Constant Time Algorithms in Sparse Graph Model

by

Huy Ngoc Nguyen

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2010

Author .....
Department of Electrical Engineering and Computer Science
August 6, 2010

Certified by .....
Alan Edelman
Professor in Applied Mathematics
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Constant Time Algorithms in Sparse Graph Model

by

Huy Ngoc Nguyen

Submitted to the Department of Electrical Engineering and Computer Science
on August 6, 2010, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science

Abstract

We focus on constant-time algorithms for graph problems in bounded degree model. We introduce several techniques to design constant-time approximation algorithms for problems such as Vertex Cover, Maximum Matching, Maximum Weighted Matching, Maximum Independent Set and Set Cover. Some of our techniques can also be applied to design constant-time testers for minor-closed properties.

In Chapter 1, we show how to construct a simple oracle that provides query access to a *fixed* Maximal Independent Set (MIS) of the input graph. More specifically, the oracle gives answers to queries of the form "Is v in the MIS?" for any vertex v in the graph. The oracle runs in constant-time, i.e., the running time for the oracle to answer a single query, is independent to the size of the input graph. Combining this oracle with a simple sampling scheme immediately implies an approximation algorithm for size of the minimum vertex cover.

The second technique, called *oracle hierarchy*, transforms classical approximation algorithms into constant-time algorithms that approximate the size of the optimal solution. The technique is applicable to a certain subclass of algorithms that compute a solution in a constant number of phases. In the transformation, *oracle hierarchy* uses the MIS oracle to simulates each phase.

The problems amenable to these techniques include Maximum Matching, Maximum Weight Matching, Set Cover, and Minimum Dominating Set. For example, for Maximum Matching, we give the first constant-time algorithm that for the class of graphs of degree bounded by d, computes the maximum matching size to within εn, for any $\varepsilon > 0$, where n is the number of vertices in the graph. The running time of the algorithm is independent of n, and only depends on d and ε.

In Chapter 2, we introduce a new tool called *partitioning oracle* which provides query access to a *fixed* partition of the input graph. In particular, the oracle gives answers to queries of the form "Which part in the fixed partition contains v?" for any vertex v in the graph. We develop methods for constructing a partitioning oracle for any class of bounded-degree graphs with an excluded minor. For any $\varepsilon > 0$, our partitioning oracle provides query access to a fixed partition of the input constant-degree minor-free graph, in which every part has size $O(1/\varepsilon^2)$, and the number of

edges removed is at most εn.

We illustrate the power of this technique by using it to extend and simplify a number of previous approximation and testing results for sparse graphs, as well as to provide new results that were unachievable with existing techniques. For instance:

- We give constant-time approximation algorithms for the size of the minimum vertex cover, the minimum dominating set, and the maximum independent set for any class of graphs with an excluded minor.
- We show a simple proof that any minor-closed graph property is testable in constant time in the bounded degree model.

Finally, in Chapter 3, we construct a more efficient partitioning oracle for graphs with constant treewidth. Although the partitioning oracle in Chapter 2 runs in time independent of the size of the input graph, it has to make $2^{\text{poly}(1/\varepsilon)}$) queries to the input graph to answer a query about the partition. Our new partitioning oracle improves this query complexity to $\text{poly}(1/\varepsilon)$ for graphs with constant treewidth.

The new oracle can be used to test constant treewidth in $\text{poly}(1/\varepsilon)$ time in the bounded-degree model. Another application is a $\text{poly}(1/\varepsilon)$-time algorithm that approximates the maximum matching size, the minimum vertex cover size, and the minimum dominating set size up to an additive εn in bounded treewidth graphs.

Thesis Supervisor: Alan Edelman
Title: Professor in Applied Mathematics

Acknowledgments

There are many people that I would like to thank for helping me getting through the years of my graduate studies.

I can not express enough gratitude to my advisor, Prof. Alan Edelman, for what he has done for me over the last 5 years. He gave me freedom to pursue my research, taught me the skills in approaching scientific problems, listened patiently to my ideas and offered insightful feedbacks. Without his indispensable advice, patience, and support, this thesis would not have been at all possible.

I would like to thank Ronitt Rubinfeld who taught me sublinear time algorithms and got me started on my first project in this field. I also thank her and Devavrat Shah for agreeing to be my thesis readers. I thank Jeremy Kepner for the research assistant opportunities in Spring and Summer 2009, and his enthusiasm in teaching me graph modeling and visualization.

I wish to thank Krzyzstof Onak for his collaboration on most of the results in this thesis. Working with Krzyzstof was a great experience for me. I learnt a lot from his wisdom, sharpness, and professional work ethics.

I thank my other co-authors Avinatan Hassidim, Johnathan Kelner, Huy L. Nguyen, Khanh DoBa for their collaboration and meaningful discussions which helped shape the research topics in this thesis.

Finally, thanks to my dear parents for their love and support, and to my beloved wife Anh for always being by myside all these years and to my beautiful son Mun for his inspiration.

Contents

Introduction

In recent years, the explosive growth of the Internet and digital media created an enormous amount of data and completely changed the concept of efficiency in algorithmic setting. Traditionally, algorithms that require linear time and/or space are generally considered to be highly efficient; in some settings, even polynomial time and space requirements are acceptable. However, today this is often no longer the case. For examples, in the graph mining area, as the Web graph has grown to the order of hundred billions of nodes and edges, even reading that graph would take a lot of time and computation power, while most of the real-time web mining applications require instant response. In these cases, algorithms that run in sublinear time or even constant time have become crucial.

Sublinear-Time Algorithms and Bounded Degree Model

The line of research on *sublinear time algorithms* was initiated to deal with massive datasets that occur more and more frequently in various contexts. The main goal in this area is to design algorithms that only read and process a small portion of input data and produce correct output with high probability. Blum, Luby and Rubinfeld [6] were the first to take this approach to design a fast testing algorithm for the linearity of functions. Since then, research in sublinear time algorithms has been expanded for many other optimization problems in graph theory, statistic, algebraic computations, geometry and computer graphics. See the surveys [16, 20, 42, 41] for detailed results and references.

In this thesis, we focus on sublinear time algorithms for graph problems in the

bounded degree model. The bounded degree model is introduced by Goldreich and Ron [21] as a natural model for sparse graphs. The model assumes that the input graphs have maximum degree bounded by a constant, and algorithms can access the graphs through queries to the adjacency lists. Formally, given an input graph $G = (V, E)$ with degree bounded by a constant d, there is a query function $f_G : V \times [d] \to V$, where for each $v \in V$ and $k \in [d]$, the function $f_G(v, k)$ returns the k^{th} neighbor of v in the adjacency list. We also assume that the computation time of $f_G(v, k)$, for all $v \in V$ and $k \in [d]$, is constant.

Given an algorithm $\mathcal{A}$ in bounded degree model, the *query complexity* of $\mathcal{A}$ is the maximum number of queries made by $\mathcal{A}$ to the input graph. Also, the conventional *time complexity* of $\mathcal{A}$ refers to maximum running time of $\mathcal{A}$. Algorithm $\mathcal{A}$ is said to run in *constant time* if its time complexity is independent to the size of the input graph.

Approximation Algorithms. We define the notion of approximation that we use. This notion combines additive and multiplicative approximation, and appears in theoretical computer science in many contexts.

Definition 1 *We say that a value $\hat{y}$ is an* (α, β)-approximation *to y, if*

$$y \leq \hat{y} \leq \alpha \cdot y + \beta.$$

We say that an algorithm A is an (α, β)-approximation algorithm *for a value $V(x)$ if it computes an (α, β)-approximation to $V(x)$ with probability at least $2/3$ for any proper input x.*

In most problems considered by us, under very natural assumptions, this notion suffices to get a multiplicative approximation. For instance, we later show a $(1, \varepsilon n)$-approximation algorithm for the maximum matching size for graphs of maximum degree at most d. If there are $\Omega(n)$ edges in the graph, the maximum matching size is at least $\Omega(n/d)$. Hence, by picking $\varepsilon = \Theta(\varepsilon'/d)$, we can get a multiplicative $(1 + \varepsilon')$-approximation to the maximum matching size.

Property Testing. Given a graph $G = (V, E)$ with maximum degree bounded by $d \geq 2$ and a combinatorial property $\mathcal{P}$ (e.g., planarity, connectivity, cycle-freeness). The graph G is said to be ϵ-*far* from statisfying $\mathcal{P}$ if one has to insert/delete at least $\epsilon d|V|$ edges in G in order to turn it into a graph satisfying $\mathcal{P}$. The meta question in the area of property testing in bounded degree model is the following: Given a graph $G = (V, E)$ with maximum degree bounded by d and a combinatorial property $\mathcal{P}$, distinguish between the case that G satisfies $\mathcal{P}$ and the case that G is ϵ-far from satisfying $\mathcal{P}$. An algorithm for a property testing problem, which is called a *property tester*, is required to output *accept* with high probability if G satisfies $\mathcal{P}$ and output *reject* if G is ϵ-far from satisfying $\mathcal{P}$. In the borderline case, where the the graph is neither ϵ-far nor satisfying $\mathcal{P}$, the tester is allowed to output arbitrarily.

It should also be noted that in addition to the bounded degree model, there is also another well-studied model for sublinear time algorithms in graph theory called the *adjacency matrix model*. This model assumes that the input graphs are dense, i.e., the number of edges is $\Omega(n^2)$ where n is the number of vertices. In this model, algorithms are allowed to access to the input graphs through the queries to the adjacency matrices. The main reason that we choose the bounded degree model over the adjacency matrix model is that we believe most of the interesting real world graphs are sparse, and thus, fit better to the bounded degree model.

Constant-Time Approximation Algorithms via Local Improvements (Chapter 1)

We present two generic techniques to design constant-time approximation algorithms for the Vertex Cover, the Maximum Matching, the Maximum Weighted Matching and the Set Cover problems. The material covered in this chapter is based on the joint work with Krzyzstof Onak [34].

Constant-time Oracle for Maximal Independent Set We construct a simple oracle which gives query access to a fixed maximal independent set (MIS) of the input

graph. More specifically, the oracle *selects* a fixed MIS of the graph and answers any query of the following form: *"Is a vertex v in the MIS?"* for any vertex v. Also, the oracle is *consistent*. That is, once the oracle answers a query about the MIS, all subsequent answers must also refer to the same MIS. In addition, for each query about the MIS, it only takes the oracle constant time to come up with the answer. Finally, in the construction of the oracle, no pre-computation is required.

Marko and Ron [32] were the first who design an oracle for MIS. They achieved an oracle with similar properties to ours, except that the query complexity of their oracle has worse dependency in ε, but better in d.

Oracle Hierarchy and Locality Lemma We introduce the oracle hierarchy technique which can be used to construct more sophisticated oracles from simpler ones. In particular, the technique shows how to take a classical approximation algorithm and turn it into an oracle that provides query access to an approximate solution (to the problem being considered). In order for the technique to work, the approximation algorithms must run in a constant number of phases such that each phase is an application of a maximal set of local improvements.

It is interesting to note that in order to simulate a phase in the approximation algorithm, the oracle hierarchy technique uses a MIS oracle (which can be either our MIS oracle or the MIS oracle by Marko and Ron) to find a maximal set of local improvements.

Using oracle hierarchy, we show how to construct oracles that give query access to approximations of the Maximum Matching, the Maximum Weighted Matching and the Set Cover. In order to show that these oracles run in constant time, we also state and prove the Locality Lemma which is a tool to bound the query complexity for these oracles.

Applications on Approximation Algorithms We show that our techniques can be applied to obtain constant-time approximation algorithms for the Vertex Cover, the Maximum Matching, the Maximum Weighted Matching and the Set Cover in

bounded degree model.

- **Vertex Cover.** We show that there exists a $(2, \varepsilon n)$-approximation algorithm of query complexity $2^{O(d)}/\varepsilon^2$, for graphs of maximum degree bounded by d. Combining the results of Parnas and Ron [35] and Marko and Ron [32] yields a $(2, \varepsilon n)$-approximation algorithm for the minimum vertex cover size of running time and query complexity $d^{O(\log(d/\varepsilon))}$. Our algorithm has better dependency on ε, but worse on d. Furthermore, Trevisan showed that for any constant $c \in [1, 2)$, a $(c, \varepsilon n)$-approximation algorithm must use at least $\Omega(\sqrt{n})$ queries (the result appeared in [35]).

- **Maximum Matching.** The only results on approximation of the maximum matching size in sublinear time that have been known before are the algorithms of Parnas and Ron [35]. Since their algorithms give a constant factor approximation to the minimum vertex cover size, they also give a constant factor approximation to the maximum matching size. The main obstacle to applying the general reduction of Parnas and Ron from distributed algorithms is that the known distributed algorithms [12, 13] for maximum matching run in a number of rounds that is polylogarithmic in the graph size, not constant.

  We show that nevertheless, there exists a $(1, \varepsilon n)$-approximation algorithm for the maximum matching for graphs of maximum degree bounded by $d \geq 2$ with query complexity $2^{d^{O(1/\varepsilon)}}$.

- **Maximum Weighted Matching.** For bounded-degree weighted graphs of all weights in $[0, 1]$, one can also show an algorithm that computes a $(1, \varepsilon n)$-approximation to the maximum weight matching with a number of queries that only depends on d and ε. We show that there exists a $(1, \varepsilon n)$-approximation algorithm for graphs of maximum degree bounded by $d \geq 2$, and edge weight is in $[0, 1]$, with query complexity $2^{2^{\tilde{O}(1/\varepsilon^2 + \log d/\varepsilon)}}$.

- **Set Cover.** Let $H(i)$ be the i-th harmonic number $\sum_{1 \leq j \leq i} 1/j$. Recall that $H(i) \leq 1 + \ln i$. We show that there is an $(H(s), \varepsilon n)$-approximation algo-

rithm of query complexity $\left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}$ for the minimum set cover size, for instances with n sets S_i, each of size at most s, and with each element in at most t different sets. As a special case of the set cover problem, we get an $(H(d+1), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{d^8}}{\varepsilon}\right)^{O(2^d)}$ for the minimum dominating set size for graphs of degree bounded by d.

Combining the results of Parnas and Ron [35] and Kuhn, Moscibroda and Wattenhofer [28] yields an $(O(\log d), \varepsilon n)$-approximation algorithm for the minimum dominating set of query complexity $d^{O(\log d)}/\varepsilon^2$.

Constant-Time Partitioning Oracle for Minor-Free Graphs and Applications (chapter 2)

Solving or even finding a good approximate solution to combinatorial graph problems (such as minimum vertex cover, maximum independent set) are typically NP-hard (see [27], [18] and [47])[1]. However, for planar graphs or graphs with an excluded minor, there exist polynomial-time approximation schemes for these problems due to the separator theorem by Lipton and Tarjan [29], Alon, Seymour and Thomas [1]. In Chapter 2, we present a new technique that takes advantage of the separator theorem to design constant-time approximation algorithms for minor-free graphs and planar graphs. Our technique can also be applied to design property testers for minor-closed properties. The results presented in this chapter are based on the joint work with Jonathan Kelner, Avinatan Hassidim and Krzyzstof Onak.

Partitioning Oracle We introduce the *partitioning oracle* that provides query access to a fixed partition of the input graph. In particular, given an input graph G, a partitioning oracle $\mathcal{O}$ for G allows any algorithm to pick an arbitrary vertex v in G and make a query to $\mathcal{O}$ of the form "What is the part in the partition that contains v?". In response, $\mathcal{O}$ makes some queries to G and outputs the part that contains v in

[1]Note that the approximation algorithms for the Minimum Vertex Cover in Chapter 2 can only achieve multiplicative approximation factor of 2.

the partition. In addition, for each query about the partition, $\mathcal{O}$ only needs to make a constant number of queries to the input graph.

Since our goal is to design constant-time algorithms, we are only interested in partitions in which every component has constant size and the number of edges removed is a small fraction of the graph size. However, in general, not all graphs have such partition. Therefore, we restrict our consideration to the family of *hyperfinite* graphs which is essentially the largest family of graphs that satisfies this property. A formal definitions follows.

Definition 2

- *Let $G = (V, E)$ be a graph. G is* (ε, k)-hyperfinite *if it is possible to remove $\varepsilon|V|$ edges of the graph such that the remaining graph has connected components of size at most k.*
- *Let ρ be a function from $\mathbb{R}_+$ to $\mathbb{R}_+$. A graph G is* ρ-hyperfinite *if for every $\varepsilon > 0$, G is $(\varepsilon, \rho(\varepsilon))$-hyperfinite.*
- *Let $\mathcal{C}$ be a family of graphs. $\mathcal{C}$ is* ρ-hyperfinite *if every graph in $\mathcal{C}$ is ρ-hyperfinite.*

Examples of bounded-degree hyperfinite families of graphs include bounded-degree graphs with an excluded minor [1] (for instance, bounded-degree planar graphs, bounded-degree graphs with constant tree-width), bounded-degree graphs of subexponential growth[2] [15], and the family of non-expanding bounded-degree graphs considered by Czumaj, Shapira, and Sohler [10].

We show that for this family of graphs, there exists a partitioning oracle for every $(\varepsilon, \rho(\varepsilon))$-hyperfinite graph that only requires $2^{d^{O(\rho(\varepsilon^3/3456000))}}/\varepsilon$ queries to the input graph to answer a query about the partition. In case $\rho(\varepsilon) = \text{poly}(1/\varepsilon)$ (which is always true for minor-free graphs), we construct a more efficient partitioning oracle that only requires $2^{\text{poly}(\varepsilon,d)}$ queries to answer a query to the oracle.

[2]The *growth* of a graph or a family of graphs is a function $g : \mathbb{Z}_+ \to \mathbb{Z}_+$ such that $g(d)$ equals the maximum number of vertices at distance at most d from any vertex d.

Applications in Approximation Algorithms We show that in case the input graph is hyperfinite with $\rho(\varepsilon) = \text{poly}(1/\varepsilon)$, there are constant-time $(1, \varepsilon n)$-approximation algorithms for minimum vertex cover, minimum dominating set, and maximum independent set whose running time is $2^{\text{poly}(1/\varepsilon)}$. Note that finding algorithms of running time $2^{(1/\varepsilon)^{o(1)}}$ is unlikely, since by setting $\varepsilon = 1/(3n)$, this would yield subexponential randomized algorithms for NP-hard problems. The above three problems are NP-hard for planar graphs, even with degree bounded by 3 [17, 18].

