Let $2^{-l+1} \leq d(u, v) < 2^{-l+2}$. Let $b \in bucket((\alpha', \beta'), a)$ be a node at level d_l containing v. Since $r_v \geq r_u$ and $d(u, v) \geq 2^{-l+1}$, we get that $v \in C(b)$. See Fig. 5.6.

Let $f \in bucket((\alpha', \beta'), a) \cap N_f(a)$. Let (u', v') with $u' \in C(a)$ and $v' \in C(f)$ be the edge added to the spanner. The edge (u, u') also exists in G because the minimum radius disk in $C(a)$ is at least ϵ^{-1} times a's length. In G' we know from the previous case that $d_{G'}(u, u') \leq (1 + c_0\epsilon) d(u, u')$.

Let a_1 and a_2 be the ancestors of a at levels d_b and d_f in Γ, respectively. Let f_1 be the ancestor of f in Γ at level d_b. In the following description the distance between a point and a square is the distance of the point from the center of the square.

We first claim that $d_{G'}(u,v) \leq (1+c_3\epsilon)d(a_1,b)$ for some constant c_3. Since f is obtained by shifting x- and y-coordinates of every point in a_2 by $\epsilon\alpha' 2^{-d_{a_2}}$ and $\epsilon\beta' 2^{-d_{a_2}}$ respectively, for some constant $c_2 \leq 2\sqrt{2}$ we have $d(u,a_2) \leq c_2\epsilon d(a_2,f)$, $d(f,v') \leq c_2\epsilon d(a_2,f)$, and $d(u,u') \leq c_2\epsilon d(u,v')$. Applying the triangle inequality we obtain

$$d(u,v') \leq d(u,a_2) + d(a_2,f) + d(f,v') \leq (1+2c_2\epsilon)d(a_2,f).$$

In G' for some constant c_4 we have

$$\begin{aligned} d_{G'}(u,v') &\leq d_{G'}(u,u') + d(u',v') \\ &\leq (1+c_0\epsilon)d(u,u') + d(u',u) + d(u,v') \\ &\leq (1+c_4\epsilon)d(u,v'). \end{aligned}$$

Substituting for $d(u,v')$ implies that $d_{G'}(u,v') \leq (1+c_3\epsilon)d(a_2,f)$. Note that $d(a_2,f) = d(a_1,f_1)$. To complete the claim we use induction over distances (as in Lemma 5.4.1). Since the edge (v',v) also exists in the graph by the inductive hypothesis we get $d_{G'}(v',v) \leq (1+c_3\epsilon)d(f_1,b)$. By construction $d(a_1,b) = d(a_1,f_1) + d(f_1,b)$, therefore we get $d_{G'}(u,v) \leq (1+c_3\epsilon)d(a_1,b)$.

To complete the proof we repeat similar arguments as in Lemma 5.5.4 to show that $d(a_1,b) \leq (1+c_0\epsilon)d(u,v)$. Substituting for $d(a_1,b)$ in terms of $d(u,v)$ completes the proof. □

Theorem 5.5.6. *Let G be a disk graph defined on $\mathcal{D}$. A $(1+\epsilon)$-spanner of G with $O(n\epsilon^{-2})$ edges can be constructed in $O(\min\{n^{4/3+\delta}\epsilon^{-2}\log^{2/3} S(\mathcal{D}), Adj(\mathcal{D}) + n\epsilon^{-2}\})$ time, where δ is any positive constant, and $Adj(\mathcal{D})$ is the time for constructing the adjacency list for G.*

Remark: The number of edges in G' can be reduced by a factor of ϵ^{-1}. Consider the set of cones $\mathcal{C}(p)$ for a point $p \in \mathcal{P}$ (as in Section 5.4.1). For each cone $co_i(p) \in \mathcal{C}(p)$, scan through all edges added to the spanner from p to a point contained in $co_i(p)$ and retain only the shortest such edge in the spanner. Since there are only ϵ^{-1} cones in $\mathcal{C}(p)$, we conclude that the number of edges in the spanner is $O(n\epsilon^{-1})$. Using the analysis of Lemma 5.4.1 along with Lemma 5.5.5, it follows that the graph is a $(1+\epsilon)$-spanner.

5.5.3 Extension to Ball Graphs

The quadtrees become 2^k-trees in $\mathbb{R}^k$. The close neighborhood and far neighborhood of any node is of size $O(\epsilon^{-k})$, but again one can sparsify to get it to $O(\epsilon^{-k+1})$. Therefore, the total number of edges is $O(n\epsilon^{-k+1})$. The sparsification can be done in $O(n\epsilon^{-k})$ time. In case

of small global scale using the result of range-searching from [2] we get a running time of $O(n^{2\ell+\delta}\epsilon^{-k}\log^{\ell} S(\mathcal{D}))$, where $\ell = 1 - 1/(\lfloor k/2 \rfloor + 2)$. In case of large global scale the running time is $O(|E| + n\epsilon^{-k})$ given the adjacency list of G. Using these observations, the following result is immediate:

Theorem 5.5.7. *Let G be a k-dimensional ball graph defined on $\mathcal{D}$. A $(1+\epsilon)$-spanner of G with $O(n\epsilon^{-k+1})$ edges can be constructed in $O(\min\{n^{2\ell+\delta}\epsilon^{-k}\log^{\ell} S(\mathcal{D}), Adj(\mathcal{D}) + n\epsilon^{-k}\})$ time, where δ is any positive constant, $\ell = 1 - 1/(\lfloor k/2 \rfloor + 2)$, and $Adj(\mathcal{D})$ is the time for constructing the adjacency list for G.*

5.5.4 Spanners for Unit Ball Graphs

If G was defined on unit balls (disks) we can speed up the construction considerably. The algorithm remains the same until the part where we compute the far neighborhood. Here the set $Roots$ contains only the non-empty squares of the ϵ-grid. For finding the far neighborhood of nodes in $Roots$, we solve a collection of bichromatic closest pair problems. If the disks corresponding to the closest pair intersect, we add the corresponding edge into the spanner. See Fig. 5.7.

We now state a general result describing the upper bounds of the spanner construction in terms of the upper bounds for the bichromatic closest pair problem. We assume in the rest of discussion that for some $c > 1$, $n^c f(n)$ is an upper bound on the time for computing a bichromatic closest pair for a total of n points in $\mathbb{R}^k$ (k constant), where $\log f(n) = o(\log n)$ (i.e., f grows slower than polynomial).

Lemma 5.5.8. *For unit ball graphs in $\mathbb{R}^k$ $(k > 2)$, the worst case running time for finding the far neighborhood for all nodes in $Roots$ is $O(n^c f(n)\epsilon^{-k})$.*

Proof. For every node t in $Roots$, we solve the bichromatic closest pair problem between $C(t)$ and $C(t')$, $t' \in Roots$ and $|C(t')| \leq |C(t)|$. Since t participates in $O(\epsilon^{-k})$ bichromatic closest pair problems, the cost of solving these problems (charged to t) is $O(|C(t)|^c f(|C(t)|)\epsilon^{-k})$. Since every disk center has at most one node in $Roots$ (say t) such that the disk center is in $C(t)$, the total cost of the procedure is at most:

$$O\left(\sum_{t \in Roots} |C(t)|^c f(|C(t)|)\epsilon^{-k}\right) \leq O(n^c f(n)\epsilon^{-k}).$$

□

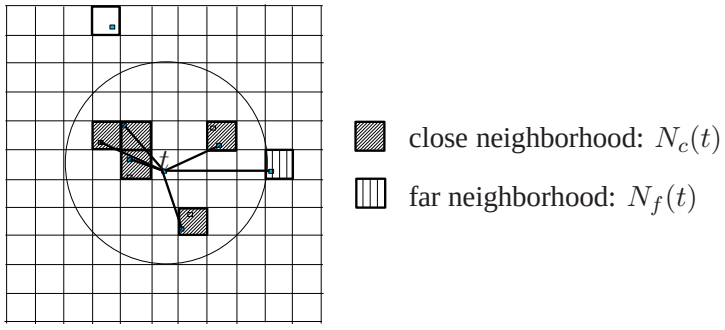


Figure 5.7. The close and far neighborhood of a node t in $Roots$ for a unit disk graph. All the squares within the circle are at most 2^{-d_t} distance away from t. Far neighborhood is determined by using bichromatic closest pair tests.

In $\mathbb{R}^k$ ($k > 2$), the currently best algorithm of Agarwal *et al.* [1] for finding a bichromatic closest pair between points sets $\mathcal{A}$ (with $\lambda_1 = |\mathcal{A}|$) and $\mathcal{B}$ (with $\lambda_2 = |\mathcal{B}|$) runs in $O((\lambda_1\lambda_2 \log\lambda_1 \log\lambda_2)^{2/3} + \lambda_1 \log^2\lambda_2 + \lambda_2 \log^2\lambda_1)$ expected time for $k = 3$ and runs in $O((\lambda_1\lambda_2)^{1-1/(\lceil k/2\rceil+1)+\delta} + \lambda_1 \log\lambda_2 + \lambda_2 \log\lambda_1)$ expected time for $k \geq 4$, where δ is any positive constant.

Theorem 5.5.9. *Let G be a unit ball graph in $\mathbb{R}^k$ ($k > 2$). A $(1+\epsilon)$-spanner of G with $O(n\epsilon^{-k+1})$ edges can be constructed in $O(n^c f(n)\epsilon^{-k})$ time.*

It is well known that the bichromatic closest pair problem in $\mathbb{R}^2$ can be solved in $O(n \log n)$ by using post-office queries [110]. The Lemma 5.5.8 can be also proved for $k = 2$, by setting $c = 1$ and $f(n) = \log n$.

Corollary 5.5.10. *The $(1+\epsilon)$-spanner G' of a unit ball graph G can be constructed in $\tilde{O}(n\epsilon^{-2})$ time for $k = 2$, in $\tilde{O}(n^{4/3}\epsilon^{-3})$ expected time for $k = 3$, and in $O(n^{2-2/(\lceil k/2\rceil+1)+\delta}\epsilon^{-k})$ expected time for $k \geq 4$, where δ is any positive constant.*

5.5.4.1 Complexity of Unit Ball Spanner $\equiv$ Complexity of EMST

Theorem 5.5.9 shows that finding a spanner of a unit ball graph in $\mathbb{R}^k$ is not much harder than computing a bichromatic closest pair for n points in $\mathbb{R}^k$. We now argue that complexities of these two problems are in fact equivalent. To make this relation more precise, define the *exponent* of a problem $\mathcal{A}$ with respect to input size n to be

$$\inf\{c \mid \text{there exists an algorithm for solving } \mathcal{A} \text{ with a running time of } O(n^c)\}.$$

Since the bichromatic closest pair and Euclidean minimum spanning tree problems have asymptotically same complexities [1, 71], their exponents are equal. The following theorem summarizes the equivalence.

Theorem 5.5.11. *The exponent of a $(1+\epsilon)$-spanner of a graph defined on n unit balls in $\mathbb{R}^k$ is the same as the exponent of a bichromatic closest pair for n points in $\mathbb{R}^k$.*

Proof. From Theorem 5.5.9, we know that exponent of the $(1+\epsilon)$-spanner is not more than the exponent of the bichromatic closest pair. For the other direction, we use the fact that the randomized expected time bounds for constructing a spanning forest of a unit ball graph in $\mathbb{R}^k$ and a Euclidean minimum spanning tree (or a bichromatic closest pair) in $\mathbb{R}^k$ are within constant factors (Eppstein [30], Theorem 4.2). Once we have a $(1+\epsilon)$-spanner any graph traversal (which can be done in linear time) can be used to construct the spanning forest of unit balls. Therefore, using the above mentioned result of Eppstein [30], we also get that the exponent of bichromatic closest pair is not more than the exponent of $(1+\epsilon)$-spanner. □

5.6 Separators in Spanner Graph

In this section we show that the spanner graph G' in $\mathbb{R}^k$ has an $O(n^{1-1/k}\epsilon^{-k+1/2}+\epsilon^{-2k+1}\log S(\mathcal{D}))$ vertex separator, whose removal leaves two sets with each having at most $7/9$ of the original vertices ($7/9$-split) with no edges going across sets. Furthermore, this separator can be found in $O(n\log n)$ time. As usual we describe our procedure for the 2-dimensional case.

We use recursive partitions of rectangles. As before the left bottom corner of the bounding box is assumed to be the origin. Let $\mathcal{X}$ be the sorted list of x-coordinates of vertices. Let $\mathcal{X}(\mathcal{R})$ be the sorted list of x-coordinates of vertices contained in rectangle $\mathcal{R}$. Similarly define $\mathcal{Y}$ and $\mathcal{Y}(\mathcal{R})$ for y-coordinates. We say a vertex crosses a line segment if any edge incident on it in G' crosses the line segment. During each step, the algorithm focuses on one rectangle, called the *active rectangle*. An active rectangle $\mathcal{R}$ has at least $2/3$ of the total vertices inside it and there exists a set of $O(\sqrt{n}\epsilon^{-3/2}+\epsilon^{-3}\log S(\mathcal{D}))$ vertices which when removed ensures that no remaining vertex has an edge in G' that crosses the boundary of $\mathcal{R}$.

At every step, the algorithm uses two line segments (called *double line separator*) to divide the currently active rectangle. A *horizontal double line separator* of an active rectangle is a set of at most two horizontal line segments that partitions the active rectangle such that there exists a set of $O(\sqrt{n}\epsilon^{-3/2}+\epsilon^{-3}\log S(\mathcal{D}))$ vertices which when removed ensures that no remaining vertex crosses either of the horizontal line segments (similarly define *vertical double line separator*). Our algorithm recursively partitions an active rectangle alternatively with a horizontal

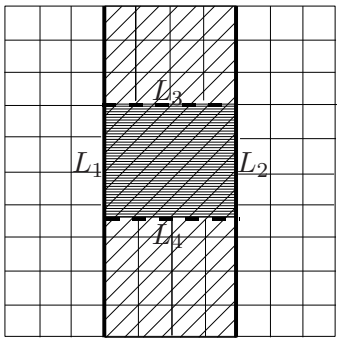


Figure 5.8. The line segments L_1, L_2, L_3, and L_4 are as defined by the algorithm. The shaded region between L_1 and L_2 represents the active rectangle after the step in which L_1 and L_2 are defined. Then after the next step the shaded region enclosed by L_1, L_2, L_3, and L_4 becomes the new active rectangle.

or a vertical double line separator and stops when none of the new rectangles created contain enough vertices to become active. The initial active rectangle is the bounding box. See Fig. 5.6.

We maintain sorted doubly linked lists for both the x- and y-coordinates with pointers between elements representing the same point in the two lists. Using this when moving to an active rectangle $\mathcal{R}$ from the previous active rectangle $\mathcal{R}_p$, the lists $\mathcal{X}(\mathcal{R})$ and $\mathcal{Y}(\mathcal{R})$ can be constructed in time proportional to the number of disk centers contained in $\mathcal{R}_p$ but which are not contained in $\mathcal{R}$.

Data Structure for double line separators: We use the following data structure to efficiently construct double line separators. Let $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$. We partition the x-axis between x_1 and x_n into intervals where every interval has at least $\sqrt{n}\epsilon^{-3/2}$ and at most $3\sqrt{n}\epsilon^{-3/2}$ consecutive points in $\mathcal{X}$. The length of the interval $[x_a, x_b]$ is the difference between the coordinates of its first and last element, i.e., $x_b - x_a$. We maintain a balanced binary search tree where leaves represent these intervals and each internal node stores a splitting value between its left and right subtree. Additionally, with each node we store the length of the longest interval in its subtree. Similarly, we construct a tree for $\mathcal{Y}$.

For updating the intervals when we go from $\mathcal{R}_p$ to $\mathcal{R}$ we use two operations: *merge* and *split plus merge*. When an interval becomes smaller than $\sqrt{n}\epsilon^{-3/2}$ we merge it with its left or right interval whichever is smaller than $2\sqrt{n}\epsilon^{-3/2}$. If neither satisfies the condition we split one of them into two halves and then do the merge. Both of these operations and the subsequent updates to the tree can be performed in $O(\log n)$ time and are only done at most n times.

Constructing double line separator for level l: Let $\mathcal{R}$ be the active rectangle when level l of Γ' is considered. We will be either dividing $\mathcal{R}$ using vertical or horizontal double line separator.

Assume without loss of generality that we are dividing $\mathcal{R}$ using vertical double line separator. We construct the vertical double line separator (line segments L_1 and L_2) for $\mathcal{R}$ as follows. Let x_m be the median of the elements in $\mathcal{X}(\mathcal{R})$. We first find the leaf node in the tree whose interval contains the median. If that leaf node represents an interval with length greater than 2^{-l+3}, then define $L_1 = L_2$ as the segment of the vertical line contained in $\mathcal{R}$ and passing through x_m. Otherwise, to construct L_1 we walk up the tree until we find the first left ancestor[5] (anc_l) having an interval of length greater than 2^{-l+3} in its subtree. We access the rightmost leaf in the left subtree of anc_l corresponding to an interval $[x_a, x_b]$ with $x_b - x_a \geq 2^{-l+3}$. Consider a vertical line ($x = x_b - 2^{-l+2}$) to the left of x_b at a distance of 2^{-l+2}. Let L_1 be the segment of the vertical line contained in $\mathcal{R}$. Similarly, we construct L_2 by considering only the right ancestors.