Applications in Property Testing We say that a graph property[3] is *minor closed* if it is closed under removal of edges, removal of vertices, and edge contraction. Examples of minor-closed families of graphs include planar graphs, outerplanar graphs, graphs of genus bounded by a constant, graphs of tree-width bounded by a constant, and series-parallel graphs.

Goldreich and Ron [21] showed an $O(1/\varepsilon^3)$ tester for the property that the input graph is a forest, i.e., does not have K_3 as a minor. Until the breakthrough result of Benjamini, Schramm, and Shapira [5], who showed that any minor-closed property can be tested in constant time, this was the only minor-closed property that was known to be testable in constant time. However, the running time of the tester of Benjamini, Schramm, and Shapira is $2^{2^{2^{\text{poly}(1/\varepsilon)}}}$, and the analysis is quite involved. Using the partitioning oracle for minor-free graphs, we give a simple proof of their result, and present a tester that runs in $2^{\text{poly}(1/\varepsilon)}$ time.

Efficient Partitioning Oracle for Constant Treewidth Graphs

It is easy to note that although the query complexities of the partitioning oracles in Chapter 2 are independent of the graph size, they are exponential in $1/\varepsilon$ and d. A natural open question is whether it is possible to construct a more efficient partitioning oracle that runs in time $\text{poly}(1/\varepsilon, d)$. Such a partitioning oracle would make a great impact in both theory and practice. Unfortunately, we don't know

[3] In this context, all graph properties are defined for graphs with no labels and are therefore closed under permutation of vertices.

how to answer that question. However, in Chapter 3, we show that if we narrow the problem domain to the family of graphs with constant treewidth, we can build a partitioning oracle that only run in time $\text{poly}(1/\varepsilon, d)$.

The partitioning oracle immediately implies the following results.

- For every family of graphs with treewidth bounded by a constant, there is a randomized approximation algorithm that computes the minimum vertex size, the maximum matching size, and the minimum dominating set size up to an additive εn in $\text{poly}(1/\varepsilon)$ time, where n is the number of vertices in the graph. Using the partitioning oracle in Chapter 2 only yields an approximation algorithm that runs in $2^{\text{poly}(1/\varepsilon)}$ time.

  Moreover, this shows better approximation algorithms than those known for general graphs. This adds to long line of research on this kind of algorithms [35, 33, 34, 45, 22].

- For every minor-closed property with bounded treewidth, there is a $\text{poly}(1/\varepsilon)$-time tester in the bounded-degree model. This strictly improves on the recent result of Yoshida and Ito [44], who showed a $\text{poly}(1/\varepsilon)$-time tester for outerplanarity. Earlier it was known that cycle-freeness is testable in $\text{poly}(1/eps)$ time [21].

The results of this chapter are based on the joint work with Alan Edelman, Avinatan Hassidim and Krzyzstof Onak.

Chapter 1

Constant-Time Approximation Algorithms via Local Improvements

There has been an enormous amount of work on Maximum Matching, Vertex Cover, and Set Cover in the classical computation model, where the whole input is read. It is obviously not possible to compute a solution to them in time sublinear in input size, since an optimal solution may itself have linear size. Can one approximate just the optimal solution size in time sublinear in the input size? This and similar questions have been asked by several researchers for various optimization problems [4, 9, 11, 24, 35]. In particular, Parnas and Ron [35] asked this question for the minimum vertex cover problem. They discovered a connection to distributed algorithms that run in a constant number of rounds. For graphs of bounded maximum or average degree, the connection yields approximation algorithms of running time independent of the size of the graph. In this chapter, we show two general techniques that can be used to construct constant-time approximation algorithms. The techniques work for all problems that were considered by Parnas and Ron, but does not rely on distributed algorithms, and for Maximum Matching, it can be used to overcome limitations of the previously known distributed algorithms.

Algorithm 1: A simple global algorithm to find a Maximal Independent Set

```
1 foreach v ∈ V do
2     Let r_v be a random value in [0, 1].
3 M := ∅
4 foreach v ∈ V in increasing order of r_v do
5     if v does not have any neighbor in M then
6         M := M ∪ {v}
7 return M.
```

1.1 Oracle for Maximal Independent Set

Let $G = (V, E)$ be the input graph. Our oracle $\mathcal{O}$ given query access to G, provides query access to a fixed MIS in G of the form "Is q in the MIS?" for every vertex q of the graph.

We describe how to construct $\mathcal{O}$ as follows. First, we show a global greedy algorithm that compute a MIS for the input graph. Then, we show how to simulate the global algorithm locally. Finally, we show that the query complexity of the local simulation is independent to the size of the input graph.

An Global Algorithm. Consider the Algorithm 1 which computes a MIS for the input graph. The algorithm starts with assigning a random value in $[0, 1]$ to each vertex in the graph. Then, the algorithm considers the vertices in increasing order of these values. For each vertex v in the graph, if v does not have any neighbor that has already been in the set M, then v is added to M. When all vertices have been considered, the algorithm output M as a MIS.

It is clear from the implementation of Algorithm 1 that M is an maximal independent set of the input graph[1]. Note that, since the Steps 3-7 of Algorithm 1 are deterministic, the MIS M only depends on the input graph G and the random bits in vector r.

[1]In some literatures, the set M is called the lexicographically first maximal independent set with respect to the random order r for the input graph.

The Constant-time Algorithm. For each query vertex $q \in V$, the oracle first determines the set of vertices adjacent to q of numbers r_v smaller than that of q, and recursively checks if any of them is in the set M. If at least one of the adjacent vertices is in M, q is not added to M; otherwise, it is.

Note that, for each vertex $v \in V$, we can query the random value r_v assigned to v as follows. When we see v for the first time, we can obtain the value of r_v by generating a random value in $[0, 1]$ on the fly and store that value in the memory. Later, when we see v again, we can retrieve r_v by looking it up in the memory.

Query Complexity. It remains to bound the query complexity of the algorithm.

Lemma 3 *Let G be a graph such that each vertex is adjacent to at most d other vertices. For each vertex v, we independently select a random number $r(v)$ from the range $[0, 1]$. The expected query complexity of an algorithm that starts from a vertex u chosen independently of the values $r(v)$, and explores all paths $w_0 = u, w_1, \ldots, w_k$ such that $r(w_0) > r(w_1) > \ldots > r(w_k)$ is $2^{O(d)}$.*

Proof

Consider a path $P = (w_0 = u, w_1, \cdots, w_{k-1})$ of length k that starts from q. The probability that the r-value in P is decreasing, i.e., $r(w_0) > r(w_1) > \ldots > r(w_{k-1})$, is at most $\frac{1}{k!}$.

Also, since there are at most d^k paths of length k that starts from u. Therefore, the number of queries to the input graphs that the algorithm has to make, is at most

$$\sum_{k=1}^{\infty} \frac{d^k}{k!} = e^d = 2^{O(d)}$$

■

By Lemma 3 and the Markov's inequality, for any $\delta > 0$, with probability $1 - \delta$, the constant-time algorithm only makes at most $2^{O(d)}/\delta$ queries to the input graph. We summarize the construction of the MIS oracle in the following theorem.

Theorem 4 *Let $G = (V, E)$ be an undirected graph with maximum degree bounded by $d \geq 2$. There is an oracle $\mathcal{O}$ given query access to G and $\delta > 0$, provides query access to a function $f_G : V \to \{\text{YES}, \text{NO}\}$ of the following properties.*

1. *All* YES *vertices in G form a MIS. That is, for any $u, v \in V$ such that $f_G(u) = f_G(v) =$ YES, $(u, v) \notin E$ and for any $w \in V$ such that w does not have any* YES *neighbor, $f_G(w) =$ YES.*

2. *For each query about f_G, with probability $1 - \delta$, $\mathcal{O}$ only needs to make $2^{O(d)}/\delta$ queries to G to answer.*

3. *The function f_G only depends on the input graph and random bits. In particular, it does not depend on the order of queries to $\mathcal{O}$*

Approximating the Size of A Maximal Independent Set It follows from the Hoeffding bound that the size of the MIS M can be estimated with constant probability and additive error at most εn by checking for $O(1/\varepsilon^2)$ randomly chosen vertices v, if they are in M. Therefore, there exists an $(1, \varepsilon)$-approximation algorithm for the size of a MIS of the input graph which only requires at most $2^{O(d)}/\varepsilon^2)$ queries to the input graph.

1.2 Oracle Hierarchy and Locality Lemma

We describe the general idea of how to construct more sophisticated oracles from simpler ones using oracle hierarchy. The applications of this technique will be described in detail in Section 1.4, 1.5, and 1.6. We also prove the Locality Lemma which can be used as a tool to bound the query complexity of oracles constructed by this technique.

1.2.1 Oracle Hierarchy

Observe that for problems like Maximum Matching, Maximum Weighted Matching or Set Cover, there are approximation algorithms which run in a constant number of phases such that each phase is an application of any maximal set of disjoint local

improvements. In addition, each local improvement considered in a given phase intersect with at most a constant number of other considered local improvements (see Section 1.4, 1.5, 1.6 for the algorithms).

The general idea behind the construction of oracles for these problems is the following. Let k be the number of phases in the approximation algorithm. We construct a sequence of oracles $\mathcal{O}_0, \mathcal{O}_1, \cdots, \mathcal{O}_k$ where, for $1 \leq i \leq k$, the oracle $\mathcal{O}_i$ implements query access to the intermediate solution constructed by the i-th phase of the algorithm. ($\mathcal{O}_0$ gives the initial solution that the algorithm starts with.) $\mathcal{O}_i$ is itself given query access to $\mathcal{O}_{i-1}$, and simulates the i-th phase of the algorithm on the output of the $(i-1)$-st phase. In order to find a maximal set of disjoint improvements, $\mathcal{O}_i$ uses the oracle in Section 1.1 (or the MIS oracle by Marko and Ron). Finally, $\mathcal{O}_k$ provides query access to a solution that the approximation algorithm could output.

1.2.2 Locality Lemma

We show that for the problems that we consider, the answer to a query about the output of the i-th phase can be computed, using in most cases only a small number of queries about the output of the $(i-1)$-st phase. We show that long chains of dependencies between prospective improvements in the i-th phase can be avoided by considering them in random order. Hence, it usually suffices to query a small neighborhood of each query point.

We now state and prove the Locality Lemma. Let us first give its informal explanation. We are given a graph of bounded degree with random numbers $r(v)$ in $[0,1]$ assigned independently to each node v of the graph. A function f is defined inductively on the nodes of the graph. The value of f at a node v is a function of only v and values of f at neighbors w of v such that $r(w) < r(v)$. The value of f at a node can be computed recursively. Suppose that we have an algorithm that does not know the numbers $r(v)$, and only wants to learn f at q different nodes. The lemma gives a bound which holds with high probability on the total number of nodes for which we must compute f in all recursive calls. A single phase of each of our algorithms can be expressed as this type of computation for a certain graph.

Lemma 5 (Locality Lemma) *Let $G = (V, E)$ be a graph of degree bounded by $d \geq 2$, and let $g : V \times (V \times A)^\star \to A$ be a function. A random number $r(v) \in [0, 1]$ is independently and uniformly assigned to each vertex v of G. A function $f_r : V \to A$ is defined recursively, using g. For each vertex v, we have*

$$f_r(v) = g(v, \{(w, f_r(w)) : r(w) < r(v)\}).$$

Let $\mathcal{A}$ be an algorithm that initially does not know r, but can adaptively query f_r at q different nodes. To answer $\mathcal{A}$'s queries, it suffices to recursively compute $f_r(v)$ for at most

$$\frac{q^2}{\delta} \cdot C^{d^4}$$

nodes v with probability at least $1 - \delta$, for any $\delta > 0$, where C is an absolute constant.

Proof Handling each query of $\mathcal{A}$ requires computing $f_r(v)$ for certain nodes v. Unfortunately, since $\mathcal{A}$'s queries may be adaptive, $\mathcal{A}$ may be able to deduce from answers to previous queries how to query f_r to increase the number of nodes for which the function must be recursively computed. Intuitively, if all $\mathcal{A}$'s query points were far away from each other, the sets of nodes explored to answer each query would likely be disjoint, and we could bound the number of nodes explored for each of them independently. Therefore, whenever we bound the number of $f_r(v)$ computed for $\mathcal{A}$'s query, we also make sure that there is no node w close to the query point such that computing $f_r(w)$ requires computing many $f_r(v)$.

We now give a few auxiliary definitions. We say that a node v can be *reached* or is *reachable* from a node w if there is a path $u_0 = w, u_1, \ldots, u_k = v$, such that $r(u_{i-1}) > r(u_i)$, for all $1 \leq i \leq k$. Less formally, v can be reached from w if we need to compute $f_r(v)$ in order to compute $f_r(w)$. The *reachability radius* of node v is the maximum distance between v and a node that is reachable from v.

What is the probability that for a given node v, the reachability radius is greater than t? The probability can be bounded by the probability that there is a path of length $t + 1$ that starts at v, and the values r are strictly decreasing along the path. There are at most $d \cdot (d - 1)^t$ such paths, and by symmetry, the probability that the

values r decrease along a fixed such path is $1/(t+2)!$ Hence the probability of the event is at most $\frac{d(d-1)^t}{(t+2)!}$.

We now consider not just a single query, but all the (potentially adaptive) queries constructed by $\mathcal{A}$. What is the probability that for each query $f_r(w)$, the reachability radius of w is at most t? After each query, $\mathcal{A}$ learns something about r, and may use this knowledge to create a malicious query. Note that it cannot learn more than $r(v)$ for nodes v that are either reachable from w or are a neighbor of a node reachable from w, where one of the queries was about $f_r(w)$. Suppose that the reachability radius for all previous query points was at most t. Let v be a vertex at distance greater than $2(t+1)$ from each of the previous query points. The probability that the reachability radius of v is at most t depends only on $r(u)$, for u at distance at most $t+1$ from v, and hence is independent of the algorithm's current knowledge. We only need to make sure that for a query about $f(v)$, for v at distance at most $2(t+1)$ to one of the previous query points, v has small reachability radius. This may depend on the knowledge of the algorithm. Hence, for any query point v, we also bound the reachability radius for all vertices at distance at most $2(t+1)$, so that the algorithm is unlikely to construct a query that requires exploring many vertices.

For each query point v, there are at most

$$1 + d\sum_{i=0}^{2t+1}(d-1)^i \leq 1 + d\sum_{i=0}^{2t+1} d^i \leq d^{2t+3}$$

vertices at distance at most $2(t+1)$. The total number of nodes close to one of the q query points is hence at most $q \cdot d^{2t+3}$. We want to bound the reachability radius by t for all of them. By the union bound, the probability that the algorithm queries some $f_r(v)$, where v has reachability radius greater than t is at most

$$qd^{2t+3} \cdot \frac{d(d-1)^t}{(t+2)!} \leq \frac{q \cdot d^{3t+4}}{(t+2)!} \leq q \cdot \left(\frac{3d^3}{t+2}\right)^{t+2}.$$

Let E_i be the event that the maximum reachability radius for all query points is exactly i, and $E_{>i}$ the event that it is greater than i. We have, $\Pr[E_{>i}] \leq q \cdot \left(\frac{3d^3}{i+2}\right)^{i+2}$.

What is the expected total number T of vertices for which we recursively compute f_r? It is

$$\begin{aligned} T &\leq \sum_{i\geq 0} \Pr[E_i] \cdot q(1 + d \cdot \sum_{0\leq j\leq i-1} (d-1)^j) \\ &\leq \sum_{i\geq 0} \Pr[E_i] \cdot qd^{i+1} \leq \sum_{i\geq 0} \Pr[E_{>i-1}] \cdot qd^{i+1} \\ &\leq \sum_{i\geq 0} q \left(\frac{3d^3}{i+1}\right)^{i+1} \cdot qd^{i+1} \leq q^2 \sum_{i\geq 0} \left(\frac{3d^4}{i+1}\right)^{i+1}. \end{aligned}$$

For $i \geq 6d^4 - 1$, we have

$$\sum_{i\geq 6d^4-1} \left(\frac{3d^4}{i+1}\right)^{i+1} \leq \sum_{i\geq 6d^4-1} 2^{-(i+1)} \leq 1.$$

Using calculus, one can show that the term $\left(\frac{3d^4}{i+1}\right)^{i+1}$ is maximized for $i + 1 = \frac{3d^4}{e}$, and hence,

$$\sum_{i<6d^4-1} \left(\frac{3d^4}{i+1}\right)^{i+1} \leq (6d^4 - 1) \cdot e^{\frac{3d^4}{e}}.$$

We get

$$\begin{aligned} T &\leq q^2 \left(1 + (6d^4 - 1) \cdot e^{\frac{3d^4}{e}}\right) \\ &\leq q^2 \cdot 6d^4 \cdot e^{\frac{3d^4}{e}} \leq q^2 C^{d^4}, \end{aligned}$$

for some constant C. By Markov's inequality, the probability that the number of queries is greater than T/δ is at most δ. Hence with probability at least $1 - \delta$, the number of queries is bounded by $q^2 C^{d^4}/\delta$. ■

Note that the Locality Lemma's upper bound is worse than the more specialized upper bound $2^{O(d)}$ in Lemma 3. The improved upper bound takes advantage of the fact that the algorithm consists of only one phase, and avoids dependencies between queries to oracles on lower levels.

1.3 Approximating the Size of the Minimum Vertex Cover

A constant-time $(2, \varepsilon n)$-approximation algorithm for the size of the minimum vertex cover seems to be the simplest application of the MIS oracle.

Consider the line graph of the input graph G. This graph has at most $dn/2$ vertices with maximum degree bounded by $2d$. Therefore, there exists an oracle $\mathcal{O}_1$ that given query access to G, provides query access to a fixed MIS M_1 of the line graph in G.

In addition, observe that a MIS of the line graph in G is a maximal matching of G. Gavril (see [19]) proved that the set of nodes matched in any maximal matching is a proper vertex cover of size at most 2 times the optimum. Furthermore, it is also well known that the size of a maximal matching is at least one half of the maximum matching size. Thus, we immediately obtain a $(2, \varepsilon n)$-approximation algorithm for the size of the minimum vertex cover.