Over the next three lemmas we show that L_1 and L_2 satisfy the properties of vertical double line separator. For the proof, we partition the edges in G' into two sets: let E_o be the set of edges which were added to G' when the nodes in $Roots$ were considered, and E_i be the remaining edges in G'. The edges in E_o are typically the long edges, whereas the edges in E_i are the short ones. Let $\mathcal{D}(l)$ be the set of disk centers (vertices) to which edges are added when nodes of level l are considered in Γ'. Every edge in G' is *attributed* to the disk of larger radius, i.e., an edge (u, v) in G' is attributed to u if $r_u \geq r_v$. The general idea is that edges from the set E_i are separated from inside of $\mathcal{R}$ and edges from the set E_o are separated from outside of $\mathcal{R}$.

Lemma 5.6.1. *Let $\mathcal{R}$ be an active rectangle of level l. Then there exists a set of $O(\sqrt{n}\epsilon^{-3/2})$ vertices which when removed ensures that no remaining edge attributed to a vertex in $\bigcup_{l' \geq l} \mathcal{D}(l')$ cross either L_1 or L_2.*

Proof. We work with L_1. The case for L_2 is similar. The edges attributed to vertices of $\bigcup_{l' \geq l} \mathcal{D}(l')$ have length at most 2^{-l+2}. Therefore, any edge attributed to $\bigcup_{l' \geq l} \mathcal{D}(l')$ and crossing L_1 must have both its endpoints within a distance of 2^{-l+2} from L_1. By construction, we guarantee the existence of such a set of $O(\sqrt{n}\epsilon^{-3/2})$ vertices. □

Lemma 5.6.2. *The distance between L_1 and L_2 is less than $2^{-l+3}\sqrt{n}\epsilon^{3/2}$.*

Proof. Let $I_1, I_2, \ldots, I_z$ be the intervals represented by the leafs of the interval tree. Let I_y be interval ($1 \leq y \leq z$) containing the median x_m. If length of interval I_y is greater than 2^{-l+3} then $L_1 = L_2$ and the distance between L_1 and L_2 is 0. So we now assume that length of interval I_y is less than 2^{-l+3}. Let L_1 pass through the interval I_x and L_2 pass though interval $I_{x'}$ ($x < y < x'$). All the intervals $I_{x+1}, \ldots, I_{x'-1}$ have length at most 2^{-l+3}. Also all the

[5] A node b is a left (right) ancestor of a node a if either (1) a is a right (left) child of b, or (2) there exists a node c that is an ancestor of a and c is a right (left) child of b.

intervals $I_1, I_2, \ldots, I_z$ have at least $\sqrt{n}\epsilon^{-3/2}$ disk centers in them. As there are only n disks, the $x' - x - 2 \leq (n - 2\sqrt{n}\epsilon^{-3/2})/(\sqrt{n}\epsilon^{-3/2})$. Hence, the distance between L_1 and L_2 is at most $(\sqrt{n}\epsilon^{3/2} - 2)2^{-l+3} + 2^{-l+3} \leq 2^{-l+3}\sqrt{n}\epsilon^{3/2}$. □

The same distance bound holds also for the separation between L_3 and L_4. As a consequence of the above lemma, we also get that the length of L_1 and L_2 for $l \geq 1$ is at most $2^{-l_p+3}\epsilon^{3/2}\sqrt{n}$, where l_p is the level preceding l in Γ'.

Lemma 5.6.3. *Let $\mathcal{R}$ be an active rectangle at level l. Then line segments L_1 and L_2 define a vertical double line separator for $\mathcal{R}$.*

Proof. We work with L_1. The case for L_2 is similar. From Lemma 5.6.1, we know that we are done if $l = 0$. Fix any level $l' < l$. Each node (square) in level l' has an area of $(\epsilon 2^{-l'})^2$. Also, any edge attributed to a vertex in $\mathcal{D}(l')$ has length at most $2^{-l'+2}$. From Lemma 5.6.2, we know that the length of L_1 is less than $2^{-l_p+3}\sqrt{n}\epsilon^{3/2}$.

In $\mathcal{R}$ consider two vertical line segments drawn to the left and right of L_1 at a distance of $2^{-l'+2}$. Consider the rectangular region within $\mathcal{R}$, between these two vertical line segments. The area of this region is at most $2^{-l_p+3}2^{-l'+3}\epsilon^{3/2}\sqrt{n}$. Each edge in E_i attributed to a vertex in $\mathcal{D}(l')$ and crossing L_1 will have at least one endpoint in this region. The number of nodes of level l' in this region is at most $2^{l'-l_p+6}\sqrt{n}\epsilon^{-1/2}$. Since from each node at most ϵ^{-1} vertices (disk centers) are present in $\mathcal{D}(l')$, we get that the number of vertices of $\mathcal{D}(l')$ present in this region is at most $2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2}$. This forms a decreasing geometric series in l'. For levels l'' which are more than $O(\log n)$ below l_p, only constant number of disks of $\mathcal{D}(l'')$ crosses L_1. Our method of picking representatives ensures that they are the same over all such l''. Summing over all $l' < l$, we get that there exists

$$\sum_{0 \leq l' \leq l_p} 2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2} = \mathrm{O}(\sqrt{n}\epsilon^{-3/2})$$

vertices which when removed ensures that no remaining edge in E_i attributed to a vertex in $\bigcup_{l'<l} \mathcal{D}(l')$ crosses L_1.

To finish the proof we consider edges from the set E_o attributed to the vertices in $\mathcal{D}(l')$ and crossing L_1. Since nodes of *Roots* appear only in the first $\log S(\mathcal{D})$ levels of Γ', we only consider the case when $l' \leq \log S(\mathcal{D})$. Consider four lines, two vertical lines drawn at distance $2^{-l'+2}$ from L_1 on both sides and two horizontal lines drawn above and below L_1 at a distance of $2^{-l'+2}$ from its end points. The rectangular region $\mathcal{R}_{l'}$ formed by these four lines has an area at most $2^{-l'+3}(2^{-l_p+3}\sqrt{n}\epsilon^{3/2} + 2^{-l'+3})$. Each edge in E_o attributed to a vertex in $\mathcal{D}(l')$ and crossing L_1 will have both its end points in $\mathcal{R}_{l'}$. Since from each node at most ϵ^{-1} vertices are present in

$\mathcal{D}(l')$, we get that the number of vertices of $\mathcal{D}(l')$ present in $\mathcal{R}_{l'}$ is at most $2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2} + 64\epsilon^{-3}$. Summing over all $l' < l$, we get that the number of vertices in $\bigcup_{l'<l}\mathcal{D}(l')$ to which edges belonging to E_o and crossing L_1 are attributed to, is at most

$$\sum_{0\leq l'\leq \min\{l_p,\log S(\mathcal{D})\}} (2^{l'-l_p+6}\sqrt{n}\epsilon^{-3/2} + 64\epsilon^{-3}) = O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S(\mathcal{D})).$$

Therefore, adding this to the result of Lemma 5.6.1, we know that there exist $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S(\mathcal{D}))$ vertices such that every edge crossing L_1 is incident on one of them. □

Final shape of the separator: L_1 and L_2 divide $\mathcal{R}$ into at most 3 rectangles, of which only the rectangle $(\mathcal{R}_b)$ between L_1 and L_2 could be active. If $\mathcal{R}_b$ is active we repeat the same procedure with $\mathcal{Y}(\mathcal{R}_b)$ to find horizontal line segments L_3 and L_4. The active rectangle thus formed is associated with the level following l in Γ' (some levels of Γ might be skipped in Γ').

The algorithm terminates when the partition of the last active rectangle produces rectangles none of which are active. The proof of termination is based on the fact that when the algorithm considers the deepest level in Γ', the horizontal double line separator divides the currently active rectangle into exactly two non-active rectangles each of which contains at most half of the vertices of G' (so neither of them could be active).

Among the rectangles created by partitioning the last active rectangle, the rectangle $\mathcal{R}_f$ containing the largest number of disk centers forms one separated component. Since the last active rectangle was partitioned by double line separators, we know there exists a set of $O(\sqrt{n}\epsilon^{-3/2} + \epsilon^{-3}\log S(\mathcal{D}))$ vertices to which all edges crossing $\mathcal{R}_f$ are incident on. This set of vertices forms the vertex separator. Also the last active rectangle had at least $2/3$ of the vertices and it gets divided into at most 3 rectangles, implying that $\mathcal{R}_f$ definitely contains at least $2/9$ of the vertices of G'.

The algorithm also extends to higher k-dimension and we get an $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S(\mathcal{D}))$ separator. The rectangle gets replaced by a k-dimensional box and line segments are $(k-1)$-dimensional hyperplanes. The algorithm recursively partitions along every dimension.

We now summarize the main result of this section in the following theorem.