Corollary 6 *There exists a $(2, \varepsilon n)$ approximation algorithm of query complexity $2^{O(d)}/\varepsilon^2$ for the minimum vertex cover size for graphs with maximum degree bounded by d.*

1.4 Approximating the Size of the Maximum Matching

1.4.1 Definitions and Notation

Let M be a *matching* in a graph $G = (V, E)$, that is, a subset of nonadjacent edges of G. A node v is *M-free* if v is not an endpoint of an edge in M. A path P is an *M-alternating* path if it consists of edges drawn alternately from M and from $E \setminus M$. A path P is an *M-augmenting* path if P is M-alternating and both endpoints of P are M-free nodes (i.e., $|P \cap M| = |P \cap (E \setminus M)| + 1$).

1.4.2 Properties of Matchings

Let $\oplus$ denote the symmetric difference of sets. If M is a matching and P is an M-augmenting path, then $M \oplus P$ is a matching such that $|M \oplus P| = |M| + 1$. Many matching algorithms search for augmenting paths until they construct a maximum matching, and one can show that in an non-maximum matching there is an augmenting path.

The correctness of our algorithm relies on the properties of matchings proven by Hopcroft and Karp [23]. The part of their contribution that is important to us is summarized below.

Fact 7 (Hopcroft and Karp [23]) *Let M be a matching with no augmenting paths of length smaller than t. Let $P^\star$ be a maximal set of vertex-disjoint M-augmenting paths of length t. Let A be the set of all edges in the paths in $P^\star$. There does not exist an $(M \oplus A)$-augmenting path of length smaller than or equal to t.*

We now prove an auxiliary lemma that connects the minimum length of an augmenting path and the quality of the matching.

Lemma 8 *Let M be a matching that has no augmenting paths of length smaller than $2t+1$. Let $M^\star$ be a maximum matching in the same graph. It holds $|M| \geq \frac{t}{t+1}|M^\star|$.*

Proof Consider the set of edges $\Delta = M \oplus M^\star$. There are exactly $|M^\star| - |M|$ more edges from $M^\star$ than from M in Δ. Since M and $M^\star$ are matchings, each vertex is incident to at most two edges in Δ. Hence Δ can be decomposed into paths and cycles. Each path of even length and each cycle contain the same number of edges from M and $M^\star$. Each path P of odd length contains one more edge from $M^\star$ than from M. It if contained one more edge from M, it would be an $M^\star$-augmenting path; an impossibility. P is then an M-augmenting path. Summarizing, we have exactly $|M^\star| - |M|$ odd-length vertex-disjoint paths in Δ, and each of them is an M-augmenting path.

Since each M-augmenting path has length at least $2t-1$, this implies that $|M| \geq t(|M^\star| - |M|)$. Hence, $|M| \geq \frac{t}{t+1}|M^\star|$. ■

1.4.3 The Algorithm

Consider the maximum matching problem in an unweighted graph of bounded degree d. It is well known that the size of any maximal matching is at least half of the maximum matching size. Because of that, we can obtain a $(2, \varepsilon n)$-approximation algorithm for the maximum matching size using the same argument as for the Vertex Cover (see Section 1.3). We now show that our technique can be used to achieve better approximations in constant time.

An Global Algorithm. We simulate the following global algorithm. The algorithm starts with an empty matching M_0. In the i-th phase, it constructs a matching M_i from M_{i-1} as follows. Let $P^\star_{i-1}$ be a maximal set of vertex-disjoint M_{i-1}-augmenting paths of length $2i-1$. Let A_{i-1} be the set of all edges in the augmenting paths in $P^\star_{i-1}$. We set $M_i = M_{i-1} \oplus A_{i-1}$. If M_{i-1} is a matching, so is M_i. By induction, all M_i are matchings. The algorithm stops for some k, and returns M_k.

We now show that M_i has no augmenting path of length smaller than $2i+1$. M_1 is a maximal matching, so it has no augmenting path of length smaller than 3. Now, for the inductive step, assume that M_{i-1}, $i \geq 1$, has no augmenting path shorter than $2i-1$. $P^\star_{i-1}$ is a maximal set of vertex-disjoint M_{i-1}-augmenting paths of length $2i-1$. Therefore, it follows by Fact 7 that M_i does not have any augmenting path shorter than $2i+1$.

Set $k = \lceil 1/\delta \rceil$, and let $M^\star$ be a maximum matching. By Lemma 8, $\frac{k}{k+1}|M^\star| \leq |M_k| \leq |M^\star|$, which yields $|M^\star| \leq \frac{k+1}{k}|M_k| \leq (1+\delta)|M^\star|$. If we had an estimate α such that $2|M_k| \leq \alpha \leq 2|M_k| + \varepsilon n/2$, we could get a $(1+\delta, \varepsilon n)$-approximation to $|M^\star|$ by multiplying α by $\frac{k+1}{2k}$, which is at most 1.

The Constant-Time Algorithm. We construct a sequence of oracles $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_k$. A query to $\mathcal{O}_i$ is an edge $e \in E$. The oracle's reply indicates whether e is in M_i. To compute the required α, it suffices to estimate the fraction of vertices that are matched in $|M_k|$. In order to do so, one can sample $O(1/\varepsilon^2)$ vertices, and for each of them, check if any incident edge is in M_k or not. The correctness of the estimate

with probability 5/6 follows from the Hoeffding bound.

The oracles $\mathcal{O}_i$ are constructed by using our technique for transforming algorithms into constant-time algorithms. $\mathcal{O}_i$ has access to $\mathcal{O}_{i-1}$, and simulates the i-th phase of the above algorithm. We assume that each M_{i-1}-augmenting path P of length $2i-1$ is assigned a random number $r(P)$, which is uniformly and independently chosen from $[0,1]$. These random numbers give a random ordering of all the M_{i-1}-augmenting paths. $P^\star_{i-1}$ is the greedily constructed maximal set of vertex-disjoint M_{i-1}-augmenting paths P considered in order of their $r(P)$. To handle a query about an edge e, the oracle first finds out if $e \in M_{i-1}$, and then, checks if there is an M_{i-1}-augmenting path in $P^\star_{i-1}$ that contains e. If there is such a path, the answer of $\mathcal{O}_i$ to the query about e is the opposite of the answer of $\mathcal{O}_{i-1}$. Otherwise, it remains the same.

The oracle can easily learn all length-$(2i-1)$ M_{i-1}-augmenting paths that contain e by querying G and O_{i-1}. To find out which augmenting paths are in $P^\star_{i-1}$, the oracle considers the following graph H_i. All the M_{i-1}-augmenting paths of length $2i-1$ are nodes of H_i. Two nodes P_1 and P_2 are connected in H_i if P_1 and P_2 share a vertex. To check if P is in $P^\star_{i-1}$, it suffices to check if any of the paths R corresponding to the vertices adjacent to P in H_i is in $P^\star_{i-1}$, for $r(R) < r(P)$. If none, $P \in P^\star_{i-1}$. Otherwise, P is not in $P^\star_{i-1}$. This procedure can be run recursively. This is basically how the MIS oracle works on input graph H_i. This finishes the description of the algorithm.

Query Complexity. It remains to bound the number of queries of the entire algorithm to the graph. This is accomplished in the following lemma.

Lemma 9 *The number of queries of the algorithm is with probability* 5/6 *of order* $\frac{2^{O(d^{9k})}}{\varepsilon^{2k+1}}$, *where* $k = \lceil 1/\delta \rceil$, *and* $d \geq 2$ *is a bound on the maximum degree of the input graph.*

Proof Our main algorithm queries $\mathcal{O}_k$ about edges adjacent to C'/ε^2 random vertices, where C' is a constant. Let $Q_{k+1} = C' \cdot d/\varepsilon^2$ be the number of the direct queries

of the main algorithm to G. These queries are necessary to learn the edges that $\mathcal{O}_k$ is queried with. Let Q_{i+1} be an upper bound on the number of queries of the algorithm to $\mathcal{O}_i$. We now show an upper bound Q_i on the number of queries to G performed by $\mathcal{O}_i$. The upper bound holds with probability at least $1 - \frac{1}{6k}$. Q_i also bounds the number of queries to O_{i-1}, since $\mathcal{O}_i$ does not query any edge it has not learnt about from G. For each received query about an edge e, $\mathcal{O}_i$ first learns all edges within the distance of $2i-1$ from e, and checks which of them are in M_{i-1}. For a single e, this can be done with at most $d \cdot 2\sum_{j=0}^{2i-2}(d-1)^j \leq 2d^{2i}$ queries to both G and $\mathcal{O}_{i-1}$, and suffices to find all length-$(2i-1)$ M_{i-1}-augmenting paths that contain e.

There are at most id^{2i-1} length-$(2i-1)$ paths in G that contain a fixed vertex v. Each such path can be broken into two paths that start at v. The length of the shorter is between 0 and $i-1$, and there are at most d^t paths of length t that start at t.

The number of length-$(2i-1)$ M_{i-1}-augmenting paths that contain e is therefore at most id^{2i-1}. Moreover, the maximum degree of H_i can be bounded by the number of length-$(2i-1)$ paths that intersect a given length-$(2i-1)$ augmenting path. Hence, the degree of H_i is at most $2i \cdot id^{2i-1} = 2i^2d^{2i-1}$. Finally, to find augmenting paths adjacent in H_i to a given augmenting path P, it suffices to learn whether e' is in M_{i-1}, for each edge e' within the radius of $2i$ from any of the vertices of P. This can be accomplished with at most $2i \cdot d\sum_{j=0}^{2i-1} d^j \leq 2id^{2i+1}$ queries to both G and $\mathcal{O}_{i-1}$.

In total, to answer queries about all, at most Q_{i+1} edges e, $\mathcal{O}_i$ must check membership in $P^\star_{i-1}$ for at most $Q_{i+1} \cdot 2id^{2i-1}$ augmenting paths. By the Locality Lemma, the number of augmenting paths for which we recursively check membership in $P^\star_{i-1}$ is with probability $1 - \frac{1}{6k}$ at most

$$(Q_{i+1} \cdot id^{2i-1})^2 \cdot C^{\left(2i^2d^{2i-1}\right)^4} \cdot 6k \leq 2^{O\left(d^{8i}\right)} \cdot kQ_{i+1}^2.$$

For each of them we compute all adjacent paths in H_i. Therefore, with probability $1 - \frac{1}{6k}$, the total number of $\mathcal{O}_i$'s queries to both $\mathcal{O}_{i-1}$ and G is bounded by

$$Q_{i+1} \cdot 2d^{2i} + 2^{O(d^{8i})} \cdot kQ_{i+1}^2 \cdot 2id^{2i+1}$$

$$\leq 2^{O(d^{8i})} \cdot kQ_{i+1}^2 =: Q_i.$$

The total number of queries to G in the entire algorithm can be bounded by

$$\sum_{j=1}^{k+1} Q_j \leq 2Q_1 \leq \left(\frac{C' \cdot d \cdot k}{\varepsilon^2}\right)^{2^k} \cdot 2^{O(d^{8k}) \cdot 2^k}$$

$$\leq \frac{2^{O(d^{9k})}}{\varepsilon^{2^{k+1}}}.$$

■

We summarize the whole algorithm in the following theorem.

Theorem 10 *There is a $(1+\delta, \varepsilon n)$-approximation algorithm for the maximum matching size that uses $\frac{2^{O(d^{9k})}}{\varepsilon^{2^{k+1}}}$ queries, where $d \geq 2$ is a bound on the maximum degree, and $k = \lceil 1/\delta \rceil$.*

Proof We run the algorithm described above. If the algorithm exceeds the number of queries guaranteed in Lemma 9, we terminate it, and return an arbitrary result. The algorithm returns a $(1 + \delta, \varepsilon n)$-approximation with probability at least $2/3$, because the sampling can return a wrong estimate with probability at most $1/6$, and the algorithm can exceed the allowed number of queries with probability at most $1/6$. ■

Finally, we can easily remove the multiplicative factor.

Corollary 11 *There is a $(1, \varepsilon n)$-approximation algorithm of query complexity $2^{d^{O(1/\varepsilon)}}$ for the maximum matching size, where $d \geq 2$ is a bound on the maximum degree.*

Proof Using the algorithm of Theorem 10, we can get a $(1+\varepsilon, \varepsilon n/2)$-approximation for the maximum matching size, using $2^{d^{O(1/\varepsilon)}}$ queries. Since the size of the maximum matching is at most $n/2$, this approximation is also a $(1, \varepsilon n)$-approximation for the maximum matching size. ■

1.5 Maximum Weight Matching

1.5.1 Definitions and notation

Given a weighted graph $G = (V, E)$ with bounded degree d and a weight function $w : E \to [0, 1]$. Let M be a matching in G, we write $w(M)$ to denote total weight of all M's matched edges. Let $M^\star$ be the matching with maximum weight, and let $\text{MWM}(G) = w(M^\star)$.

Similarly, let S be a collection of edges in G, we write $w(S)$ is the total weight of all edges in S, and write $g_M(S)$ to denote *gain* of S on matching M which is defined as $g_M(S) = w(S \cap (E \setminus M)) - w(S \cap M)$.

A simple path P is an M-augmenting path if P is M-alternating and $g_M(P) > 0$. In addition, we say a simple path P is $k^\star$-M-augmenting path if P is an M-augmenting path with at most k edges.

1.5.2 Existence of constant factor improvement

The correctness of our algorithm relies on the following lemma proven by Pettie and Sanders [36].

Lemma 12 (Pettie and Sanders [36]) *For any matching M and any constant k, there exists a collection of vertex-disjoint $k^\star$-augmentation such that $g_M(A) = w(M \oplus A) \geq w(M) + \frac{k+1}{2k+1}\left(\frac{k}{k+1}\text{MWM}(G) - w(M)\right)$*

Algorithm 2: A Global Algorithm for the Maximum Weight Matching

1 Remove all the edges of weight less than $\varepsilon/2$ in G. $k := 3/\varepsilon$

2 $L := 2\log(\varepsilon/3)/\log(\frac{6+3\varepsilon}{6+4\varepsilon})$

3 $W := \log 6/\varepsilon^2/\log(1+\varepsilon/3)$

4 $M := \emptyset$

5 **for** $i := 1, 2, \cdots, L$ **do**

6 | **for** $j := W, W-1, \cdots, 1$ **do**

7 | | $S^{\star}_{i,j} :=$ a maximal set of $k^{\star}$-M-augmenting paths with gain in the range $\left((1+\varepsilon/3)^{j-1}, (1+\varepsilon/3)^{j}\right]$

8 | | $M := M \oplus S^{\star}_{i,j}$

9 **return** M

A Global Algorithm. Consider Algorithm 2 that computes a matching for the input graph G. The algorithm starts by removing all edges of weight less than $\varepsilon/2$ in G. Then, it takes an empty matching M and augments it by repeatedly finding a maximal set of disjoint $k^{\star}$-M-augmenting paths and applying them to M. Note that, in each augmenting step, the algorithm only considers $k^{\star}$-M-augmenting paths of gain in a certain range, i.e., in every j-th iteration of the inner loop (Step 7-9), the algorithm only considers augmenting paths of gain in range $\left((1+\varepsilon/3)^{j-1}, (1+\varepsilon/3)^{j}\right]$.

We show that the matching M that Algorithm 2 computes, is an $(1+\varepsilon, \varepsilon n)$-approximation of the Maximum Weighted Matching.

Let M_i be the matching at the end of the i-th iteration of the outer loop (lines 3−6). Let $M_{i,j}$ be the matching at the end of the j-th inner loop iteration of i-th outer loop iteration. Let $S^{\star}_{i,j}$ be the set of $k^{\star}$-augmenting paths selected in the j-th inner loop iteration of the i-th outer loop iteration. Let $S^{\star}_{i}$ be the set of $k^{\star}$-augmenting paths selected in the i-th iteration, i.e., $S^{\star}_{i} = \bigcup_j S^{\star}_{i,j}$.

It is straightforward to check that, after removing all edges of weight less than $\varepsilon/2$, the total weight of the Maximum Weighted Matching in G only changes by at

most $\varepsilon n/2$

$$MWM(G_{\text{after Step 1}}) \geq MWM(G_{\text{before Step 1}}) - \varepsilon/2n$$

For simplicity, from now on, we will write G' to refer to the graph that remains after Step 1. By Lemma 12, for each $1 \leq i \leq L$, there exists a collection of vertex-disjoint $k^\star$-M_{i-1}-augmenting paths A_i such that

$$g_{M_{i-1}}(A_i) \geq \frac{k+1}{2k+1}\left(\frac{k}{k+1}\text{MWM}(G') - w(M_{i-1})\right)$$

Consider a $k^\star$-M_{i-1}-augmenting path P in A_i. There must exists an augmenting path $P' \in P_i^\star$ such that P' intersects P and the gain of P' is at least the gain of P divides by $(1+\varepsilon/3)$, i.e, $g(P') \geq g(P)/(1+\varepsilon/3)$. (Otherwise, the path P can be added to some set $P_{i,j}^\star$, and this contradicts to the fact that $P_{i,j}^\star$ is maximal.) In addition, each $k^\star$-augmenting path in $P_i^\star$ intersects with at most $k+1$ paths in A_i. Therefore, the total gain of all augmenting paths in $P_i^\star$ is at least $\frac{1}{(k+1)(1+\varepsilon/3)} g_{M_{i-1}}(A_i)$. Thus,

$$\begin{aligned}
w(M_i) - w(M_{i-1}) &\geq \frac{1}{(k+1)(1+\varepsilon/3)} g_{M_{i-1}}(A_i) \\
&\geq \frac{1}{(2k+1)(1+\varepsilon/3)}\left(\frac{k}{k+1}\text{MWM}(G') - w(M_{i-1}\right) \\
&\geq \frac{\varepsilon}{6+4/\varepsilon}\left((1-\varepsilon/3)MWM(G') - w(M_{i-1})\right)
\end{aligned}$$

Thus,

$$\begin{aligned}
w(M_L) &\geq (1 - \frac{6+3\varepsilon}{6+4\varepsilon})^L)(1-\varepsilon/3)\text{MWM}(G') \\
&\geq (1-\varepsilon)\text{MWM}(G) - \varepsilon n/2
\end{aligned}$$

Therefore, the matching M outputted by Algorithm 2 is a $(1+\varepsilon, \varepsilon n)$-approximation of the Maximum Weighted Matching in G.