Theorem 5.6.4. *Let G be a k-dimensional ball graph defined on $\mathcal{D}$ and G' be a $(1+\epsilon)$-spanner of G constructed as described above. An $O(n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S(\mathcal{D}))$ vertex separator decomposition of G' with $7/9$-split can be found in $O(n\log n)$ time.*

5.7 Approximate Distance Queries

Gao and Zhang [45] discuss many approximate proximity problems for unit disk graphs. Using a well-separated pair decomposition, they show that a unit disk graph can be preprocessed in $O(n\sqrt{n \log n}\epsilon^{-3})$ time, such that subsequent distance queries can be answered with $(1+\epsilon)$-stretch in constant time.

We note that other than the standard advantages of using a sparse spanner for solving approximate proximity problems, the separator helps us to also support fast answering of distance queries in ball graphs. First, we define some notations that will be used throughout this section. For a set U of vertices in G, we use $Ne(U)$ to denote the neighborhood of U, i.e.,

$$Ne(U) = \{v \in V \mid \exists u \in U \text{ such that } (u, v) \in E\}.$$

We use a rooted binary tree T_G to represent a separator decomposition of a graph $G = (V, E)$. Each node $s \in T_G$ is labeled by two subsets of vertices $V(s) \subseteq V$ and $\mathit{Sep}(s) \subseteq V(s)$. Let $G(s) = (V(s), E(s))$ denote the subgraph induced by $V(s)$. Then $\mathit{Sep}(s)$ is the separator in $G(s)$. The root $r \in T_G$ has $V(r) = V$ and $\mathit{Sep}(r)$ is a separator in G. For any $s \in T_G$, the labels of its children s_0, s_1 are defined as follows: let $V_1 \subset V(s)$ and $V_2 \subset V(s)$ be the components separated by $\mathit{Sep}(s)$ in $G(s)$. Then

$$V(s_0) = V_1 \cup (\mathit{Sep}(s) \cap Ne(V_1)) \quad \text{and} \quad V(s_1) = V_2 \cup (\mathit{Sep}(s) \cap Ne(V_2)).$$

Following previous notation, let G' represent the spanner graph, Γ represent the forest from the quad-dissection. We use m to denote the number of edges in G. For a node t, denote by $p_s(t)$ the parent of t in $T_{G'}$.

5.7.1 Distance Query Algorithm

The algorithm DIST-QUERY is similar to the one used by Arikati *et al.* [5] for answering distance queries in planar graphs. The algorithm produces an estimate $\delta(u, v)$ for the distance between vertices u and v.

Let $\text{LCA}(t_1, t_2)$ denote the least common ancestor of nodes t_1, t_2 in $T_{G'}$. The shortest path need not necessarily stay inside $V(\text{LCA}(t_1, t_2))$. For this reason we also look at the ancestor subgraph of $V(\text{LCA}(t_1, t_2))$. The observation is that in such a case the shortest path has to pass through some separator vertex of these subgraphs. Using the algorithm of Schieber and Vishkin [93], the least common ancestor (LCA) queries can be answered in $O(1)$ time after linear time preprocessing. For the Step 4d, the distances used are available as they are already precomputed

in the preprocessing Step 2b.

In the next two subsections, we analyze two variants of this query algorithm, which mainly differ only in the parameter $\widehat{G}$ passed to the algorithm DIST-QUERY. For answering distance queries with a $(1+\epsilon)$-approximation we use the spanner graph G' for $\widehat{G}$. For answering distance queries with a strong $(1+\epsilon)$-approximation we use the original graph G for $\widehat{G}$.

DIST-QUERY$(G, \widehat{G}, u, v)$

Preprocessing: Let G' be a $(1+\epsilon)$-spanner of G

1. Compute $T_{G'}$ the separator decomposition tree for G'
2. For each node $s \in T_{G'}$ do
   (a) $H(s) \leftarrow$ graph induced by $V(s)$ on $\hat{G}$
   (b) From each node in *Sep*(s) do a single-source shortest path computation on $\widehat{G}$

Reporting distance between $\mathbf{u}, \mathbf{v} \in \mathbf{V}$:

3. If edge (u, v) exists, $\delta(u, v) \leftarrow d_G(u, v)$
4. Else
   (a) $\delta(u, v) \leftarrow \infty$
   (b) Let t_1 and t_2 be the leaves in $T_{G'}$ such that $u \in V(t_1)$ and $v \in V(t_2)$
   (c) $t \leftarrow$LCA(t_1, t_2), i.e., least common ancestor of t_1, t_2 in $T_{G'}$
   (d) While t is not the root of $T_{G'}$ do
      (i) $\delta(u, v) \leftarrow \min\{\delta(u, v), \min_{z \in \mathit{Sep}(t)}\{d_G(u, z) + d_G(z, v)\}\}$
      (ii) set $t = p_s(t)$
5. Report $\delta(u, v)$

5.7.1.1 $(1+\epsilon)$-Approximate Distance Query Algorithm

The algorithm (Fig. 5.9) for answering $(1+\epsilon)$-approximate distance queries relies on the fact that G' is a $(1+\epsilon)$-spanner. The following theorem analyzes the algorithm DIST-QUERY$_{(1+\epsilon)}$.

Algorithm DIST-QUERY$_{(1+\epsilon)}$
Input: ball graph $G = (V, E)$, vertices $u, v \in V$
Output: DIST-QUERY(G, G', u, v) (where G' is the spanner constructed as above)

Figure 5.9. Algorithm for $(1+\epsilon)$-approximate answering of distance queries.

Theorem 5.7.1. *Let G be a k-dimensional ball graph defined on $\mathcal{D}$ and G' be a $(1+\epsilon)$-spanner of G constructed as described above. The graph G' can be preprocessed in $\tilde{O}(nf(n, S(\mathcal{D}), \epsilon))$ time*

and space such that subsequent distance queries under the d_G metric can be answered with $(1+\epsilon)$-stretch in $O(f(n, S(\mathcal{D}), \epsilon))$ time, where $f(n, S(\mathcal{D}), \epsilon) = n^{1-1/k}\epsilon^{-k+1/2} + \epsilon^{-2k+1}\log S(\mathcal{D})$.

Proof. For every pair of query vertices u, v, the algorithm reports the exact distance between u and v in G'. Since G' is a $(1+\epsilon)$-spanner,

$$d_G(u, v) \leq d_{G'}(u, v) \leq (1+\epsilon)d_G(u, v).$$

Therefore, the answers are $(1+\epsilon)$-approximate.

The running time of the preprocessing is dominated by the time for running SSSP from all the separator nodes. Using Dijkstra for SSSP gives a running time of $\tilde{O}(nf(n, S(\mathcal{D}), \epsilon))$ for the preprocessing phase. The space needed for storing all the results of SSSP computations is $\tilde{O}(nf(n, S(\mathcal{D}), \epsilon))$. The query time is dominated by the Step 4d of DIST-QUERY and can be bounded by $O(f(n, S(\mathcal{D}), \epsilon))$. □

5.7.1.2 Strong $(1+\epsilon)$-Approximate Distance Query Algorithm

The algorithm (Fig. 5.10) for answering strong $(1+\epsilon)$-approximate distance queries uses the original graph G for $\widehat{G}$. Thus, the shortest path computations (Step 2b of algorithm DIST-QUERY) are performed on G itself. Since we compute distances using G, the distance queries can be answered more accurately. But since G has more edges than G', requirements for the preprocessing time and space are now higher. To illustrate the main ideas in our algorithm we

Algorithm DIST-QUERY$_{\text{strong}(1+\epsilon)}$
Input: ball graph $G = (V, E)$, vertices $u, v \in V$
Output: DIST-QUERY(G, G, u, v)

Figure 5.10. Algorithm for strong $(1+\epsilon)$-approximate answering of distance queries.

start by considering the easier case where all the disks have the same radius.

Theorem 5.7.2. *Let G be a unit disk graph with $m = \Omega(n \log n)$ edges. The graph G can be preprocessed in $O(m\sqrt{n}\epsilon^{-1})$ time, producing a data structure of size $O(n^{3/2}\epsilon^{-1})$, such that subsequent distance queries can be answered approximately, in $O(\sqrt{n}\epsilon^{-1})$ time. The outputs produced are strong $(1+\epsilon)$-approximate distance estimates.*

Proof. If there is an edge between u and v in G, then the actual distance is the estimate. Otherwise, we know $d_G(u, v) > 2$. Consider a shortest path $P_{sh} = (u_1, u_2, \ldots, u_z)$ between $u_1 = u$ and $u_z = v$ with $k > 2$ and no shortcuts, i.e., no edge from u_{j-1} to u_{j+1} for any j between 2 and $k-1$. Such a path P_{sh} exists due to the triangle inequality. Since $d(u_{j-1}, u_{j+1}) >$

2, $\max\{d(u_{j-1}, u_j), d(u_j, u_{j+1})\} \geq 1$. Let $(sq(u_1), sq(u_2), \ldots, sq(u_z))$ be the sequence of squares in $Roots$ such that $sq(u_i)$ contains u_i. Remember that a square contains a point if the point lies within the boundary of the squares. We know the path $(R_{sq(u_1)}, \ldots, R_{sq(u_z)})$ exists in G' (by construction). Let s be the deepest level node in $T_{G'}$ where this path gets separated. This implies that among the vertices $\{R_{sq(u_1)}, \ldots, R_{sq(u_z)}\}$ at least one is in $Sep(s)$, say $R_{sq(u_i)}$.