1.5.3 The Constant-Time Algorithm.

Let we first describe how to construct an oracle for the matching M_L computed by the global algorithm.

We construct a sequence of oracles $\mathcal{O}_{1,1}$, $\mathcal{O}_{1,2}$, $\cdots$, $\mathcal{O}_{1,W}$, $\cdots$. $\mathcal{O}_{L,1}$, $\mathcal{O}_{L,2}$, $\cdots$, $\mathcal{O}_{L,W}$, each to simulate an (inner) iteration of the Algorithm 2. A query to $\mathcal{O}_{i,j}$ is an edge $e \in E$. The oracle's reply indicates whether e is in $M_{i,j}$. Given $1 \leq i \leq L$ and $1 \leq j \leq W$, we construct $\mathcal{O}_{i,j}$ as follows. First, $\mathcal{O}_{i,j}$ makes queries to $\mathcal{O}_{i,j-1}$ (or $\mathcal{O}_{i-1,W}$ if $j = 1$) and G' to learn about graph structure and matching around query edge e. Then, it checks if there is any $k^\star$-augmenting path in $P^\star_{i,j}$ that contains e. If yes, the answer of $\mathcal{O}_{i,j}$ is opposite to the answer of $\mathcal{O}_{i,j-1}$ (or $\mathcal{O}_{i-1,W}$ if $j = 1$). Otherwise, it stays the same.

The oracle $\mathcal{O}_{L,W}$ is the oracle that we want to construct. In other words, $\mathcal{O}_{L,W}$ provides the query access to the matching M_L computed by Algorithm 2.

Recall that M_L is a $(1+\varepsilon, \varepsilon n)$-approximation of the Maximal Weighted Matching in G. Therefore, if we had an estimate α such that $w(M_L) - \varepsilon n/4 \leq \alpha \leq w(M_L) + \varepsilon n/4$, then we could easily get a $(1+\varepsilon, \varepsilon n)$-approximation to MWM($G$) by subtracting $\varepsilon n/4$ from α. To compute the required α, it suffices to estimate the total weight of edges in M_L. In order to do so, we can sample $s = O(1/\varepsilon^2)$ vertices, and for each of them check if any incident edge is in M_L or not. If yes, add its weight to a sum T. Then, the value of α can be computed as $nT/2s$. The correctness of the estimate with probability $5/6$ follows from Hoeffding bound.

Query Complexity. The following lemma shows an upper bound on the number of queries made by our constant-time algorithm.

Lemma 13 *With probability* $5/6$, *the number of queries of the constant-time algorithm is of order* $\frac{W^{2k^2} 2^{O(d^{6k} 2^W)}}{\varepsilon^{2k+1}}$, *where* $k = \lceil 3/\varepsilon \rceil$, $W = \lceil \log(\varepsilon)/\log(1+\varepsilon/3) \rceil + 1$, *and* $d \geq 2$ *is a bound on the maximum degree of th input graph.*

Proof Our main algorithm queries $\mathcal{O}_{L,W}$ about the edges adjacent to C'/ε^2 random vertices, where C' is constant. Let $Q_{L+1,1} = C'd/\varepsilon^2$ be the number of the direct

queries of the constant-time algorithm to both G' and $\mathcal{O}_{L,W}$. Let $Q_{i,j+1}$ (or $Q_{i+1,1}$ in case $j = W$) be an upper bound on the number of queries of the algorithm to $\mathcal{O}_{i,j}$ and G'. We now show an upper bound $Q_{i,j}$ on the number of queries to both G' and $\mathcal{O}_{i,j-1}$ (or $\mathcal{O}_{i-1,W}$ in case $j = 1$) performed by $\mathcal{O}_{i,j}$. The upper bound holds with probability $\frac{1}{6LW}$.

Let $H_{i,j}$ be the intersection graph of augmenting paths in the $(i,j)^{th}$ iteration, i.e., all $k^\star$-$M_{i,j-1}$-augmenting paths with gain in range $\left((1+\varepsilon/3)^{j-1}, (1+\varepsilon/3)^{j}\right]$ are nodes in $H_{i,j}$ and two nodes P_1 and P_2 are connected if they share a vertex. The degree of $H_{i,j}$ is at most $(k+1)^2 d^{k+1}$.

In addition, for each edge e in G', the number of $k^\star$-augmenting paths that contain e is at most kd^k. Therefore, in total, to answer all $Q_{i,j+1}$ queries, $\mathcal{O}_{i,j}$ only need to check membership in $P^\star_{i,j}$ for at most $Q_{i,j+1} \cdot kd^k$ augmenting paths. By Locality Lemma, with probability $1 - \frac{1}{6LW}$, the number of augmenting paths that we have to recursively check membership is at most

$$(Q_{i,j+1} \cdot kd^k)^2 \cdot C^{((k+1)^2 d^{k+1})^4} \cdot 6LW = 2^{d^{O(1/\varepsilon)}} Q^2_{i,j+1}$$

(Note that, from Algorithm 2, $k := 3/\varepsilon = O(1/\varepsilon)$, $L := 2\log(\varepsilon/3)/\log(\frac{6+3\varepsilon}{6+4\varepsilon}) = \tilde{O}(1/\varepsilon)$ and $W := \log 6/\varepsilon^2/\log(1+\varepsilon/3) = \tilde{O}(1/\varepsilon)$.)

Now observe that for a given augmenting path P in $H_{i,j}$, it takes at most $(k+1)d^{k+1}$ queries to G' and $\mathcal{O}_{i,j-1}$ to compute the set of augmenting paths that intersect P.

Therefore, with probability $1 - \frac{1}{6LW}$, the total number of $\mathcal{O}_{i,j}$'s queries to both $\mathcal{O}_{i,j-1}$ and G' is bounded by

$$2^{d^{O(1/\varepsilon)}} Q^2_{i,j+1} \cdot (k+1)d^{k+1}$$

$$2^{d^{O(1/\varepsilon)}} Q^2_{i,j+1} =: Q_{i,j}$$

Therefore, the total number of queries to G in the entire algorithm can be bounded by

$$\begin{aligned}\sum_{i=1}^{L}\sum_{j=1}^{W} Q_{i,j} + Q_{L+1,1} \leq 2Q_{1,1} &= \left(\frac{C'}{\varepsilon^2}\right)^{2^{LW}} \cdot 2^{d^{1/\varepsilon}.2^{LW}} \\ &= \left(\frac{C'}{\varepsilon^2}\right)^{2^{\tilde{O}(1/\varepsilon^2)}} \cdot 2^{d^{1/\varepsilon}.2^{\tilde{O}(1/\varepsilon^2)}} \\ &= 2^{2^{\tilde{O}(1/\varepsilon^2+\log d/\varepsilon)}}\end{aligned}$$

■

We summarize our result in the following theorem.

Theorem 14 *There is a $(1+\varepsilon, \varepsilon n)$-approximation algorithm for the maximum weight matching that use $2^{2^{\tilde{O}(1/\varepsilon^2+\log d/\varepsilon)}}$ queries, where $d \geq 2$ is a bound on the maximum degree.*

We can also remove the multiplicative factor.

Corollary 15 *There is a $(1, \varepsilon n)$-approximation algorithm of query complexity $2^{2^{\tilde{O}(1/\varepsilon^2+\log d/\varepsilon)}}$ for the maximum weight matching, where $d \geq 2$ is a bound on the maximum degree.*

Proof Omitted. ■

1.6 Set Cover

In the minimum set cover problem, an input consists of subsets S_1 to S_n of $U = \{1, \ldots, m\}$. Each element of U belongs to at least one of the sets S_i. The goal is to cover U with the minimum number of sets S_i, that is, to find a minimum size set I of indices such that $\bigcup_{i \in I} S_i = U$. In this work, we want to approximate the optimal size of I.

We assume that for each set S_i, we have query access to a list of elements of S_i, and that for each element $u \in U$, we have query access to a list of indices of sets S_i that contain u.

1.6.1 The Classical Greedy Algorithm

Theorem 16 *There is an $(H(s), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{(st)^4}}{\varepsilon}\right)^{O(2^s)}$ for the minimum set cover size for instances with all n sets S_i of size at most s, and each element in at most t different sets.*

Proof We simulate the classical greedy algorithm [25, 31] for the set cover problem. The algorithm starts from an empty cover, and keeps adding the set S_i which covers most elements that have not yet been covered, until the whole set U is covered. The approximation factor of the algorithm is at most $H(s) \leq 1 + \ln s$.

We first consider all sets in random order and add to the cover those that cover s new elements at the time they are considered. Let $\mathcal{C}_1$ be the set of sets that were already included into the cover. We then consider the remaining sets, also in random order, and we add to the cover those that cover $s-1$ new elements. This way we get $\mathcal{C}_2$, the set of all sets already included in the cover. We keep doing this until we cover the whole set U, and $\mathcal{C}_s$ is the final cover. We show that in most cases one can check if a set is in the cover without simulating the whole process.

We create a sequence of oracles $\mathcal{O}_1$, $\mathcal{O}_2$, ..., $\mathcal{O}_s$ that correspond to the process described above. A query to an oracle $\mathcal{O}_j$ is an index i of a set S_i. The oracle's reply indicates whether S_i is in $\mathcal{C}_j$.

How is $\mathcal{O}_j$ implemented? We assume that each set S_i is assigned a random number r_{ji}, which is uniformly and independently chosen from $[0, 1]$. These random numbers give a random ordering of sets S_i. To handle a query about a set S_k, we first ask $\mathcal{O}_{j-1}$ if S_k was already included in $\mathcal{C}_{j-1}$. (For $j = 1$, we assume that $\mathcal{C}_{j-1} = \mathcal{C}_0 = \emptyset$, so no query is necessary in this case.) If S_k was already in $\mathcal{C}_{j-1}$, then it is also in $\mathcal{C}_j$. Otherwise, we learn first the elements of S_k (at most s queries) and what sets S_i they belong to (at most st further queries). Then, we check for each of these S_i if

it was already in $\mathcal{C}_{j-1}$ (at most st queries to $\mathcal{O}_{j-1}$), and for all of the S_i's that have $r_{ji} < r_{jk}$, we recursively check if they are in $\mathcal{C}_j$. Finally, using this knowledge, we can verify what number of elements of S_k is not yet covered, when S_k is considered. If the number of these elements is $s - j + 1$, then S_k is in $\mathcal{C}_j$. Otherwise, the number of the elements is lower than $s - j + 1$, and S_k is not in $\mathcal{C}_j$.

It is obvious that the above procedure simulates the classical greedy algorithm. Our sublinear approximation algorithm queries $\mathcal{O}_s$ for C'/ε^2 sets chosen at random, where C' is a sufficiently large constant, to estimate the fraction of sets which are in the cover to within $\varepsilon n/2$ with probability 5/6. By adding $\varepsilon n/2$ to the estimate, we get the desired approximation. We want to bound the number of queries. We set Q_s to $(C'/\varepsilon^2)^2 \cdot 6s \cdot C^{(st)^4}$, and define Q_j, for $1 \leq j \leq s-1$, to be $Q_j \leq (Q_{j+1} \cdot (st + 1))^2 \cdot 6s \cdot C^{(st)^4}$. By Lemma 5, each Q_i bounds the number of sets for which we check if they are in $\mathcal{C}_i$ with probability $1 - s \cdot \frac{1}{6s} = 1 - \frac{1}{6} = \frac{5}{6}$. It can be shown by induction that

$$Q_{s-i} = \left(\frac{6s \cdot (st+1) \cdot C^{(st)^4}}{\varepsilon} \right)^{O(2^i)} = \left(\frac{2^{(st)^4}}{\varepsilon} \right)^{O(2^i)}$$

with probability at least 5/6. So with probability 5/6, the total number of queries is at most

$$(s + st) \cdot \sum_{i=1}^{s} Q_s = \left(\frac{2^{(st)^4}}{\varepsilon} \right)^{O(2^s)}.$$

Summarizing, with probability 2/3, the algorithm uses the above number of queries, and returns the desired approximation. ■

1.6.2 Application to Dominating Set

In the dominating set problem, one is given a graph, and chooses a minimum size subset S of its vertices such that each vertex v of the graph is either in S or is adjacent to a vertex in S.

Theorem 17 *There is an $(H(d+1), \varepsilon n)$-approximation algorithm of query complexity $\left(\frac{2^{d^8}}{\varepsilon}\right)^{O(2^d)}$, for the minimum dominating set size for graphs with the maximum degree bounded by d.*

Proof The problem can be trivially expressed as an instance of the set cover problem. For each vertex v, we have a set S_v of size at most $d+1$ that consists of v and all neighbors of v. We want to cover the set of vertices of the graph with the minimum number of sets S_v. To approximate the minimum set cover size, we use the algorithm of Theorem 16. ■

1.6.3 Query Lower Bound Discussion

Trevisan proved that for every two positive constants γ and ϵ, there exists a constant d such that obtaining with constant probability a $(2-\gamma, \varepsilon n)$-approximation of the size of a minimum vertex cover of graphs over n vertices and degree d, requires $\Omega(\sqrt{n})$ queries [35]. Note that vertex cover is a special case of set cover, in which we want to cover the set of edges of the graph. For each vertex v, we create a set S_v that consists of edges incident to v. This implies that the above lower bound also applies to set cover, and shows that it is impossible to get a constant approximation factor less than 2 with the number of queries independent of the input size.

1.7 Remarks

Follow up on the work in this chapter, Yoshida, Yamamoto, and Ito [46] show that the query complexity of the MIS oracle can be improved exponentially by adding a simple pruning mechanism. In particular, they prove that the pruned MIS oracle has expected query complexity $O(d^2)$. In addition, they show a new technique to analyze the query complexities of approximation algorithms for the maximum matching, minimum vertex cover and set cover. For example, they show that the $(1, \varepsilon n)$-approximation algorithm for the maximum matching only requires at most $d^{O(\frac{1}{\varepsilon^2})}(\frac{1}{\varepsilon})^{O(\frac{1}{\varepsilon})}$ queries to the input graph.

Chapter 2

Constant-Time Partitioning Oracle for Minor-Free Graphs and Applications

Solving combinatorial graph problems (such as minimum vertex cover, maximum independent set, minimum dominating set) has been one of the main research goals of theoretical computer science. In the early 1970s, many of those problems unfortunately turned out to be as hard as the satisfiability problem, due to the breakthrough result of Karp ([27], see the survey [18]). In the 1990s, the discovery of the PCP theorem resulted in showing that even finding good approximate solutions is often NP-hard (see for instance [47]).

In spite of these negative results, multiple methods for finding good approximate solutions for several restricted families of graphs have been developed over the years. Notably, Lipton and Tarjan [29] proved the separator theorem for planar graphs, which resulted in polynomial-time approximation schemes for several combinatorial problems, which remain NP-hard even restricted to planar graphs [30]. The separator theorem was generalized to arbitrary graphs with an excluded minor by Alon, Seymour, and Thomas [1], and similar polynomial-time approximation schemes immediately followed.

An important implication of the separator theorem is that any graph with a fixed

excluded minor with maximum degree bounded by d can be partitioned into small components of size at most $\text{poly}(d, 1/\varepsilon)$ by removing only an ε-fraction of edges for any $\varepsilon > 0$. In this chapter, we develop techniques for locally computing such a partition for minor-free families of graphs and in general, for hyperfinite families of bounded-degree graphs (see Introduction for the formal definition of hyperfinite graphs). We construct a *partitioning oracle* that given query access to a graph from a specific family of graphs, provides query access to a fixed partition, and queries a fraction of the graph independent of the graph size. Just like knowing the entire partition is useful for finding a good approximate solution, our local version is useful for approximating the size of the optimal solution in time independent of the actual graph size. Our partitioning oracles also find applications to other approximating and testing problems that we describe in more detail below.

2.1 Preliminaries

Partitions. We say that P is a *partition* of a set S if it is a family of nonempty subsets of S such that $\bigcup_{X \in P} X = S$, and for all $X, Y \in P$ either $X = Y$ or $X \cap Y = \emptyset$. We write $P[q]$ to denote the set in P that contains an element $q \in S$.

Graph Minors. A graph H is a *minor* of a graph G, if H can be obtained from G by vertex removals, edges removals, and edge contractions. A graph is *H-minor free* if it does not have H as a minor. A graph property $\mathcal{P}$ is *minor-closed* if for every graph $G \in \mathcal{P}$, every minor of G also belongs to $\mathcal{P}$. The Robertson-Seymour theorem [40] says that every minor-closed property can be expressed via a constant number of excluded minors. Moreover, Robertson and Seymour [39] showed that for every minor H, there is a deterministic $O(n^3)$-time algorithm for checking if H is present in the input graph.

Lemma 18 (Proposition 4.1 in [1]) *For every graph H, there exists a constant C_H such that if G is an n-vertex H-minor free graph, then there exists a set S of at*

most $C_H \cdot n/\sqrt{t}$ vertices of G such that removing vertices of S leaves no connected component on more than t nodes ($t > 1$).

Corollary 19 *Let H be a fixed graph. There exists a constant $C_H > 1$ such that for every $\varepsilon \in (0,1)$, every H-minor free graph with degree bounded by d is $(\varepsilon dn, C_H^2/\varepsilon^2)$-hyperfinite.*

Proof Set t in Lemma 18 to $C_H^2/\varepsilon^2 > 1$. One can remove from G at most εn vertices so that each remaining connected component has at most C_H^2/ε^2 vertices. Since the degree of each vertex in G is bounded by d, it suffices to remove from G the edges incident to those vertices to achieve the same property. The number of these edges is at most εdn. ■

Vertex Cover. We write $\mathrm{VC}(G)$ to denote the minimum vertex cover size for a graph G.

Bounded Degree Subgraphs. Let G be a graph. We write $G|_k$, $k \in \mathbb{N}$ to denote G without the edges that are incident to vertices of degree higher than k in G. For a family of graphs $\mathcal{C}$, we define:

$$\mathcal{C}|_k = \{G|_k : G \in \mathcal{C}\}.$$

2.2 Partitioning Oracles and Their Applications

We now define the main tool that is used in this chapter. A partitioning oracle provides query access to a global partition of the graph into small components.

Definition 20 *We say that $\mathcal{O}$ is an* (ε, k)-partitioning oracle *for a family $\mathcal{C}$ of graphs if given query access to a graph $G = (V, E)$ in the bounded degree model, it provides query access to a partition P of V. For a query about $v \in V$, $\mathcal{O}$ returns $P[v]$. The partition has the following properties:*

- *P is a function of the graph and random bits of the oracle. In particular, it does not depend on the order of queries to $\mathcal{O}$.*
- *For every $v \in V$, $|P[v]| \leq k$ and $P[v]$ induces a connected graph in G.*
- *If G belongs to $\mathcal{C}$, then $|\{(v,w) \in E : P[v] \neq P[w]\}| \leq \varepsilon|V|$ with probability* 9/10.

The most important property of our oracles is that they compute answers in time independent of the graph size by using only local computation. We give two methods for constructing partitioning oracles for different classes of graphs. We briefly describe them below.

A Generic Partitioning Oracle for Hyperfinite Graphs (Section 2.3).

We give a generic oracle that works for *any* hyperfinite family of graphs.

Theorem 21 *Let G be an $(\varepsilon, \rho(\varepsilon))$-hyperfinite graph with degree bounded by $d \geq 2$. Suppose that the value $\rho(\varepsilon^3/3456000)$ is known, that is, it can either be efficiently computed, or is hard-wired for a given $\varepsilon > 0$ of interest. There is an $(\varepsilon d, \rho(\varepsilon^3/3456000))$-partitioning oracle with the following properties. The oracle answers every query, using $2^{d^{O(\rho(\varepsilon^3/3456000))}}/\varepsilon$ queries to the graph. If q is the total number of queries to the oracle, the total amount of the oracle's computation is bounded by $q \log q \cdot 2^{d^{O(\rho(\varepsilon^3/3456000))}}/\varepsilon$.*

The oracle's construction is based on locally simulating a greedy global partitioning procedure. The procedure repeatedly selects a random vertex v and tries to remove a small component containing it by cutting few edges. If there is no such component, only v is removed. This procedure can easily be simulated locally using a local computation paradigm in Chapter 1.

An Efficient Partitioning Oracle for Graphs with Small Components (Section 2.4).

Recall that a $(\varepsilon, \rho(\varepsilon))$-hyperfinite if it is possible to remove $\varepsilon|V|$ edges of the graph such that the remaining graph has connected components of size at most k. We now focus on ρ-hyperfinite families of graphs with $\rho(\varepsilon) = \text{poly}(1/\varepsilon)$. That is, for every $\varepsilon > 0$, one can remove at most εn edges of the graph to obtain small components of size $\text{poly}(1/\varepsilon)$. In this case, the generic oracle of Theorem 21 makes $2^{d^{\text{poly}(1/\varepsilon)}}$ queries to the graph to answer each query to the oracle. We give a more efficient oracle that makes $2^{\text{poly}(1/\varepsilon,d)}$ queries.

Theorem 22 *Let $R : \mathbb{R}^2 \to \mathbb{R}$ be a polynomial. Let $\mathcal{C}$ be a family of graphs such that, for every $d \in \mathbb{N}_+$, and every $\varepsilon \in (0,1)$, $\mathcal{C}|_d$ is $(\varepsilon, R(d,\varepsilon))$-hyperfinite. There is a uniform partitioning oracle that takes $d \in \mathbb{Z}_+$ and $\varepsilon \in (0,1)$ as input and acts as an $(\varepsilon, \text{poly}(1/\varepsilon, d))$-partitioning oracle for $\mathcal{C}|_d$. Let q be the number of queries to the oracle. The oracle makes $q \cdot 2^{\text{poly}(1/\varepsilon,d)}$ queries to the input graph and the total amount of the oracle's computation is $(q \log q) \cdot 2^{\text{poly}(1/\varepsilon,d)}$.*

Recall that every minor-free family of graphs is $(\varepsilon, \text{poly}(1/\varepsilon))$-hyperfinite (Corollary 19). Theorem 22 immediately implies a $(\varepsilon, \text{poly}(1/\varepsilon, d))$-partitioning oracle that makes $2^{\text{poly}(1/\varepsilon,d)}$ queries to the input graphs per query.

In addition, it is also possible to construct an efficient partitioning oracle for minor-free graphs based on a deterministic clustering method of Czygrinow, Hańćkowiak, and Wawrzyniak [14, Section 2]. The oracle only requires $d^{\text{poly}(1/\varepsilon)}$ queries to the input graphs per query to the oracle (for more details, see the full paper [34]). We also note that for graphs with polynomial growth, Jung and Shah [26] provide methods that can be used to construct a partitioning oracle that makes $\text{poly}(1/\varepsilon)$ queries to the graph for each query to the oracle.

2.2.1 Constant-Time Approximation Algorithms

We describe how to use partitioning oracles to design constant-time approximation algorithms for combinatorial problems for hypefinite families of graphs. As an example, consider the minimum vertex cover problem. We use a partitioning oracle to

obtain access to a partition of the input graph. The union of optimal vertex covers over all connected components constitutes a set of size within an additive $O(\varepsilon n)$ of the minimum vertex cover size of the original graph. By sampling $O(1/\varepsilon^2)$ vertices and computing the fraction of those that belong to the optimal vertex cover in their component, we obtain a $(1, O(\varepsilon n))$-approximation to the minimum vertex cover size of the original graph.

A formal theorem and proof follow. To achieve a good approximation, a bound on the average degree is needed. Note that every family of graphs with an excluded minor has average degree bounded by a constant.

Theorem 23 *Let $\mathcal{C}$ be a family of graphs with average degree bounded by $\tilde{d}$. Let $\varepsilon > 0$. Let $\mathcal{O}$ be an $(\varepsilon/3, k)$-partitioning oracle for the family $\mathcal{C}|_{3\tilde{d}/\varepsilon}$. There is a $(1, \varepsilon n)$-approximation algorithm for the minimum vertex cover size in any graph $G = (V, E)$ in $\mathcal{C}$. The algorithm*

- *gives $\mathcal{O}$ query access to the graph $G|_{3\tilde{d}/\varepsilon}$,*
- *makes $O(1/\varepsilon^2)$ uniformly distributed queries to $\mathcal{O}$,*
- *uses $2^{O(k)}/\varepsilon^2 + O(\tilde{d}k/\varepsilon^3)$ time for computation.*

The same holds for the maximum independent set problem, and the minimum dominating set problem.

Proof All edges from G missing in $G|_{3\tilde{d}/\varepsilon}$ can be covered by vertices of degree greater than $3\tilde{d}/\varepsilon$ in G. We write $G' = (V, E')$ to denote $G|_{3\tilde{d}/\varepsilon}$. Note that the number of such vertices is by Markov's inequality at most $\varepsilon n/3$. Therefore, we have

$$\mathrm{VC}(G) - \varepsilon n/3 \leq \mathrm{VC}(G') \leq \mathrm{VC}(G).$$

The adjacency list of every vertex v in G' can easily be computed in $O(3\tilde{d}/\varepsilon)$ time. If the degree of v is greater than $3\tilde{d}/\varepsilon$ in G, then v is an isolated vertex in G'. Otherwise, we go over the neighbors of v in G, and each neighbor w in G remains a neighbor in G' if and only if w has degree at most $3\tilde{d}/\varepsilon$ in G. We give $\mathcal{O}$ query access

to G'. With probability $9/10$, $\mathcal{O}$ provides query access to a partition P such that the number of edges $(v, w) \in E'$ with $P[v] \neq P[w]$ is at most $\varepsilon n/3$. Let $G'' = (V, E'')$ be G' with those edges removed. Since they can be covered with $\varepsilon n/3$ vertices, we have

$$\mathrm{VC}(G') - \varepsilon n/3 \leq \mathrm{VC}(G'') \leq \mathrm{VC}(G'),$$

that is,

$$\mathrm{VC}(G) - 2\varepsilon n/3 \leq \mathrm{VC}(G'') \leq \mathrm{VC}(G).$$

To get a $(1, \varepsilon n)$-approximation to $\mathrm{VC}(G)$, it suffices to estimate $\mathrm{VC}(G'')$ up to $\pm\varepsilon n/6$. By the Chernoff bound, we achieve this with probability $9/10$ by sampling $O(1/\varepsilon^2)$ vertices and computing the fraction of them in a fixed minimum vertex cover of G''. Such a vertex cover can be obtained by computing a minimum vertex cover for each connected component of G'' independently. Therefore, for every vertex v in the sample, we obtain $P[v]$ from $\mathcal{O}$. We compute a minimum vertex cover for the component induced by $P[v]$ in such a way that the vertex cover does not depend on which vertex in $P[v]$ was the query point. Finally, we check if the query point v belongs to the computed vertex cover for the component. In total, our procedure takes at most $O\left(k \cdot \tilde{d}/\varepsilon^3\right) + 2^{O(k)}/\varepsilon^2$ time.

To prove the same statement for minimum dominating set, we assume that all the high degree nodes are in the dominating set, and we take this into account when we compute optimal solutions for each connected component in the partition. This can increase the solution size by $\varepsilon n/3$ at most. For maximum independent set, it suffices to recall that the sum of the size of the maximum independent set and the size of the minimum vertex cover equals n, so a $(1, \varepsilon n)$-approximation to one of the problems immediately implies a $(1, \varepsilon n)$-approximation to the other one. ■

We now use the fact that the average degree of a graph with an excluded minor is $O(1)$. We combine Theorem 22 and Theorem 23, and achieve the following corollary.

Corollary 24 *For every H-minor free family of graphs (with no restriction on the maximum degree), there are $(1, \varepsilon n)$-approximation algorithms for the optimal solution*

Algorithm 3: Tester for H-Minor Freeness (for sufficiently large graphs)

Input: query access to a partition P given by an $(\varepsilon d/4, k)$-partitioning oracle for H-minor free graphs with degree bounded by d for the input graph

```
f := 0;
for j = 1, ..., t_1 (where t_1 = O(1/ε^2)) do
    Pick a random v ∈ V and a random i ∈ [d];
    if v has ≥ i neighbors, and the i-th neighbor of v not in P[v] then
        f := f + 1
if f/t_1 ≥ (3/8)ε then REJECT;
Select independently at random a set S of t_2 = O(1/ε) vertices of the graph;
if the graph induced by ⋃_{x∈S} P[x] is not H-minor free then REJECT;
else ACCEPT
```

size for minimum vertex cover, maximum independent set, and minimum dominating set that run in $2^{\text{poly}(1/\varepsilon)}$ *time.*

2.2.2 Testing Minor-Closed Properties

We now describe how partitioning oracles can be used for testing if a bounded-degree graph has a minor-closed property. The constant-time testability of minor-closed properties was first established by Benjamini, Schramm, and Shapira [5].

We now recall the definition of *property testing* in the bounded degree model [21]. A graph G is *ε-far* from a property $\mathcal{P}$ if it must undergo at least εdn graph operations to satisfy $\mathcal{P}$, where a single graph operation is either an edge removal or an edge insertion. An *ε-tester* T for property $\mathcal{P}$ is a randomized algorithm that has query access to G in the sense defined in the preliminaries, and:

- if G satisfies $\mathcal{P}$, T accepts with probability at least $2/3$,
- if G is ε-far from $\mathcal{P}$, T rejects with probability at least $2/3$.

Lemma 25 *Let H be a fixed graph. Let $\mathcal{O}$ be an $(\varepsilon d/4, k)$-partitioning oracle for the family of H-minor free graphs with degree bounded by d. There is an ε-tester for the property of being H-minor free in the bounded-degree model that provides $\mathcal{O}$ with query access to the input graph, makes $O(1/\varepsilon^2)$ uniform queries to $\mathcal{O}$, and uses $O((dk \log k)/\varepsilon + k^3/\varepsilon^6) = \text{poly}(d, k, 1/\varepsilon)$ time for computation.*

Proof For sufficiently large graphs, our tester is Algorithm 3. The value t_1 equals C_1/ε^2 for a sufficiently high constant C_1 such that by the Chernoff bound the number of edges cut by the partition P is approximated up to $\pm\varepsilon dn/8$ with probability 9/10. Let $t_3 = C_2/\varepsilon$ be an upper bound on the expected time to hit a set of size $\varepsilon|X|$ by independently taking random samples from X, where C_2 is a sufficiently large constant. We set t_2 in the algorithm to $10 \cdot q \cdot t_3$, where q is the number of connected components in H. Finally, we set t_4 to $C_3 \cdot k \cdot t_2^2$ for a sufficiently high constant C_3 such that for graphs on more than t_4 nodes, the probability that two samples from S belong to the same component $P[v]$ is at most 1/10.

If the number of vertices in the graph is at most $t_4 = O(k/\varepsilon^2)$, we read the entire graph, and check if the input is H-minor free in $O((k/\varepsilon^2)^3)$ time via the cubic algorithm of Robertson and Seymour [39]. For larger graphs, we run Algorithm 3. The loop in Lines 2–4 takes at most $O(d/\varepsilon^2)$ time. In Line 7, the induced graph can be read in $O((dk\log k)/\varepsilon)$ time, and then $O((k/\varepsilon)^3)$ time suffices to test whether it is H-minor free. Therefore, the amount of computation that the algorithm uses is $O((dk\log k)/\varepsilon + (k/\varepsilon^2)^3)$.

If G is H-minor free, then the fraction of edges cut by P is with probability $1-1/10$ at most $\varepsilon dn/4$. If this is the case, the estimate on the number of cut edges edges is at most $3\varepsilon dn/8$ with probability $1 - 1/10$. Moreover, every induced subgraph of G is also H-minor free, so G cannot be rejected in the loop in Line 5 of the algorithm. Hence, G is accepted with probability at least $8/10 > 2/3$.

Consider now the case when G is ε-far. If the partition P cuts more than $\varepsilon dn/2$ edges, the graph is rejected with probability $1 - 1/10 > 2/3$. Suppose now that P cuts fewer than $\varepsilon dn/2$ edges and the tester does not reject in Line 5. Let G' be the new graph after the partition. G' remains $\varepsilon/2$-far from H-minor freeness, and there are at least $\varepsilon dn/2$ edges that must be removed to get an H-minor free graph. This implies that G' is $\varepsilon/2$-far from H_i-minor freeness also for every connected component H_i, $1 \leq i \leq q$, of H. For every i, at least an $\varepsilon dn/2$ edges belong to a component of G' that is not H_i-minor free. It follows that at least εn vertices are incident to such an edge. Therefore, it suffices to pick in expectation t_3 random nodes to find a

Algorithm 4: The global partitioning algorithm with parameters k and δ

```
1  (π_1, ..., π_n) := random permutation of vertices;
2  P := ∅;
3  for i = 1, ..., n do
4      if π_i still in the graph then
5          if there exists a (k, δ)-isolated neighborhood of π_i in the remaining graph
           then
6              S := this neighborhood
7          else
8              S := {π_i}
9          P := P ∪ {S};
10         remove vertices in S from the graph
```

component that is not H_i-minor free. For q connected components of H, it suffices to pick in expectation $q \cdot t_3$ random nodes to find each of them. By picking, $10 \cdot q \cdot t_3$ random nodes, we find the components with probability $1 - 1/10$. Furthermore, since the considered graph is large, i.e., has at least t_4 nodes, the components for each i are different with probability $1 - 1/10$, and the graph is rejected in Line 7. Therefore, the probability that a graph that is ε-far is accepted is at most $3/10 < 1/3$. ■

By combining Theorem 22 with Lemma 25, we obtain a $2^{\text{poly}(1/\varepsilon)}$-time tester for H-minor freeness for graphs of degree $O(1)$. Since every minor-closed property can be expressed via a finite set of excluded minors H [40], it suffices to test if the input is ε/s-far from being minor free for each of them, where s is their number. We arrive at the following theorem.

Theorem 26 *For every minor-closed property $\mathcal{P}$, there is a uniform ε-tester for $\mathcal{P}$ in the bounded-degree model that runs in $2^{\text{poly}(1/\varepsilon)}$ time.*

2.3 A Simple Generic Partitioning Oracle

We start by presenting a partitioning oracle that works for any family of hyperfinite graphs. We later show more efficient oracles for specific families of hyperfinite graphs.

2.3.1 The Oracle

We introduce an auxiliary definition of a small subset S of vertices that contains a specific node, and has a small number of outgoing edges relatively to S.

Definition 27 Let $G = (V, E)$ be a graph. For any subset $S \subset V$, we write $e_G(S)$ to denote the number of edges in E that have exactly one endpoint in S.

We say that $S \subseteq V$ is a (k, δ)-*isolated neighborhood* of $v \in V$ if $v \in S$, the subgraph induced by S is connected, $|S| \leq k$, and $e_G(S) \leq \delta|S|$.

We now show that a random vertex has an isolated neighborhood of required properties with high probability.

Lemma 28 *Let $G = (V, E)$ be a $\rho(\varepsilon)$-hyperfinite graph with degree bounded by d, where $\rho(\varepsilon)$ is a function from $\mathbb{R}_+$ to $\mathbb{R}_+$. Let $G' = (V', E')$ be a subgraph of G that is induced by at least δn vertices. For any $\varepsilon \in (0, 1)$, the probability that a random vertex in G' does not have a $(\rho(\varepsilon^2\delta/28800), \varepsilon/120)$-isolated neighborhood in G' is at most $\varepsilon/120$.*

Proof Any induced subgraph of G can still be partitioned into components of size at most $\rho(\varepsilon)$ by removing at most εn edges. Since G' has at least δn vertices, it is still $(\varepsilon/\delta, \rho(\varepsilon))$-hyperfinite for any $\varepsilon > 0$, or equivalently, it is $(\varepsilon, \rho(\varepsilon \cdot \delta))$-hyperfinite for any $\varepsilon > 0$.

Therefore, there is a set $S' \subseteq E'$ of at most $(\varepsilon^2/28800)|V'|$ edges such that if all the edges in S' are removed, the number of vertices in each connected component is at most $\rho(\varepsilon^2\delta/28800)$. Denote the achieved partition of vertices into connected components by P. We have

$$\mathbb{E}_{v \in V'}\left[\frac{e_G(P[v])}{|P[v]|}\right] = \sum_{S \in P} \frac{|S|}{|V'|} \cdot \frac{e_G(S)}{|S|} = \frac{2|S'|}{|V'|} \leq \frac{\varepsilon^2}{14400}.$$

By Markov's inequality, the probability that a random $v \in V'$ is such that $e(P[v])/|P[v]| > \frac{\varepsilon}{120}$ is at most $\varepsilon/120$. Otherwise, $P[v]$ is an $(\rho(\varepsilon^2\delta/28800), \varepsilon/120)$-isolated neighborhood of v. ■

Finally, we now use the above lemma to construct a partitioning oracle.