We compute single source shortest path from the vertex $R_{sq(u_i)}$, which is at most $\sqrt{2}\epsilon$ distance away from u_i. There is also an edge between u_i and $R_{sq(u_i)}$ in G. This proves that the estimate is only $O(\epsilon)$ greater than the length of an actual shortest path. It is also a strong $(1+\epsilon)$-approximate estimate, because there exists an edge in the path of P_{sh} with length at least 1.

The running time of the preprocessing is dominated by the time for running SSSP from all the separator nodes. Using Dijkstra's algorithm for SSSP gives a running time of $O(m\sqrt{n}\epsilon^{-1})$ for the preprocessing phase. Note that since $m = \Omega(n \log n)$, every run of Dijkstra's algorithm can be performed in $O(m)$ time using Fibonacci heaps. The space needed for storing all the results of SSSP computations is $O(n^{3/2}\epsilon^{-1})$. The query time is dominated by the Step 4d and can be bounded by $O(\sqrt{n}\epsilon^{-1})$. □

Strong $(1+\epsilon)$-approximate Distance Queries in Ball Graphs: Again we describe the algorithm for the case of disk graphs and then state the extensions to higher dimensions. For two squares s and s' in Γ, the distance $dist(s, s')$ is the Euclidean distance between their centers. Extending the notion of representative vertices, we define representative paths.

We define a *representative path* in G' for every edge of G. For a vertex w in G, let $sq(w)$ denote the deepest level node in Γ containing w. Consider an edge (u, v) of the graph G. In the rest of the discussion we assume without loss of generality that $r_u \leq r_v$. The representative path $P(u, v)$ starts at $R_{sq(u)}$ and ends at $R_{sq(v)}$.

Let a be the deepest level node in $Roots$ containing u. Let b be a node in Γ containing v with $d_a = d_b$. Since u and $R_{sq(u)}$ (similarly v and $R_{sq(v)}$) are always contained in the same node in Γ, we get that $R_{sq(u)}$ is contained in a and $R_{sq(v)}$ is contained in b. The representative path $P(u, v)$ for an edge (u, v) is defined using the following case distinction:

Case 1: If the distance between a and b is greater than 2^{-d_a}, then by Lemma 5.5.1, we know that there exists some $c \in Bucket(a)$ containing v (and thus also $R_{sq(v)}$). Since $dist(a, c) \geq 2^{-d_c}$, we also know that $r_v \geq 2^{-d_c-1}$ and $v \in C(c)$. In G' there is an edge between R_a and R_c. From Lemma 5.5.3, we know there exists a path in G' between $R_{sq(u)}$ and R_a and between $R_{sq(v)}$ and R_c. Define the representative path $P(u, v)$ as $(R_{sq(u)}, \ldots, R_a, R_c, \ldots, R_{sq(v)})$.

Case 2: Otherwise, consider the deepest level nodes f, g in Γ, such that (1) $R_{sq(u)}$ is contained in f and $R_{sq(v)}$ is contained in g, and (2) there exists an edge (R_f, R_g) in G'. From Lemma 5.5.3,

we know there exists a path between $R_{sq(u)}$ to R_f and between $R_{sq(v)}$ and R_g in G'. Define the representative path $P(u,v)$ as $(R_{sq(u)}, \ldots, R_f, R_g, \ldots, R_{sq(v)})$.

For every representative path, we also define a pair of nodes in Γ as its *covering nodes*. If $P(u,v)$ is defined using Case 1, then the nodes a and c are the covering nodes. If $P(u,v)$ is defined using Case 2, then f and g are the covering nodes. In both cases, all vertices in $P(u,v)$ are contained in one of the covering nodes with u and v contained in different covering nodes.

Lemma 5.7.3. *Let (u,v) be an edge in G of length greater than $2^{-l_{max}}$. Let $P(u,v)$ be its representative path in G' with p and q as the covering nodes. Then dist$(p,q) \geq \max\{c_5 2^{-d_p}, c_5 2^{-d_q}\}$, for some constant c_5.*

Proof. We again use the same case distinction as in defining the path $P(u,v)$. Assume p contains u, and q contains v. If $P(u,v)$ is defined using Case 1, then $d_p \geq d_q$. Let r be the ancestor of p which is at the same level as q. Since q is at least 2^{-d_q} away from r, we get that *dist*$(p,q) \geq c_5 2^{-d_q}$ (p is a square inside r).

If $P(u,v)$ is defined using Case 2, then $d_p = d_q$. Let p' be the child of p containing u. Let q' be the child of q containing v. Since there is no edge between $R_{p'}$ and $R_{q'}$, we conclude *dist*$(p',q') \geq 2^{-d_q-1}$. This implies that *dist*$(p,q) \geq c_5 2^{-d_q}$. The existence of nodes p' and q' is guaranteed if $d(u,v) \geq (1+\epsilon/\sqrt{2})2^{-l_{max}}$ (remember that l_{max} is the deepest level in Γ). Otherwise, *dist*$(p,q) = \Omega(2^{l_{max}})$ and again *dist*$(p,q) \geq c_5 2^{-d_q}$. The constant $c_5 \geq 1/4$. □

The following theorem shows that we get a strong $(1+\epsilon)$-approximation.

Theorem 5.7.4. *Let G be a disk graph on $\mathcal{D}$ with $m = \Omega(n \log n)$ edges. The graph G can be preprocessed in $O(m\sqrt{n}\epsilon^{-1} + m\epsilon^{-2}\log S(\mathcal{D}))$ time, producing a data structure of size $O(n^{3/2}\epsilon^{-1} + n\epsilon^{-2}\log S(\mathcal{D}))$, such that subsequent distance queries can be answered approximately, in $O(\sqrt{n}\epsilon^{-1} + \epsilon^{-2}\log S(\mathcal{D}))$ time. The outputs produced are strong $(1+\epsilon)$-approximate distance estimates.*

Proof. If there is an edge between u and v in G, then the actual distance is the estimate. As in Theorem 5.7.2, consider a shortest path $P_{sh} = (u_1, u_2 \ldots, u_z)$ between $u_1 (= u)$ and $u_z (= v)$ with $k > 2$ and no edge shortcuts. Since there is always an edge if the distance between two vertices is less than $2^{-l_{max}+1}$, we get $d(u_{j-1}, u_{j+1}) \geq 2^{-l_{max}+1}$ and $\max\{d(u_{j-1}, u_j), d(u_j, u_{j+1})\} \geq 2^{-l_{max}}$ for any j between 2 and $k-1$. Let $(sq(u_1), sq(u_2), \ldots, sq(u_z))$ be the sequence of squares in Γ such that $sq(u_i)$ contains u_i. Let s be the deepest level node in $T_{G'}$ with $R_{sq(u_1)}, R_{sq(u_z)} \in V(s)$.

If any vertex $R_{sq(u_i)}$ from $\{R_{sq(u_1)}, \ldots, R_{sq(u_z)}\}$ is in the separator *Sep*(s), then we do a single source shortest path from $R_{sq(u_i)}$ on $H(s)$ (graph constructed in Step 2a) of the algorithm DIST-QUERY). Now $R_{sq(u_i)}$ is at most $\epsilon 2^{-l_{max}+1/2}$ distance away from u_i and there exists an edge between $R_{sq(u_i)}$ and u_i in G. This proves the estimate produced is only $O(\epsilon)$ greater than the shortest path. The strong $(1+\epsilon)$-approximation follows as there is an edge in P_{sh} which is of length at least $2^{-l_{max}}$.

Otherwise, consider two vertices $R_{sq(u_a)}$ and $R_{sq(u_b)}$ from $\{R_{sq(u_1)}, \ldots, R_{sq(u_z)}\}$ which are separated into different components by *Sep*(s) and for which there exists an edge (u_a, u_b) in G (by the assumption about the path P_{sh} we get $|a-b| = 1$). Since there is no edge between $R_{sq(u_a)}$ and $R_{sq(u_b)}$, we conclude $d(u_a, u_b) \geq 2^{-l_{max}}$. Consider the representative path $P(u_a, u_b)$ in G'. There exists at least one vertex (say z) on the path $P(u_a, u_b)$ in the separator *Sep*(s). Let p and q be the two covering nodes for $P(u_a, u_b)$. By definition all the vertices in $P(u_a, u_b)$ are contained in either p or q. Assume without loss of generality that z and u_a are contained in p.

In G there exist edges (z, R_p) and (R_p, u_a). Since z, R_p and u_a are all contained in p we get $d(z, R_p), d(R_p, u_a) \leq \epsilon 2^{-d_p+1/2}$. On the other hand, *dist*$(p, q) \geq \max\{c_5 2^{-d_p}, c_5 2^{-d_q}\}$ by Lemma 5.7.3. This also implies that $d(u_a, u_b) \geq c_6 2^{-d_p}$ for some constant c_6. Putting all together we get $d(z, R_p) \leq c_7 \epsilon d(u_a, u_b)$ and $d(R_p, u_a) \leq c_7 \epsilon d(u_a, u_b)$ for some constant c_7, implying in G there is a path from z to u_a of length at most $2c_7 \epsilon d(u_a, u_b)$. Therefore, when we do the single source shortest path computation from z on $H(s)$, the error we make in taking the detour can be bounded by some constant times $\epsilon d(u_a, u_b)$. Implying that our estimate is a strong $(1+\epsilon)$-approximation.

The running time analysis follows as in Theorem 5.7.2. The preprocessing time and space are related to the size of the separator in G. □

Given a ball graph G in $\mathbb{R}^k$, its spanner G' can be constructed in $O(n^{2\ell+\delta}\epsilon^{-k}\log^{\ell} S(\mathcal{D}))$ time, where $\ell = 1 - 1/(\lfloor k/2 \rfloor + 2)$ (Theorem 5.5.7). The separator decomposition of G' can be constructed in $O(n \log n)$ time (Theorem 5.6.4). By repeating the same analysis as in the case of disk graphs (Theorem 5.7.4), we have the following result.

Theorem 5.7.5. *Let G be a k-dimensional ball graph on $\mathcal{D}$ with $m = \Omega(n \log n)$ edges. The graph G can be preprocessed in $O(mn^{1-1/k}\,\epsilon^{-k+1} + m\epsilon^{-k}\log S(\mathcal{D}))$ time, producing a data structure of size $O(n^{2-1/k}\,\epsilon^{-k+1} + n\epsilon^{-k}\log S(\mathcal{D}))$, such that subsequent distance queries can be answered approximately, in $O(n^{1-1/k}\,\epsilon^{-k+1} + \epsilon^{-k}\log S(\mathcal{D}))$ time. The outputs produced are strong $(1+\epsilon)$-approximate distance estimates.*

Chapter 6

Conclusion and Open Problems

In this thesis, we presented approximation algorithms for the following two problems: (a) counting the copies of a graph in another graph and (b) estimating distances in geometric intersection graphs. In this chapter, we conclude with some exciting open problems for future research.

Approximation Algorithms for Graph Counting Problems: In Chapter 3, we presented the first general subcase of the subgraph isomorphism problem that is almost always efficiently approximable. The natural question arising from this work is what other classes of graphs have an ordered bipartite decomposition and more importantly which of them have one of bounded width. Another interesting problem would be to investigate the general complexity of the ordered bipartite decomposition and possibly characterize its relation to other existing graph decomposition schemas. The notion of bounded width decomposition is a natural sufficient condition for the class of algorithms based on the principle of the function Count (Chapter 2) to give almost always an fpras. But the necessary condition for the general approach to work is still unclear. Finally, a challenging open problem is to obtain any such general result for counting in arbitrary dense graphs.

In Chapter 4, we presented a very simple, randomized algorithm for approximately counting the perfect matchings of a graph. We have also shown that for almost all graphs the estimator runs in time almost linear in the size of the graph. This is the fastest known algorithm for approximately counting perfect matchings of random graphs. To do better than $O(n^2)$, one could think of an estimator that inspects only a fraction of edges. Indeed, such unbiased estimators can easily be constructed by estimating the number of edges by sampling. However, on the flip side, such a sub-linear estimator may have a much higher variance. Designing sub-linear estimators for counting problems would be an interesting research direction.

In this thesis, we have concentrated on counting graph copies, but the general framework of function Count can be applied to solve similar combinatorial counting problems. It would be interesting to see how such simple estimators perform on other combinatorial counting problems.

Distance Approximation in Geometric Intersection Graphs: In Chapter 5, we presented the first algorithm for constructing sparse spanners in geometric intersection graphs. The bottleneck in the running time of our spanner algorithms is the time for finding the intersections between sets of balls. Faster algorithms for finding the intersections would improve the running time of the spanner construction. For disk graphs, where all disks have almost the same radius (between $[1-\epsilon, 1]$), our spanner construction for unit disk graphs still works by replacing the bichromatic closest pair algorithm with Sharir's [94] bichromatic disk intersection algorithm. Therefore, we get a running time of $O(n\epsilon^{-2}\log^2 n)$.

An obvious open problem is to obtain algorithms with improved running time. We would suspect that some of the n^{δ} in the construction time could be reduced to $poly(\log n)$ by a more careful analysis of the data structure used for halfspace range searching. Another interesting problem would be to investigate faster algorithms for constructing the modified Yao graph.

All the results presented in this paper can also be extended to cases when intersections are between squares, or regular polygons, or other disk-like objects, as well as their higher dimensional versions. This follows as our algorithms don't use any special properties of balls (or disks). However, the partitioning scheme only works if all the objects have almost the same aspect ratio. An interesting open problem is to extend the results to intersection graphs between objects not having this property.

Bibliography

[1] AGARWAL, P. K., EDELSBRUNNER, H., SCHWARZKOPF, O., AND WELZL, E. Euclidean minimum spanning trees and bichromatic closest pairs. *Discrete & Computational Geometry. 6* (1991), 407–422.

[2] AGARWAL, P. K., AND MATOUŠEK, J. Dynamic half-space range reporting and its applications. *Algorithmica 13*, 4 (Apr. 1995), 325–345.

[3] ALBER, J., AND FIALA, J. Geometric separation and exact solutions for the parameterized independent set problem on disk graphs. *Journal of Algorithms 52*, 2 (2004), 134–151.

[4] ALON, N., FRIEZE, A. M., AND WELSH, D. Polynomial time randomized approximation schemes for Tutte-Gröthendieck invariants: The dense case. *Random Structures & Algorithms 6*, 4 (1995), 459–478.

[5] ARIKATI, S. R., CHEN, D. Z., CHEW, L. P., DAS, G., SMID, M. H. M., AND ZAROLIAGIS, C. D. Planar spanners and approximate shortest path queries among obstacles in the plane. In *ESA '96* (1996), vol. 1136, Springer, pp. 514–528.

[6] ARORA, S., AND BARAK, B. *Computational Complexity: A Modern Approach*. Cambridge University, 2008.

[7] ARTYMIUK, P. J., BATH, P. A., GRINDLEY, H. M., PEPPERRELL, C. A., POIRRETTE, A. R., RICE, D. W., THORNER, D. A., WILD, D. J., WILLETT, P., ALLEN, F. H., AND TAYLOR, R. Similarity searching in databases of three-dimensional molecules and macromolecules. *Journal of Chemical Information and Computer Sciences 32* (1992), 617–630.

[8] ARYA, S., DAS, G., MOUNT, D. M., SALOWE, J. S., AND SMID, M. Euclidean spanners: Short, thin, and lanky. In *STOC '95* (1995), ACM, pp. 489–498.

[9] ARYA, S., MOUNT, D. M., AND SMID, M. Randomized and deterministic algorithms for geometric spanners of small diameter. In *FOCS* (1994), IEEE, pp. 703–712.

[10] ARYA, S., MOUNT, D. M., AND SMID, M. H. M. Dynamic algorithms for geometric spanners of small diameter: Randomized solutions. *CGTA: Computational Geometry: Theory and Applications 13*, 2 (1999), 91–107.

[11] AWERBUCH, B. Complexity of network synchronization. *Journal of the ACM 32*, 4 (1985), 804–823.

[12] BARABÁSI, A.-L. *Linked: The New Science of Networks*. Perseus Books Group, 2002.

[13] BASWANA, S., KAVITHA, T., MEHLHORN, K., AND PETTIE, S. New constructions of (α, β)-spanners and purely additive spanners. In *SODA* (2005), SIAM, pp. 672–681.

[14] BERN, M. W., EPPSTEIN, D., AND TENG, S.-H. Parallel construction of quadtrees and quality triangulations. *International Journal of Computational Geometry & Applications 9*, 6 (1999), 517–532.

[15] BEZÁKOVÁ, I., BHATNAGAR, N., AND VIGODA, E. Sampling binary contingency tables with a greedy start. In *SODA* (2006), SIAM, pp. 414–423.

[16] BEZÁKOVÁ, I., STEFANKOVIC, D., VAZIRANI, V. V., AND VIGODA, E. Accelerating simulated annealing for the permanent and combinatorial counting problems. In *SODA* (2006), SIAM, pp. 900–907.