Proof [Proof of Theorem 21] We set $k = \rho(\varepsilon^3/3456000)$ and $\delta = \varepsilon/120$. Consider the global Algorithm 4 with these parameters. The algorithm partitions the vertices of the input graph into sets of size at most k. We define a sequence of random variables X_i, $1 \leq i \leq n$, as follows. X_i corresponds to the i-th vertex removed by Algorithm 4 from the graph. Say, the remaining graph has $n - t$ vertices, and the algorithm is removing a set S of r vertices. Then we set $X_{t+1} = \ldots = X_{t+r} = e_{G'}(S)/r$, where G' is the graph before the removal of S. Note that $\sum_{i=1}^{n} X_i$ equals the number of edges between different parts in P. For every i, if X_i corresponds to a set S that was a (k, δ)-isolated neighborhood of the sampled vertex, then $X_i \leq \delta = \varepsilon/120$. Otherwise, we only know that $X_i \leq d$. However, by Lemma 28, if $i \leq n - \varepsilon n/120$, this does not happen with probability greater than $\varepsilon/120$. Therefore, for every $i \leq n - \varepsilon n/120$, we have

$$\mathbb{E}[X_i] \leq \varepsilon/120 + d \cdot \varepsilon/120 \leq 2\varepsilon d/120.$$

For $i > n - \varepsilon n/120$, we again use the bound $X_i \leq d$. Altogether, this gives that the expected number of edges connecting different parts of P is at most $2\varepsilon dn/120 + \varepsilon dn/120 < \varepsilon dn/40$. Markov's inequality implies that the number of such edges is at most $\varepsilon dn/2$ with probability $1 - 1/20$.

Algorithm 4 can be simulated locally as follows. For each vertex v, we want to compute $P[v]$. Instead of a random permutation, we independently assign a random number $r(v)$ uniformly selected from the range $[0, 1]$. We only generate $r(v)$'s when they are necessary, and we store them as they may be needed later again. The numbers generate a random ordering of vertices. To compute $P[v]$, we first recursively compute $P[w]$ for each vertex w with $r(w) < r(v)$ and distance to v at most $2 \cdot k$. If $v \in P[w]$ for one of those w, then $P[v] = P[w]$. Otherwise, we search for a (k, δ)-isolated neighborhood of v, keeping in mind that all vertices in $P[w]$ that we have recursively computed are no longer in the graph. If we find such an neighborhood, we set $P[v]$ to it. Otherwise, we set $P[v] = \{v\}$.

We now analyze the above local simulation procedure, using Lemma 3 in Chapter 1. Our computation graph is $G^\star = (V, E^\star)$, where $E^\star$ connects all pairs of vertices that are at distance at most $2 \cdot k$ in the input graph. The degree of $G^\star$ is bounded by $D = d^{O(k)}$. The expected number of vertices w for which we have to compute $P[w]$ to obtain $P[v]$ for a fixed vertex v is at most $T = 2^{d^{O(k)}}$. Suppose that we run the procedure for every vertex in the graph, but we never recursively compute $P[w]$ for more than $T' = 40T/\varepsilon$ vertices w. The probability that we do not compute $P[v]$ for a given v is by Markov's inequality at most $\varepsilon/40$. The expected number of vertices that we fail for is bounded by $\varepsilon n/40$. Using Markov's inequality again, with probability $1 - 1/20$, the number of such vertices is bounded by $\varepsilon n/2$.

The oracle works as follows for a given query vertex v. It first checks whether there is at least one vertex u for which we compute $P[u]$ using at most T' recursive calls, and $v \in P[u]$. This can be done by running the recursive simulation procedure for every vertex u at distance less than k from v. If there is such vertex u, the oracle returns $P[u]$. Otherwise, it returns the singleton $\{v\}$. This procedure clearly provides query access to a partition P' of the vertex set with all components of size at most k.

Let us show now that the number of edges cut in P' is likely to be small. First, we know that P cuts at most $\varepsilon dn/2$ vertices with probability at least $1 - 1/20$. P' additionally partitions components of in P in which the recursive evaluation does not finish in T' recursive calls for any vertex. The number of such vertices, and also the number of vertices in such components, is at most $\varepsilon n/2$ with probability $1 - 1/20$. This means that with probability $1 - 1/20$, at most $\varepsilon dn/2$ additional edges are cut in P'. Therefore, with probability at least $1 - 1/20 - 1/20 = 9/10$, the P' cuts at most εdn edges.

Let us now bound the number of graph queries that the oracle makes to answer a single query. It simulates the recursive procedure starting at $d^{O(k)}$ vertices with at most $T' = 2^{d^{O(k)}}/\varepsilon$ recursive calls each time. Each recursive call has query complexity $d^{O(k)}$ associated with reading the neighborhood of radius $O(k)$ at every vertex. In total, this gives a bound of $2^{d^{O(k)}}/\varepsilon$ queries. For q queries, the amount of computation the oracle uses is $q \log q \cdot 2^{d^{O(k)}}/\varepsilon$. The extra logarithmic factor comes from a dictionary

that is necessary to store random numbers $r(v)$ assigned to vertices v of the graph, once they are generated. ■

2.4 A Partitioning Oracle for Hyperfinite Graphs

We focus on the family of hyperfinite graphs where $\rho(\varepsilon) = \text{poly}(1/\varepsilon)$. We show that there is an efficient partitioning oracle for this family of graphs which only makes at $2^{\text{poly}(1/\varepsilon,d)}$ queries to the graph to answer a query to the oracle.

We describe a local distributed partitioning algorithm that builds the required partition. Recall that a distributed algorithm is local if it runs in a constant number of rounds. Such an algorithm can be used to construct an efficient partitioning oracle, since for every query point, one can compute the distributed algorithm's output by simulating it for nodes within a constant radius around the query point. See the paper of Parnas and Ron [35] for more details.

2.4.1 Preliminaries

Given an undirected graph $G = (V, E)$. For a set of vertices $S \subseteq V$, the volume $\mu(S)$ is defined to be,

$$\mu(S) := \sum_{x \in S} \mu(x)$$

where $\mu(x)$ denotes the degree of the vertex x. The conductance of S is defined to be

$$\phi(S) := e_G(S)/\mu(S)$$

Growing an Isolated Neighborhood Using Volume-Biased Evolving Set Process

Starting from a given vertex v, searching for an isolated neighborhood that *contains* v is one of the challenges most partitioning oracles have to face. In this oracle, we use the Volume-Biased Evolving Set Process (VBESP) of Andersen and Peres [2]. This method does not only take much less queries to the graph (linear in the size of the

neighborhoods), but also guarantee some nice local properties of the neighborhoods which will be crucial for our analysis. The following lemma mimics, with minor modification, the Theorem 2 in the paper of Andersen and Peres [2].

Lemma 29 *Let $G = (V, E)$ be an undirected graph with degree bounded by $d \geq 2$. Let $A \subseteq V$ be a $(k, \varepsilon^5/(6,220,800,000 \cdot d^3 \cdot \log(2kd)))$-isolated neighborhood where $\varepsilon \in (0,1)$ and $k \geq 1$. There is a subset $C \subseteq A$ with size of at least $(1-\varepsilon)|A|$ for which the following holds. For every $v \in C$, with probability at least $1 - \varepsilon/60$, we can find an isolated neighborhood S_v will satisfy all the following:*

1. *S_v is a $(2k, \varepsilon/30)$-isolated neighborhood.*
2. *$|S_v \setminus A| \leq \varepsilon \cdot |S_v|/30 \cdot d$*

We need the following proposition which mimics Proposition 5 in the paper of Andersen and Peres [2].

Proposition 30 *Let (X_i) be a lazy random walk Markov chain starting from the vertex x. For any set $A \subseteq V$ and integer T, let*

$$\mathrm{esc}(x, T, A) := \mathcal{P}_x \left[\cup_{j=0}^{T} (X_j \notin A) \right],$$

which is the probability that a lazy random walk starting from x leaves A within the first T steps, and define $A_T := \{x \in A | \mathrm{esc}(x, T, A) \leq d \cdot T \cdot \phi(A)/\varepsilon\}$. Then, $|A_T| \geq (1 - \varepsilon/2) \cdot |A|$

Proof By Theorem 2.5 in [43], it is shown that $\mu(A)^{-1} \sum_{x \in A} \mu(x) \mathrm{esc}(x, T, A) \leq T\phi(A)/2$. By Markov inequality,

$$|A_T^c| \leq \mu(A_T^c) \leq \varepsilon \mu(A)/2d \leq \varepsilon |A|/2$$

. ■

Proof [Proof of Lemma 29]

Since A induces a connected subgraph of G,

$$\phi(A) \leq e_G(A)/|A| \leq \varepsilon^6/(6,220,800,000 \cdot d^5 \cdot \log(2kd))$$

We simulate the volume-biased ESP started from $S_0 = \{v\}$ as in [2] with stopping time τ such that the process stops when $\tau = T = 81,000\frac{1}{\varepsilon^3}d^2\log(2kd)$ or $|S_\tau|$ is larger than $2k$. By Lemma 1 in [2],

$$\hat{E}\left[\sum_{j=0}^{\tau}\phi(S_j)^2\right] \leq 4\hat{E}\log\frac{\mu(S_\tau)}{\mu(S_0)} \leq 4\log(2kd)$$

Therefore, by Markov's inequality, the event $\sum_{j<\tau}\phi(S_j)^2 \leq 960\cdot\log(2kd)/\varepsilon$ holds with probability at least $1-\varepsilon/240$. Let S_v be the smallest conductance set in the sample path $(S_0,\cdots,S_\tau)$,

$$\hat{P}_v\left[\phi(S_v) \leq \sqrt{960\cdot\tau^{-1}\cdot\log(2kd)/\varepsilon}\right] \geq 1-\varepsilon/240 \tag{2.1}$$

Also, by Lemma 2 in [2], if we consider a sample path $(S_0,\cdots,S_T)$ from the volume-biased evolving set process,

$$\begin{aligned}
\hat{P}_x\left[\max_{t\leq T}\frac{\mu(S_t\setminus A)}{\mu(S_t)} > 240\frac{1}{\varepsilon}\mathrm{esc}(x,T,A)\right] &\leq \varepsilon/240\\
\hat{P}_x\left[\forall_{t\leq T}\frac{\mu(S_t\setminus A)}{\mu(S_t)} < 240\frac{1}{\varepsilon}\mathrm{esc}(x,T,A)\right] &\geq 1-\varepsilon/240\\
\hat{P}_x\left[\forall_{t\leq T}\mu(S_t\setminus A) < 240\frac{1}{\varepsilon}\mathrm{esc}(x,T,A)\mu(S_t)\right] &\geq 1-\varepsilon/240\\
\hat{P}_x\left[\forall_{t\leq T}|S_t\setminus A| < 240\frac{1}{\varepsilon}\mathrm{esc}(x,T,A)d|S_t|\right] &\geq 1-\varepsilon/240
\end{aligned} \tag{2.2}$$

Let $C := A_T = \{x \in A | \text{esc}(x, T, A) \leq d \cdot T \cdot \phi(A)/\varepsilon\}$. For $x \in C$:

$$\hat{P}_x \left[\forall_{t \leq T} |S_t \setminus A| < 240 \frac{1}{\varepsilon^2} T \cdot \phi(A) \cdot d^2 |S_t| \right] \geq 1 - \varepsilon/240$$

$$\hat{P}_x \left[\forall_{t \leq T} |S_t \setminus A| < \frac{\varepsilon |S_t|}{30d} \right] \geq 1 - \varepsilon/240 \quad (3)$$

By (1), (3) and the Union Bound, for $x \in C$:

$$\hat{P}_x \left[\phi(S_v) \leq \sqrt{960 \cdot \tau^{-1} \cdot \log(2kd)/\varepsilon} \ \& \ \forall_{t \leq T} |S_t \setminus A| < \frac{\varepsilon |S_t|}{30d} \right] \geq 1 - \varepsilon/120$$

$$\hat{P}_x \left[\frac{e_G(S_v)}{|S_v|} \leq d\sqrt{960 \cdot \tau^{-1} \cdot \log(2kd)/\varepsilon} \ \& \ \forall_{t \leq T} |S_t|) < 2k \right] \geq 1 - \varepsilon/120$$

$$\hat{P}_x \left[\frac{e_G(S_v)}{|S_v|} \leq d\sqrt{960 \cdot T^{-1} \cdot \log(2kd)/\varepsilon} \right] \geq 1 - \varepsilon/120$$

$$\hat{P}_x \left[\frac{e_G(S_v)}{|S_v|} \leq \varepsilon/30 \right] \geq 1 - \varepsilon/120 \quad (4)$$

The lemma follows from (3), (4) and the Union Bound. ■

Note that it is also possible to use a simple brute force search algorithm to search for isolated neighborhoods. However, such method does not work well with the design of this oracle since it does not guarantee a local property as Property 2 of Lemma 29.

Hyperfinite Graphs With Polynomial Hyperfinite Function

The following corollary can be proved by the same argument in the proof of Lemma 28.

Corollary 31 *Let $R : \mathcal{R}^2 \to \mathcal{R}$ be a polynomial. Let $G = (V, E)$ be a $(\epsilon, R(d, \epsilon))$-hyperfinite graph with degree bounded by d. Given $\varepsilon \in (0, 1)$, let K be the smallest integer that satisfies the following constraint:*

$$K \geq R\left(\varepsilon^8/(1,492,992,000,000 \cdot d^5 \cdot \log(2Kd)), d\right)$$

Let $G' = (V', E')$ be a subgraph of G, which is induced by at least $\varepsilon/4 \cdot n$ vertices. The probability that a random vertex in G' does not have an $(K, \varepsilon^6/(6,220,800,000 \cdot d^5 \cdot$

$\log(2Kd)))$-*isolated neighborhood in* G' *is at most* $\varepsilon/30$. *Furthermore, there exists a partition* P *for* G' *such that the probability that* $P[v]$, *for a random vertex* $v \in V'$, *is not an* $(K, \varepsilon^6/6,220,800,000 \cdot d^5 \cdot \log(2Kd))$-*isolated neighborhood is at most* $\varepsilon/30$.

In order to simplify the notations, given a polynomial $R : \mathcal{R}^2 \to \mathcal{R}$, let us define $\mathcal{K}_R : \mathcal{R}_+ \to \mathcal{R}_+$ such that $\mathcal{K}_R(\varepsilon)$ is the minimum positive integer satisfying the following constrain:

$$\mathcal{K}_R(\varepsilon) \geq R\left(\varepsilon^8/(1,492,992,000,000 \cdot d^5 \cdot \log(2\mathcal{K}_R(\varepsilon) \cdot d)), d\right)$$

Lemma 32 *For any polynomial* $R : \mathcal{R}^2 \to \mathcal{R}$, $\mathcal{K}_R(\varepsilon)$ *is also a polynomial* $1/\varepsilon$ *and* d.

Proof Let r be the degree of R. Let C be a constant such that $R(\varepsilon, d) \leq C(d/\varepsilon)^r$. Consider the following

$$p(\varepsilon) = (1,492,992,000,000 \cdot 3r \cdot C \cdot d^5 \cdot \log(2d) \cdot \frac{1}{\varepsilon^8})^{3r}$$

It can be checked that $p(\varepsilon) \geq R\left(\varepsilon^8/(1,492,992,000,000 \cdot d^5 \cdot \log(2p(\varepsilon)d)), d\right)$. Therefore, $\mathcal{K}_R(\varepsilon) \leq p(\varepsilon) = \text{poly}(1/\varepsilon, d)$.

■

Also, define $\beta_R : \mathcal{R}_+ \to \mathcal{R}_+$ such that

$$\beta_R(\varepsilon) = \varepsilon^6/(6,220,800,000 \cdot d^5 \cdot \log(2\mathcal{K}_R(\varepsilon) \cdot d))$$

.

2.4.2 The Partitioning Oracle

The oracle simulates a distributed algorithm that repeatedly selects and removes many disjoint isolated neighborhoods at once. As we will show in the analysis, after each step of selecting and removing disjoint neighborhoods, the size of the graph is expected to decrease by a fraction of $1/\text{poly}(1/\varepsilon, d)$. Therefore, by repeating the step

Algorithm 5: A single step of selecting and removing disjoint neighborhoods

1 For every vertex v in G, color v white.
2 For every vertex v in G, if the neighborhood grown by VBESP method starting from v is a $(2\mathcal{K}_R(\varepsilon), \varepsilon/30)$-isolated neighborhood, set S_v to that neighborhood. Otherwise, set $S_v := \emptyset$
3 For every vertex v in G, if v appears in more than $60\mathcal{K}_R^2(\varepsilon)/\varepsilon$ neighborhoods grown in Step 2, color v black and set $S_v := \emptyset$.
4 For every white vertex v in G, remove black vertices from S_v. If S_v is no longer a $(2\mathcal{K}_R(\varepsilon), \varepsilon/15 + \varepsilon^2/450)$-isolated neighborhood, set $S_v := \emptyset$
5 For every vertex v in G, set r_v be a random number in $[0, 1]$.
6 For every vertex v in G, if S_v is $(2\mathcal{K}_R(\varepsilon), \varepsilon/15 + \varepsilon^2/450)$-isolated neighborhood and S_v does not intersect with any S_u such that $r_u <= r_v$, add S_v to the partition and remove it from G.

poly$(1/\varepsilon, d)$ times, the size of the graph that remains is small (less than εn) with high probability and can be ignored. Since there are only poly$(1/\varepsilon, d)$ steps and each step only takes poly$(1/\varepsilon, d)$ communication rounds, it can be deduced that the distributed algorithm requires only poly$(1/\varepsilon, d)$ communication rounds. Applying the simulating technique described in the paper of Ron and Parnas [35], we can obtain a partitioning oracle of query complexity $2^{\text{poly}(1/\varepsilon,d)}$.

Removing A Maximal Set of Isolated-Neighborhoods

We describe a single step of the distributed algorithm. Algorithm 5 starts by finding an isolated neighborhood for each of the vertex that remains in the graph. Then, it considers these neighborhoods in a random order and selects them greedily so that no pair of selected neighborhoods intersect. Finally, the algorithm removes all selected neighborhoods from the graph at once. The follow lemma shows that after an execution of Algorithm 5, the number of vertices in the graph is expected to decrease by a large enough fraction.

Lemma 33 *Let $G = (V, E)$ be a $(\epsilon, R(d, \epsilon))$-hyperfinite graph with degree bounded by $d \geq 2$. Given $\varepsilon \in (0, 1)$. Let $G' = (V', E')$ be a subgraph of G that is induced by at least $\varepsilon n/4$ vertices. Let $G'' = (V'', E'')$ be the graph that remains after an execution*

of Algorithm 5 on input G'. Then we have

$$E[|V''|] \leq \gamma \cdot |V'|$$

where $\gamma := 1 - \frac{6\varepsilon - \varepsilon^2}{1440 \cdot \mathcal{K}_R(\varepsilon)^4}$.

Proof By Corollary 31, there exists a partition P for G' and a subset $S \subseteq V'$ of size at least $(1-\varepsilon/30)\cdot|V'|$ such that and for every $v \in S$, $P[v]$ is an $(\mathcal{K}_R(\varepsilon), \beta_R(\varepsilon))$-isolated neighborhood in G'. A part in$P$ is *good* if it is a $(\mathcal{K}_R(\varepsilon), \beta_R(\varepsilon))$-isolated neighborhood. Let P_{good} be the set of good parts in P. Since each part in the partition has at most $\mathcal{K}_R(\varepsilon)$ vertices, P has at least $\frac{1-\varepsilon/30}{\mathcal{K}_R(\varepsilon)}|V'|$ distinct good parts,

$$|P_{good}| \geq \frac{1-\varepsilon/30}{\mathcal{K}_R(\varepsilon)}|V'| \tag{1}$$

Let v be a vertex in G'. Let $f(v) = |S_v|$. Let $g(v)$ be the number of grown isolated neighborhoods that contain v (i.e. $g(v) = \{u \in V_k : v \in S_u\}$). From the definition, v is colored black in Step 3 if and only if $g(v) \geq 60 \cdot \mathcal{K}_R^2(\varepsilon)/\varepsilon$. Also,

$$\sum_{v \in V'} g(v) = \sum_{v \in V'} f(v) \leq 2\mathcal{K}_R(\varepsilon) \cdot |V'|$$

Therefore, the number of black vertices is at most

$$2\mathcal{K}_R(\varepsilon) \cdot |V'| \cdot \frac{\varepsilon}{60\mathcal{K}_R^2(\varepsilon)} = \frac{\varepsilon}{30\mathcal{K}_R(\varepsilon)}|V'| \tag{2}$$

By (1), (2) and the Union bound, the number of good parts that do not contain any black vertices is at least

$$\frac{1-\varepsilon/15}{\mathcal{K}_R(\varepsilon)}|V'| \tag{3}$$

For each part A of the partition P, let X_A be an indicator variable for the event that there is some vertex in A can grow a $(2\mathcal{K}_R(\varepsilon), \varepsilon/30)$-isolated neighborhood. By

Lemma 29,

$$\Pr_{A \in P_{good}}\Big[X_A = 1\Big] \geq 1 - \varepsilon/60$$
$$E_{A \in P_{good}}\Big[X_A\Big] \geq 1 - \varepsilon/60$$
$$E\Big[\sum_{A \in P_{good}} X_A^c\Big] \leq \frac{\varepsilon}{60}|P_{good}|$$

By Markov's in equality, with probability at least $1/2$, $\sum_{A \in P_{good}} X_A^c \leq \frac{\varepsilon}{30}|P_{good}|$. Thus, with probability at least $1/2$,

$$\sum_{A \in P_{good}} X_A \geq (1 - \varepsilon/30)|P_{good}| \geq \frac{(1-\varepsilon/30)^2}{\mathcal{K}_R(\varepsilon)}|V'|/ \geq \frac{1-\varepsilon/15}{\mathcal{K}_R(\varepsilon)}|V'| \tag{4}$$

We say a part A is *a surviving* if A is good and A does not contain any black vertices and there is some vertex in A can grow a $(2\mathcal{K}_R(\varepsilon), \varepsilon/30)$-isolated neighborhood . Let $P_{surviving} \subseteq P$ be the set of surviving parts. By (3), (4) and the Union bound, with probability at least $1/2$,

$$P_{surviving} \geq \frac{1-\varepsilon/6}{\mathcal{K}_R(\varepsilon)}|V'| \tag{5}$$

Now consider a surviving part A. Let v be a vertex in A that can grow a $(2\mathcal{K}_R(\varepsilon), \varepsilon/30)$-isolated neighborhood, and let S_v be the neighborhood grown by v. Since A is surviving, $S_v \cap A$ does not contain any black vertices. The part of S_v outside A is very small and contains at most $\frac{\varepsilon}{30d}|S_v|$ vertices. Therefore, in the worst case, the removal of black vertices from S_v only adds at most $\varepsilon \cdot |S_v|/30$ edges and removes at most $\varepsilon \cdot |S_v|/30$ vertices from S_v. Therefore, even after the removal of black vertices, S_v is still a $(2\mathcal{K}_R(\varepsilon), \varepsilon/15 + \varepsilon^2/450)$-isolated neighborhood.

Let $r(v)$ be the number of grown isolated neighborhoods intersecting with S_v in Step 6. Observe that, at this step, all vertices in S_v are white. Therefore, we can bound $r(v)$ by

$$\sum_{u \in S_v} g(u) \leq 60\mathcal{K}_R^2(\varepsilon)\varepsilon|S_v| \leq \frac{120\mathcal{K}_R^3(\varepsilon)}{\varepsilon}$$

Observe that if S_v has higher rank than every other S_u that intersects with S_v, then S_v will be removed from the graph. Therefore, with probability at least $\frac{\varepsilon}{120\mathcal{K}_R^3(\varepsilon)}$, S_u will be removed from the graph. Since S_v contains at least one vertex of A, *with probability at least* $\frac{\varepsilon}{120\mathcal{K}_R^3(\varepsilon)}$, *at least one vertex of* A *will be removed.*

For a surviving part A, let Z_A be the indicator variable for the event that at least one vertex is removed from A after the execution of Algorithm 5,

$$\begin{aligned} \Pr_{A \in P_{surviving}}\Big[Z_A = 1\Big] &\geq \frac{\varepsilon}{120\mathcal{K}_R^3(\varepsilon)} \\ E_{A \in P_{surviving}}\Big[Z_A\Big] &\geq \frac{\varepsilon}{120\mathcal{K}_R^3(\varepsilon)} \\ E\Big[\sum_{A \in P_{surviving}} Z_A\Big] &\geq \frac{\varepsilon}{120\mathcal{K}_R^3(\varepsilon)} |P_{surviving}| \end{aligned} \tag{6}$$

By (5) and h(6),

$$E\Big[\sum_{A \in P_{surviving}} Z_A\Big] \geq \Pr\Big[P_{surviving} \geq (1-\varepsilon/6)\cdot|V'|/\mathcal{K}_R(\varepsilon)\Big]\cdot\frac{(\varepsilon-\varepsilon^2/6)}{120\mathcal{K}_R^4(\varepsilon)}|V'| \geq \frac{6\varepsilon-\varepsilon^2}{1440\mathcal{K}_R^4(\varepsilon)}|V'|$$

Thus,

$$E\Big[|V''|\Big] \leq E\Big[|V'| - \sum_{A \in P_{surviving}} Z_A\Big] \leq (1-(6\varepsilon-\varepsilon^2)/1440K^4)\cdot|V'|$$

■

It can be shown that Algorithm 5 only requires a constant communication rounds. For a vertex v in the graph, observe that there exists a vertex $u \in S_v$ such that the distance from u to v is at most $\mathcal{K}_R(\varepsilon)$ (see the paper by Andersen and Peres [2] for details). Thus, the distance from v to any vertex in S_v is at most $3\mathcal{K}_R(\varepsilon)$. Let us consider each of the steps in the algorithm that requires communication between different vertices:

Step 2 From the observation above, it takes $3\mathcal{K}_R(\varepsilon)$ communication rounds to compute

S_v.

Step 3 Observe that in order to compute the number neighborhoods containing v, it is sufficient to compute the set of neighborhoods grown by all vertices within distance of $3\mathcal{K}_R(\varepsilon)$ from v. Thus, this step takes at most $6\mathcal{K}_R(\varepsilon)$ communication rounds.

Step 4 It takes $3\mathcal{K}_R(\varepsilon)$ communication rounds to decide which vertices in S_v are black and remove them.

Step 6 In order to decide if S_v is removed, it is sufficient to compute the set of neighborhoods grown by all vertices within a distance of $6\mathcal{K}_R(\varepsilon)$ from v. Thus, this step takes at most $9\mathcal{K}_R(\varepsilon)$ communications rounds.

In total, Algorithm 5 requires at most $21\mathcal{K}_R(\varepsilon)$ communication rounds.

Full Distributed Algorithm

Our distributed algorithm computes the required partition by executing Algorithm 5 multiple times. The following lemma shows that after a small number of executions of Algorithm 5, with high probability the size of the remaining graph can be neglected.

Lemma 34 *Let $G = (V, E)$ be a $(\epsilon, R(d, \epsilon))$-hyperfinite graph with degree bounded by $d \geq 2$. Given $\varepsilon \in (0, 1)$. There is a distributed partitioning algorithm that runs in $\text{poly}(d, 1/\varepsilon)$ rounds, and determines a partition of G such that:*

- *The size of each connected component is bounded by $\text{poly}(d, 1/\varepsilon)$.*
- *With probability $9/10$, the number of cut edges is at most $\varepsilon d|V|$.*

Proof We describe our distributed algorithm. Let $M := 2 \cdot (\log(1/\varepsilon) + 6)/(-\log(1 - \gamma/2) \cdot \gamma)$. Our distributed algorithm first runs Algorithm 5 M times to select and remove neighborhoods. Then, for each vertex v that remains in the graph, v constitutes a part of size one in the partition.

It is clear from the distributed algorithm that the size of each connected component is bounded by $2\mathcal{K}_R(\varepsilon) = \text{poly}(d, 1/\varepsilon)$ for polynomial R.

It remains to bound the number of cut edges. Let us first bound the number of edges that remain after M execution of Algorithm 5.

For $0 \leq k \leq M$, let $G_k = (V_k, E_k)$ be the graph that remains after k executions of Algorithm 5. Let Y_k be the random variables defined as following,

$$Y_k = \begin{cases} |V_k| & \text{if } |V_k| \geq \varepsilon n/4 \\ \gamma |V_{k-1}| & \text{otherwise} \end{cases}$$

By Lemma 33, for all k, $0 \leq k \leq$,

$$E\Big[Y_k - Y_{k+1}|\Big] \geq \gamma Y_k \tag{7}$$

We also have,

$$\begin{aligned} E\Big[Y_k - Y_{k+1}\Big] \leq \\ \Pr\Big[Y_k - Y_{k+1} \geq \gamma \cdot Y_k/2\Big] \cdot Y_k + \Pr\Big[Y_k - Y_{k+1} \leq \gamma \cdot Y_k/2\Big] \cdot \gamma \cdot Y_k/2 \leq \\ \Pr\Big[Y_k - Y_{k+1} \geq \gamma \cdot Y_k/2\Big] Y_k + \gamma \cdot Y_k/2 \end{aligned} \tag{8}$$

By (7) and (8),

$$\begin{aligned} \Pr\Big[Y_k - Y_{k+1} \geq \gamma \cdot Y_k/2\Big] \geq \gamma/2 \\ \Pr\Big[\log(Y_k) - \log(Y_{k+1}) \geq -\log(1-\gamma/2)\Big] \geq \gamma/2 \\ E\Big[\log(Y_k) - \log(Y_{k+1})\Big] \geq -\log(1-\gamma/2) \cdot \gamma/2 \\ E\Big[\log(Y_0) - \log(Y_M)\Big] \geq -\log(1-\gamma/2) \cdot \gamma \cdot M/2 = \log(1/\varepsilon) + 6 \\ E\Big[Y_M\Big] \leq \varepsilon n/64. \end{aligned}$$

By Markov's inequality

$$\Pr\Big[Y_M \geq \varepsilon n/4\Big] \leq 1/16.$$

Therefore,

$$\Pr\left[|E_M| \leq \varepsilon nd/2\right] \geq \Pr\left[|V_M| \leq \varepsilon n/2\right] \geq \Pr\left[Y_M + \varepsilon n/4 \leq \varepsilon n/2\right] \geq 9/10$$

Thus, with probability 9/10, the number of remaining edges after M executions of the Algorithm 5 is at most $\varepsilon nd/2$. Also, observe that the number of edges going out of all removed neighborhoods is at most $\varepsilon dn/10$. Therefore, with probability 9/10, the total of cut edges is at most εdn.

■

Proof of Theorem 22

Proof

For every query the oracle locally simulates the distributed algorithm from Lemma 34 with the algorithm's ε set to ε/d. See the paper of Parnas and Ron [35] for details how to conduct such simulation. The required number of queries to the graph is $d^{\text{poly}(d,1/\varepsilon)}$ for every query to the oracle. The oracle needs to store previous coin tosses for each seen vertex in the graph. The total computation time for q queries is at most $q \log q \cdot 2^{\text{poly}(1/\varepsilon,d)}$, where the extra $\log q$ factor comes from the use of a dictionary. ■

Chapter 3

Efficient Partitioning Oracle for Constant Treewidth Graphs

As we have seen in the previous chapter, there exists a partitioning oracle for family of graphs with an excluded minor whose running time is independent to the graph size. However, the running time of that oracle is exponential in both $1/eps$ and d. This raises a natural question whether there exists a partitioning oracle for minor-free graphs that runs in $\text{poly}(d, 1/\varepsilon)$ time. An affirmative answer would imply a polynomial tester for minor-closed properties in the bounded degree model, solving Open Problem 4 of Benjamini *et al.* [5].

In this chapter, we try to make a step closer to the answer of the open question. We restrict our consideration to the family of graphs with constant treewidth and show that it is possible to construct a parititioning oracle for this famliy of graph that runs $\text{poly}(1/\varepsilon)$ time.

Bounded Treewidth. The *tree decomposition* and *treewidth* were first introduced by Robertson and Seymour [37, 38], and later found many applications in the design on algorithms, and in machine learning (a nice though outdated survey is [7], and see also [3, 8] for more recent applications). Given a graph $G = (V, E)$, a treewidth decomposition of G is a pair $(\mathcal{X}, \mathcal{T})$, where $\mathcal{X} = (X_1, X_2, \cdots, X_m)$ is a family of subsets of V and $\mathcal{T}$ is a tree whose nodes are the subsets X_i, satisfying the following

properties:

1. Every vertex of V is associated with at least one node in tree $\mathcal{T}$. That is $\cup_i X_i = V$.

2. For every edge $(u, v) \in E$, there is a node $X_i \in \mathcal{X}$ that contains both u and v.

3. For every vertex $v \in V$, the set of nodes in $\mathcal{T}$ associated with v forms a connected subset of $\mathcal{T}$.

The width of a tree decomposition is the size of its largest set X_i minus one. The *treewidth* of G is defined as taking the minimum over decompositions of this quantity.

Examples of graphs families with bounded treewidth include outerplanar graphs, series-parallel graphs, cactus graphs, and pseudoforests. We show that there is an efficient partitioning oracle for the family of graphs with constant treewidth.

Theorem 35 *Let $G = (V, E)$ be a graph with maximum degree bounded by d and treewidth bounded by h. There is an oracle $\mathcal{O}$ that given an $\varepsilon \in (0, 1/2)$, and query access to G, provides query access to a function $f : V \to \mathcal{P}(V)$ of the following properties:*

1. *For all $v \in V$, $v \in f(v)$.*

2. *For all $v \in V$, and all $w \in f(v)$, $f(v) = f(w)$.*

3. *With probability $9/10$, $|\{(v, w) \in E : f(v) \neq f(w)\}| \leq \varepsilon |V|$.*

4. *For all $v \in V$, $|f(v)| = \tilde{O}(\frac{2^{2(h+1)} d^7 h^7}{\varepsilon^3})$.*

5. *For all v, the oracle uses $(\frac{d}{\varepsilon})^{O(h)} \cdot \log Q$ time to compute $f(v)$, where Q is the number of previous queries to the oracle.*

6. *The function f is independent of the queries to $\mathcal{O}$.*

Our main result immediately implies efficient approximation algorithms (run in poly$(1/\varepsilon)$ time) for many graph problems such as minimum vertex cover, maximum independent set and set cover. The oracle can also be used to design efficient property

tester for constant-treewidth properties such as cycle-freeness, outerplanariry. See Chapter 3 for how to use a partitioning oracle to design approximation algorithms and property tester.

In Section 3.1, we show how to construct an efficient partitioning oracle for forests and trees. Although, this oracle only works for a very restricted family of graphs, its construction is simple and contains some flavors that are crucial to the design of the oracle for constant treewidth graph in Section 3.2.5. In Section 3.2.5 and Section 3.3, we prove our main result showing an partitioning oracle for constant treewidth graphs.