[17] BOLLOBÁS, B. Random graphs. In *Combinatorics (Swansea, 1981)*, vol. 52 of *London Math. Soc. Lecture Note Ser.* Cambridge Univ. Press, 1981, pp. 80–102.

[18] BOLLOBÁS, B. *Random Graphs*. Academic Press, London, England, 1985.

[19] CALLAHAN, P. B., AND KOSARAJU, S. R. Faster algorithms for some geometric graph problems in higher dimensions. In *SODA '93* (1993), SIAM, pp. 291–300.

[20] CALLAHAN, P. B., AND KOSARAJU, S. R. A decomposition of multidimensional point sets with applications to K-nearest-neighbors and N-body potential fields. *Journal of ACM 42*, 1 (1995), 67–90.

[21] CHIEN, S. A determinant-based algorithm for counting perfect matchings in a general graph. In *SODA* (2004), SIAM, pp. 728–735.

[22] CHIEN, S., RASMUSSEN, L., AND SINCLAIR, A. Clifford algebras and approximating the permanent. *Journal of Computer and System Sciences 67*, 2 (2003), 263–290.

[23] DONG, H., WU, Y., AND DING, X. An ARG representation for chinese characters and a radical extraction based on the representation. In *International Conference on Pattern Recognition* (1988), pp. 920–922.

[24] DOR, D., HALPERIN, S., AND ZWICK, U. All-pairs almost shortest paths. *SIAM Journal on Computing 29*, 5 (2000), 1740–1759.

[25] DYER, M., FRIEZE, A., AND KANNAN, R. A random polynomial-time algorithm for approximating the volume of convex bodies. *Journal of the ACM 38*, 1 (Jan. 1991), 1–17.

[26] DYER, M., FRIEZE, A. M., AND JERRUM, M. Approximately counting Hamilton paths and cycles in dense graphs. *SIAM Journal on Computing 27*, 5 (1998), 1262–1272.

[27] DYER, M. E., AND GREENHILL, C. S. On markov chains for independent sets. *Journal of Algorithms 35*, 1 (2000), 17–49.

[28] ELKIN, M., AND PELEG, D. $(1+\epsilon,\beta)$-spanner constructions for general graphs. *SIAM Journal on Computing 33*, 3 (2004), 608–631.

[29] EPPSTEIN, D. Spanning trees and spanners. In *Handbook of Computational Geometry*, J.-R. Sack and J. Urrutia, Eds. Elsevier, 2000, ch. 9, pp. 425–461.

[30] EPPSTEIN, D. Testing bipartiteness of geometric intersection graphs. In *SODA '04* (2004), SIAM, pp. 860–868.

[31] ERDŐS, P., AND RÉNYI, A. On the evolution of random graphs. *Publication of Mathematical Institute of Hungarian Academy of Science 5* (1960), 17–61.

[32] ERICKSON, J. On the relative complexities of some geometric problems. In *CCCG '95* (1995), pp. 85–90.

[33] FEDER, T., AND MIHAIL, M. Balanced matroids. In *STOC* (1992), ACM Press, pp. 26–38.

[34] FRIEZE, A. M., AND JERRUM, M. An analysis of a Monte Carlo algorithm for estimating the permanent. *Combinatorica 15*, 1 (1995), 67–83.

[35] FRIEZE, A. M., JERRUM, M., MOLLOY, M. K., ROBINSON, R., AND WORMALD, N. C. Generating and counting Hamilton cycles in random regular graphs. *Journal of Algorithms 21*, 1 (1996), 176–198.

[36] FRIEZE, A. M., AND MCDIARMID, C. Algorithmic theory of random graphs. *Random Structures & Algorithms 10*, 1-2 (1997), 5–42.

[37] FRIEZE, A. M., AND SUEN, S. Counting the number of Hamilton cycles in random digraphs. *Random Structures & Algorithms 3*, 3 (1992), 235–242.

[38] FU, B. Theory and application of width bounded geometric separator. In *STACS '06* (2006), vol. 3884, Springer, pp. 277–288.

[39] FÜRER, M., AND KASIVISWANATHAN, S. Approximately counting embeddings into random graphs. In *APPROX-RANDOM* (2008), Springer, pp. 416–429.

[40] FÜRER, M., AND KASIVISWANATHAN, S. P. An almost linear time approximation algorithm for the permanent of a random (0-1) matrix. In *FSTTCS* (2004), vol. 3328, Springer, pp. 263–274.

[41] FÜRER, M., AND KASIVISWANATHAN, S. P. Approximately counting perfect matchings in general graphs. In *ALENEX/ANALCO* (2005), SIAM, pp. 263–272.

[42] FÜRER, M., AND KASIVISWANATHAN, S. P. Approximate distance queries in disk graphs. In *WAOA '06* (2006), vol. 4368, Springer, pp. 174–187.

[43] FÜRER, M., AND KASIVISWANATHAN, S. P. Spanners for geometric intersection graphs. In *WADS '07* (2007), vol. 4619, Springer, pp. 312–324.

[44] GAO, J., GUIBAS, L. J., HERSHBERGER, J., ZHANG, L., AND ZHU, A. Geometric spanner for routing in mobile networks. In *MobiHoc '01* (2001), pp. 45–55.

[45] GAO, J., AND ZHANG, L. Well-separated pair decomposition for the unit-disk graph metric and its applications. *SIAM Journal on Computing 35*, 1 (2005), 151–169.

[46] GODSIL, C., AND GUTMAN, I. On the matching polynomial of a graph. *Algebraic Methods in Graph Theory* (1981), 241–249.

[47] GUDMUNDSSON, J. Constructing sparse t-spanners with small separators. In *FCT '03* (2003), vol. 2751, Springer, pp. 86–97.

[48] GUDMUNDSSON, J. Personal communication, 2006.

[49] GUDMUNDSSON, J., LEVCOPOULOS, C., NARASIMHAN, G., AND SMID, M. Approximate distance oracles for geometric graphs. In *SODA '02)* (2002), ACM, pp. 828–837.

[50] GUDMUNDSSON, J., LEVCOPOULOS, C., NARASIMHAN, G., AND SMID, M. H. M. Approximate distance oracles revisited. In *ISAAC '02* (2002), vol. 2518, Springer, pp. 357–368.

[51] GUPTA, P., JANARDAN, R., AND SMID, M. Algorithms for some intersection searching problems involving circular objects. *International Journal of Mathematical Algorithms 1* (1999), 35–52.

[52] HAR-PELED, S. Approximation algorithms in geometry. Available at: `http://valis.cs.uiuc.edu/~sariel/research/book/aprx/`.

[53] HEUN, V., AND MAYR, E. W. Embedding graphs with bounded treewidth into optimal hypercubes. *Journal of Algorithms 43*, 1 (2002), 17–50.

[54] HOPCROFT, J. E., AND KARP, R. M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing 2*, 4 (1973), 225–231.

[55] HUNT, H. B., MARATHE, M. V., RADHAKRISHNAN, V., RAVI, S. S., ROSENKRANTZ, D. J., AND STEARNS, R. E. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *Journal of Algorithms 26*, 2 (1998), 238–274.

[56] JANSON, S. Poisson approximation for large deviations. *Random Structures & Algorithms 1*, 2 (1990), 221–230.

[57] JANSON, S. The numbers of spanning trees, Hamilton cycles and perfect matchings in a random graph. *Combinatorics, Probability and Computing* (1994), 97–126.

[58] JANSON, S., ŁUCZAK, T., AND RUCIŃSKI, A. *Random graphs*. Wiley-Interscience, 2000.

[59] JERRUM, M. *Counting, Sampling and Integrating: Algorithms and Complexity*. Birkhäuser, 2003.

[60] JERRUM, M., AND SINCLAIR, A. Approximating the permanent. *SIAM Journal on Computing 18*, 6 (Dec. 1989), 1149–1178.

[61] JERRUM, M., AND SINCLAIR, A. Polynomial-time approximation algorithms for the Ising model. *SIAM Journal on Computing 22*, 5 (1993), 1087–1116.

[62] JERRUM, M., AND SINCLAIR, A. The Markov chain Monte-Carlo method: An approach to approximate counting and integration. In *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997, 1997, pp. 482– 520.

[63] JERRUM, M., SINCLAIR, A., AND VIGODA, E. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *Journal of ACM 51*, 4 (2004), 671–697.

[64] JERRUM, M., VALIANT, L. G., AND VAZIRANI, V. V. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science 43* (1986), 169–188.

[65] JERRUM, M. R., VALIANT, L. G., AND VAZIRANI, V. V. Random generation of combinatorial structures from a uniform distribution. *Theoretical Computer Science 43*, 2–3 (1986), 169–188.

[66] KARMARKAR, N., KARP, R., LIPTON, R., LOVÁSZ, L., AND LUBY, M. A Monte-Carlo algorithm for estimating the permanent. *SIAM Journal on Computing 22*, 2 (Apr. 1993), 284–293.

[67] KARP, R. M., AND LUBY, M. Monte-Carlo algorithms for the planar multiterminal network reliability problem. *Journal of Complexity 1*, 1 (1985), 45–64.

[68] KENYON, C., RANDALL, D., AND SINCLAIR, A. Matchings in lattice graphs. In *STOC '93* (1993), ACM Press, pp. 738–746.

[69] KNUTH, D. E. Estimating the efficiency of backtrack programs. *Mathematics of Computation 29*, 129 (Jan. 1975), 121–136.

[70] KRUMKE, S. O., MARATHE, M. V., AND RAVI, S. S. Models and approximation algorithms for channel assignment in radio networks. *Wireless Networks 7*, 6 (2001), 575–584.

[71] KRZNARIC, D., LEVCOPOULOS, C., AND NILSSON, B. J. Minimum spanning trees in d dimensions. *Nordic Journal of Computing 6*, 4 (1999), 446–461.

[72] KUHN, F., AND ZOLLINGER, A. Ad-hoc networks beyond unit disk graphs. In *DIALM-POMC '03* (2003), ACM, pp. 69–78.

[73] LA PENA, V. D., AND GINÉ, E. *Decoupling, from Dependence to Independence*. Springer Verlag, New York, 1999.

[74] LEVINSON, R. Pattern associativity and the retrieval of semantic networks. *Computers & Mathematics with Applications 23*, 6–9 (1992), 573–600.

[75] LI, X.-Y. Algorithmic, geometric and graphs issues in wireless networks. *Wireless Communications and Mobile Computing 3*, 2 (2003).

[76] LI, X.-Y., CALINESCU, G., AND WAN, P.-J. Distributed construction of planar spanner and routing for ad hoc wireless networks. In *INFOCOM '02* (2002), vol. 3, IEEE, pp. 1268–1277.

[77] LI, X.-Y., WAN, P.-J., AND FRIEDER, O. Coverage in wireless ad hoc sensor networks. *IEEE Transactions on Computers 52*, 6 (2003), 753–763.

[78] LI, X.-Y., WAN, P.-J., AND WANG, Y. Power efficient and sparse spanner for wireless ad hoc networks. In *IEEE International Conference on Computer Communications and Networks (ICCCN01)* (2001), IEEE, pp. 564–567.

[79] LI, X.-Y., AND WANG, Y. Efficient construction of low weight bounded degree planar spanner. In *COCOON '03* (2003), vol. 2697, Springer, pp. 374–384.

[80] LUKOVSZKI, T. New results on geometric spanners and their applications. PhD. thesis, University of Paderborn, 1999.

[81] LUKS, E. M. Isomorphism of graphs of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences 25* (1982), 42–65.

[82] MEAD, C., AND CONWAY, L. *Introduction to VLSI System*. Addison-Wesley, Reading, 1980.

[83] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, 1995.

[84] NARASIMHAN, G., AND SMID, M. *Geometric Spanner Networks*. Cambridge University Press, 2007.

[85] PELEG, D., AND SCHÄFFER, A. A. Graph spanners. *J. Graph Theory 13* (1989), 99–116.

[86] PURDOM, P. W. Tree size by partial backtracking. *SIAM Journal of Computing 7*, 4 (1978), 481–491.

[87] RAJARAMAN, R. Topology control and routing in ad hoc networks: A survey. *SIGACT News 33* (2002), 60–73.

[88] RASMUSSEN, L. E. Approximating the permanent: A simple approach. *Random Structures & Algorithms 5*, 2 (1994), 349–362.

[89] RASMUSSEN, L. E. Approximately counting cliques. *Random Structures & Algorithms 11*, 4 (1997), 395–411.

[90] RIORDAN, O. Spanning subgraphs of random graphs. *Combinatorics, Probability & Computing 9*, 2 (2000), 125–148.

[91] RUPPERT, J., AND SEIDEL, R. Approximating the d-dimensional complete euclidean graph. In *CCCG '91* (1991), pp. 207–210.

[92] SANTI, P. Topology control in wireless ad hoc and sensor networks. *CSURV: Computing Surveys 37* (2005).

[93] SCHIEBER, B., AND VISHKIN, U. On finding lowest common ancestors: Simplification and parallelization. *SIAM Journal on Computing 17*, 6 (1988), 1253–1262.

[94] SHARIR, M. Intersection and closest-pair problems for a set of planar discs. *SIAM Journal on Computing 14* (1985), 448–468.

[95] SIPSER, M. *Introduction to the Theory of Computation*. PWS, 1996.

[96] SLOANE, N. *The On-Line Encyclopedia of Inetger Sequences*. published electronically at http://www.research.att.com/njas/sequences/, 1996-2004.

[97] SMID, M. The well-separated pair decomposition and its applications. In *Handbook on Approximation Algorithms and Metaheuristics*, T. Gonzalez, Ed. Chapman & Hall/CRC, 2007, pp. 53–1 – 53–12.

[98] SMITH, W. D., AND WORMALD, N. C. Geometric separator theorems & applications. In *FOCS '98* (1998), IEEE, pp. 232–243.

[99] SRINIVAS, A., AND MODIANO, E. Minimum energy disjoint path routing in wireless ad-hoc networks. In *MOBICOM '03* (2003), ACM, pp. 122–133.

[100] STAHS, T., AND WAHL, F. M. Recognition of polyhedral objects under perspective views. *Computers and Artificial Intelligence 11* (1992), 155–172.

[101] STEIN, C., AND KARGER, D. R. A new approach to the minimum cut problem. *Journal of the ACM 43*, 4 (1996), 601–640.

[102] STRANG, G. *Introduction to Linear Algebra*. Wellesley-Cambridge Press, 1993.

[103] THORUP, M., AND ZWICK, U. Approximate distance oracles. *Journal of ACM 52*, 1 (2005), 1–24.

[104] TODA, S. On the computational power of PP and $\oplus P$. In *FOCS* (1989), IEEE, pp. 514–519.

[105] VALIANT, L. G. The complexity of computing the permanent. *Theoretical Computer Science 8* (1979), 189–201.

[106] VIGODA, E. A note on the glauber dynamics for sampling independent sets. *Electronic Journal of Combinatorics 8*, 1 (2001).

[107] WANG, Y., AND LI, X.-Y. Localized construction of bounded degree and planar spanner for wireless ad hoc networks. *MONET 11*, 2 (2006), 161–175.

[108] WANG, Y., LI, X.-Y., AND FRIEDER, O. Distributed spanners with bounded degree for wireless ad hoc networks. *Int. J. Found. Comput. Sci 14*, 2 (2003), 183–200.

[109] WASSERMAN, L. *All of Statistics: A Concise Course in Statistical Inference*. Springer, 2005.

[110] YAO, A. C.-C. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM Journal on Computing 11*, 4 (Nov. 1982), 721–736.

[111] ZWICK, U. Exact and approximate distances in graphs - A survey. In *ESA '01* (2001), vol. 2161, Springer, pp. 33–48.

Vita

Shiva Prasad Kasiviswanathan

346D IST Building
Department of Computer Science and Engineering
Pennsylvania State University
University Park, PA 16802

Phone: +1-(814)-404-6946
Fax: +1-(814)-865-3176
kasivisw@cse.psu.edu
www.cse.psu.edu/~kasivisw

RESEARCH INTERESTS

Approximation Algorithms, Combinatorial Optimization, Database Privacy and its connections to Machine Learning theory, Information theory, and Statistics.

EDUCATION

Ph.D. Candidate, Computer Science and Engineering 2003 - 2008
Pennsylvania State University, University Park, PA, USA.

Bachelor of Engineering, Computer Science and Engineering 1999 - 2003
Sri Venkateswara College of Engineering, University of Madras, India.

WORK EXPERIENCE

Theory Group, Pennsylvania State University Fall 2003 - 2005, Fall 2007-current
Research Assistant, supervised by Prof. Martin Fürer.
Research in algorithms, computational complexity, computational geometry.

CCS-5 Group, Los Alamos National Lab Summers of 2005, 2006
Graduate Research Intern, supervised by Dr. Anders Hansson and Dr. Gabriel Istrate.
Research in combinatorics, social networks.

Pennsylvania State University Spring 2005 - Spring 2007
Teaching Assistant for CSE 430 Project Design.
Guided around 30 senior year students through the process of constructing an autonomous robot.

Theory Group, Pennsylvania State University Summer 2007
Research Assistant, supervised by Prof. Sofya Raskhodnikova.
Research in database privacy, property testing.

SELECTED PUBLICATIONS

[1] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, A. Smith. What Can We Learn Privately? In FOCS, 2008.

[2] M. Fürer, S. P. Kasiviswanathan. Approximately Counting Embeddings into Random Graphs. In RANDOM, 2008.