3.1 A Simple Partitioning Oracle for Forests

Theorem 36 (Local Partitions for Forests) *Let $T = (V, E)$ be a forest with maximum degree bounded by d. There is an oracle $\mathcal{O}$ that given an $\varepsilon \in (0, 1/2)$, and query access to T, provides query access to a function $f : V \to \mathcal{P}(V)$ of the following properties:*

1. *For all $v \in V$, $v \in f(v)$.*
2. *For all $v \in V$, and all $w \in f(v)$, $f(v) = f(w)$.*
3. *With probability $9/10$, $|\{(v, w) \in E : f(v) \neq f(w)\}| \leq \varepsilon |V|$.*
4. *For all $v \in V$, $|f(v)| \leq O((d/\varepsilon^2) \cdot \log(1/\varepsilon))$.*
5. *For all v, the oracle uses $\tilde{O}((d/\varepsilon^2) + (1/\varepsilon) \cdot \log Q)$ time to compute $f(v)$, where Q is the number of previous queries to the oracle.*
6. *The function f is independent of the queries to $\mathcal{O}$.*

Proof The oracle does not begin with a fixed partition of the graph; instead it builds different parts of the partition to answer the queries it is asked. However, not every partition of the graph is permissable. Instead, we begin with a global partitioning algorithm (Algorithm 6 in this case), and the oracle simulates the algorithm on

Algorithm 6: Global-Forest-Partitioning

1 $G := T$;
2 **forall the** *vertex* v **do** $s[v] := \{v\}$;
3 **while** *there exists a node* v *of degree 1 such that* $|s[v]| < 12/\varepsilon$ **do**
4 | Let u be the neighbor of v. ;
5 | $s[u] := s[u] \cup s[v]$;
6 | Remove v from G.
7 Remove all edges incident to vertices of degree strictly greater than 2.;
8 Independently remove each remaining edge from G with probability $\varepsilon/60$.;
9 **foreach** *connected component* C *in* G **do**
10 | $V_C :=$ set of vertices in C ;
11 | $S := \bigcup_{v \in V_C}$ s[v];
12 | **if** $|V_C| \leq 2 \cdot \lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil$ **then**
13 | | **forall the** $v \in S$ **do** $f(v) := S$;
14 | **else**
15 | | **forall the** $v \in S$ **do** $f(v) := \{v\}$

different parts of the graph. We begin by analyzing the global partitioning algorithm, and then discuss how it can be simulated by the oracle.

Algorithm 6 clearly constructs a partition of the vertices of the graph. It starts by shrinking some branches into single nodes in the loop in Step 3, and then partitions the remaining vertices. It first removes edges incident to vertices of degree strictly higher than 2, which leaves us with a collection of paths. Then, it removes each of the remaining edges with some fixed probability, which (with high probability) cuts long paths into small components. If a component is small, then all the vertices of the component and all the vertices in the branches that were merged into the component constitute a single part. Otherwise, if the component is large, then each of the vertices constitutes a separate part of size 1.

Let us now bound the probability that the number of edges between different parts of the partition is greater than $\varepsilon|V|$. The edges between different parts are the edges removed by the algorithm in steps 7,8, and the edges between vertices which were in a large component in step 12, and were thus separated in step 15.

- Step 7: For each leaf v, we have $|s[v]| \geq 12/\varepsilon$. Since the sets $s[v]$ are disjoint, the total number of leaves is at most $\varepsilon|V|/12$. This implies that the number of edges

incident to nodes of degree higher than 2 is bounded by $4 \cdot \varepsilon|V|/12 = \varepsilon|V|/3$.

- Step 8: The expected number of edges removed is bounded by $\varepsilon|V|/60$. The probability that more than $\varepsilon|V|/3$ edges are removed is bounded by Markov's inequality by $1/20$.

- Step 15: What is the probability the endpoints of an edge e end up in different parts because of the decision taken in Step 15? After Step 7, the graph is a set of disjoint disconnected paths. There are two cases: either the endpoints of e are consecutive vertices on a path or both were shrunk to the same node on a path. The probability that after Step 8 more than $\lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil$ consecutive edges survived in any of the two directions on the path is at most $2 \cdot \left(1 - \frac{\varepsilon}{60}\right)^{\lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil} \leq 2 \cdot e^{-\ln(120/\varepsilon)} = \varepsilon/60$. If the number of consecutive edges that survived on the path is bounded by $\lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil$ in both directions, the size of the component is bounded by $2 \cdot \lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil$. Therefore, e connects two different parts in a decomposition due to a decision in Step 15 with probability at most $\varepsilon/60$. In expectation, we have at most $\varepsilon|V|/60$ such edges, and by Markov's inequality, the probability that more than $\varepsilon|V|/3$ edges are removed is bounded by $1/20$.

Summarizing, the number of edges connecting different parts is at most $\varepsilon|V|$ with probability at most $2 \cdot 1/20 = 1/10$ by the union bound, as required by Claim 3.

We now bound the size of each part in the partition. The size of a connected component in G that passes the test in Step 12 is at most $O(1/\varepsilon \cdot \log(1/\varepsilon))$. For each node v, the size of the set $s[v]$ is bounded by 1 (v itself) plus the number of nodes in at most d different branches that were shrunk into v. A branch can only be shrunk if the number of nodes in it is bounded by $12/\varepsilon$. The size of each $s[v]$ is hence bounded by $O(d/\varepsilon)$, and the size of any $f(v)$ is bounded by $O(1/\varepsilon \cdot \log(1/\varepsilon)) \cdot O(d/\varepsilon) = O((d/\varepsilon^2) \cdot \log(1/\varepsilon))$, as stated by Claim 4 of the theorem.

Let C be a connected component in the original graph T. Let $e = (u, v)$ be an edge in C. If we remove e from C, we get two components C_u and C_v containing u and v, respectively. The edge e is not shrunk in the loop in Step 3 if and only if the

number of vertices in both C_u and C_v is at least $12/\varepsilon$. We call such an edge *inner*. If none of the edges of C are inner, the loop in Step 3 shrinks C into a single vertex, and for every vertex v in C, $f(v)$ equals the set of vertices in C. On the other hand, if there is an inner edge in C, the component created from C by the loop in Step 3 and the sets $s[\cdot]$ for this component are unique. This explains why the order in which we consider vertices in the loop does not matter.

We now show how Algorithm 6 can be simulated locally. Let $q \in V$ be a query to $\mathcal{O}$. For each neighbor u of q, we first check if the number of nodes in the connected component that are closer to u than to q is less than $12/\varepsilon$. This can be done in $O(1/\varepsilon)$ time for each u. For each such u, we shrink the entire branch outgoing from q towards u. Then we consider the number of edges incident to q that were not shrunk. If there is 0 of them, $f(q)$ equals the all the vertices in the connected component of q. If there is more than 2 of them, then all these edges would be removed in Step 7 of Algorithm 6, and $f(q)$ consists of q and all vertices that were merged into q. If there is exactly 1 of them, say u, and the number of nodes q corresponds to is less than $1/\varepsilon$, we merge q into u, and repeat the whole procedure for u. Otherwise, q is incident to 1 or 2 inner edges. We now grow a component, which initially consists of just one vertex. We pick an unexplored inner edge, and if hasn't been done before, we remove it with probability $\varepsilon/60$ (as in Step 8 of Algorithm 6). We store the result for future references. If the edge hasn't been removed, we find the number of inner edges that are incident to its unexplored endpoint. If it is at most 2, we grow the component by adding that endpoint. We keep doing that until no unexplored inner edge remains, or the component size becomes greater than $2 \cdot \lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil$. If the component size is greater than $2 \cdot \lceil (60/\varepsilon) \cdot \ln(120/\varepsilon) \rceil$, each vertex that contributes to it constitutes its own part, i.e., $f(q) = \{q\}$. Otherwise, if the component have stopped growing, $f(q)$ consists of all vertices of the component and all vertices that were merged into the vertices of the component. The procedure clearly corresponds to what the global algorithm does.

We now assume that a dictionary operation (insertion, deletion, look-up) costs $O(\log n)$ for a collection of size n. The running time of the algorithm is then at most

$\tilde{O}((d/\varepsilon^2)+(1/\varepsilon)\cdot\log Q)$. Finally, the partition is computed such that it is independent of queries to the oracle. It is only a function of coin tosses that correspond to Step 8 of Algorithm 6. ■

3.2 Partitioning Oracle for Constant Treewidth

3.2.1 Definitions and Notations

Let $G = (V, E)$ be a graph and S be a subset of V. We write $N(S)$ to denote the set of vertices that are not in S and adjacent to at least one vertex in S. We write $\eta(S)$ to denote the *cut-size* of S which is defined to be the size of $N(S)$,

$$\eta(S) = |N(S)|$$

We write $\phi(S)$ to denote the *vertex conductance* of S which is defined as

$$\phi(S) = \frac{\eta(S)}{|S|}$$

Definition 37 *Let $G = (V, E)$ be a graph. We say that $S \subseteq V$ is a* neighborhood *of v in G if $v \in S$ and the subgraph induced by S is connected. Given $k, c \geq 1$ and $\delta \in (0, 1)$, we say that S is a* (k, δ, c)-isolated neighborhood *of $v \in V$ if S is neighborhood of v in G, $|S| \leq k$, $\eta(S) \leq c$ and $\phi(S) \leq \delta$.*

Definition 38 *Let $G = (V, E)$ be a graph and and let A be a family of neighborhoods in G. A subset $B \subseteq A$ is a* cover *of A if for every neighborhood $T \in A$*

$$T \subseteq \bigcup_{S \in B} S$$

3.2.2 Local Isolated Neighborhoods in Constant Treewidth Graphs

Lemma 39 *Let $G = (V, E)$ be a graph with treewidth bounded by h and maximum degree bounded by d. Given $\varepsilon, \delta \in (0, 1/2)$, there exists a function $g : V \to \mathcal{P}(V)$ of the following properties:*

1. *For all $v \in V$, $v \in g(v)$.*
2. *For all $v \in V$, $|g(v)| \leq \frac{14400d^3(h+1)^5}{\delta\varepsilon^2}$.*
3. *For all $v \in V$, $g(v)$ is connected.*
4. *Let $\mathcal{B}$ be the subset of V such that $v \in \mathcal{B}$ if and only if $g(v)$ is a $\left(\frac{14400d^3(h+1)^5}{\delta\varepsilon^2}, \delta, 2(h+1)\right)$-isolated neighborhood of v in G. The size of $\mathcal{B}$ is at least $(1 - \varepsilon/20)|V|$.*

Basically, Lemma 39 shows that given a bounded treewidth and bounded degree graph, for almost every vertex v in the graph, we can find an isolated neighborhood of v. The proof of the lemma is somewhat technical, and is postponed to Section 3.3.

3.2.3 Computing Isolated Neighborhoods Locally

The following lemma shows that given a vertex v in a bounded treewidth and bounded degree graph, if there exists an isolated neighborhood of v, one of such neighborhood can be found locally with a polynomial number of queries.

Lemma 40 *Let $G = (V, E)$ be a graph. There is a local algorithm that given a vertex $v \in V$ and $k, c \geq 1$ and $\delta \in (0, 1)$, makes at most dk^{c+1} queries to the graph and satisfies the following properties:*

1. *If there exists a (k, δ, c)-isolated neighborhood of v in G, the algorithm returns one such neighborhood.*
2. *Otherwise, if no such isolated neighborhood exists, the algorithm returns $\{v\}$.*

Procedure Find-Isolated-Neighborhood(v,k,δ,c)

```
1  Run BFS from v until BFS stops or exactly k vertices are visited.
2  Let S be the neighborhood found.
3  if S is a (k, δ, c)-isolated neighborhood in the original graph then
4      Output S and terminate
5  if c > 0 then
6      foreach w ∈ S \ {v} do
7          Remove w from the graph.
8          Find-Isolated-Neighborhood(v,k,δ,c − 1) ;
9          Put w back to the graph.
10 else
11     return {v}
```

Proof Consider Procedure Find-Local-Neighborhood that finds a (k, δ, c)-isolated neighborhood of v in G if at least one such neighborhood exists. The procedure is simply a brute force search that tries to search for a (k, δ, c)-isolated neighborhood S of v by guessing vertices in $N(S)$. At each iteration, it uses BFS search to explore the input graph around v until k vertices are visited or BFS stops (the latter happens when the size of the connected component that contains v is less than k). In both cases, the procedure checks if the set of visited vertices is a (k, δ, c)-isolated neighborhood of v. If it is, the procedure outputs the set and terminates. Otherwise, if the set is not a (k, δ, c)-isolated neighborhood, at least one of the vertices in the set must be in $N(S)$. The procedure guesses a vertex, removes it from the graph and tries again. Since the neighborhood S, if exists, has cut-size bounded by r, i.e., $|N(S)| \leq r$. Procedure Find-Local-Neighborhood only needs to have a recursion depth of r to search for all possibilities. Therefore, Procedure Find-Local-Neighborhood must return a (k, δ, c)-isolated neighborhood of v if such at least one such neighborhood exists and return $\{v\}$ otherwise.

Since, at each iteration, there are at most k possible guesses for a vertex in $N(S)$, the procedure only makes at most dk^{c+1} queries to the graph[1]. ■

[1]The extra factor of k comes from the execution of the BFS algorithm in each iteration.

When the values of k, δ, c are clear from context, for any $u, v \in V$, we say u *covers* v if the isolated neighborhood found by Procedure Find-Local-Neighborhood(u,k,δ,c) contains v. Observe that if Find-Local-Neighborhood(u,k,δ,c) does not terminate right after it found the first (k, δ, c)-isolated neighborhood of u in Step 4, it will find every possible (k, δ, c)-isolated neighborhood of u. Therefore, if u covers v, for any $u, v \in V$, then u must be visited by Find-Local-Neighborhood(v,k,δ,c) (without termination in Step 4) at some point. Therefore, for any $v \in V$, there are only k^{c+1} vertices that can cover v. This also implies it takes at most $dk^{2(c+1)}$ queries to find the set of vertices that cover v.

3.2.4 Cover Set

Lemma 41 *Given an undirected graph $G = (V, E)$ and a vertex $v \in V$. Let $\mathcal{A}$ be a family of neighborhoods of v such that all the neighborhoods in $\mathcal{A}$ have cut-size bounded by k. One of the following properties must be true:*

(a) $|\mathcal{A}| \leq 2^k$.

(b) There exists $S^* \in \mathcal{A}$ *such that* $S^* \subseteq \bigcup_{S \in \mathcal{A} \setminus S^*} S$.

Proof Assume that (b) is false, we will prove that (a) must be true and thus proving the lemma. For each $S \in \mathcal{A}$, there must exists a vertex $v_S \in V$ such that $v_S \notin \bigcup_{S' \in \mathcal{A} \setminus S} S'$. Let T be the BFS tree of G rooted at v. Let $P_S = (v^0 = v, v^1, \cdots, v^{n_S} = v_S)$ be the path from v to v_S in T. Let T^* be the union of all these paths, $T^* = \bigcup_{S \in \mathcal{A}} P_S$. Observe that T^* is also the subtree of T such that each leaf of T^* is a vertex v_S for some $S \in \mathcal{S}$ and vice versa, for each $S \in \mathcal{S}$, v_S is a leaf of T^*. In other words, the leaf set of T^* is exactly $\{v_S : S \in \mathcal{A}\}$.

Consider a path P_S for some $S \in \mathcal{A}$. An edge (u_1, u_2) is a branch of P_S in T^* if $(u_1, u_2) \in T^*$, $u_1 \in P_S$ and $u_2 \notin P_S$. We can show that P_S can only have at most k branches.

Consider $N(S)$ - the neighbor set of S which includes all the vertices that are not in S and adjacent to at least one vertex in S. By definition, $|N(S)| \leq k$. Also, any

path P'_S, for $S' \neq S$, must contains a vertex $N(S)$. Therefore, every branch of P_S must end up at one vertex in $N(S)$. This implies that the number of branches of P_S is at most k.

Given T^*, we can encode a path P_S for any $S \in \mathcal{A}$ as follows. Give all vertices an arbitrary order. Start from the root, each vertex in the path can be encoded by its order in the children list of its parent. In particular, if the parent has d children, then the child can be encoded with $\lceil \log d \rceil$ bits. Note that if the parent only has one child, the encoding of the child is free, i.e. no bit is needed to encode the child. The total number of bits needed is:

$$\sum_{v \in P_S \setminus \{v_S\}} \lceil \log d(v_k) \rceil$$

where $d(v)$ is the number of children of v. Since the number of branches of P_S is at most k, we have

$$\sum_{v \in P_S \setminus \{v_S\}} d(v_k) - 1 \leq k$$

Also, since $\log a \leq a - 1$ for $a \geq 1$, we have

$$\sum_{v \in P_S \setminus \{v_S\}} \lceil \log d(v_k) \rceil \leq \sum_{v \in P_S \setminus \{v_S\}} d(v_k) - 1 \leq k$$

Therefore, at most k bits is sufficient to encode P_S for every $S \in \mathcal{A}$. Thus, the size of $\mathcal{A}$ is at most 2^k. ■

Corollary 42 *Let $\mathcal{A}$ be a family of neighborhoods such that all neighborhoods in $\mathcal{A}$ have cut-size bounded by k. There exists a cover of $\mathcal{A}$ with size of at most 2^k.*

3.2.5 Proof of Theorem 35

Proof

Algorithm 7: Global-Constant-Treewidth-Partitioning

```
forall the v ∈ V do Unmark v
foreach v ∈ V do
    S_v := Find-Isolated-Neighborhood(v,k,δ,2(h + 1))
    Let r_v be a random value in (0, 1).
foreach v ∈ V in increasing order of r_v do
    U := {w ∈ S_v : w is unmarked }.
    forall the w ∈ U do f[w] := U
    Mark all vertices in U.
Output f.
```

Algorithm 8: Local-Bounded-Treewidth-Partitioning

```
Q_q := the set of vertices that cover q.
Let u be the vertex in Q_q such that r_u is smallest.
S_u := Find-Isolated-Neighborhood(u,k,δ,2(h + 1))
if S_u = {u} then
    return S_u.
Let P = ∅
foreach w ∈ S_u do
    Q_w := the set of vertices that cover w.
    Let u_w be the vertex in Q_w such that r_{u_w} is smallest.
    if u = u_w then
        P := P ∪ {w}
return P.
```

Consider the global partitioning Algorithm 7 where

$$\delta := \tilde{O}(\varepsilon/(2^{2(h+1)}h^2d^2))$$

$$k := \tilde{O}(2^{2(h+1)}d^7h^7\frac{1}{\varepsilon^3})$$

The algorithm clearly constructs a partition of the vertices in the graph. It starts with all vertices in the graph unmarked. For each vertex $v \in V$, the algorithm tries to find a $(k, \delta, 2(h+1))$-isolated neighborhood of v in G. If one such neighborhood is found, S_v is set to that neighborhood. Otherwise, if no such neighborhood exists, S_v is set to $\{v\}$. Next the algorithm starts partitioning the graph. It considers the vertices in a random order. For each vertex v in that order, all the unmarked vertices

in S_v constitute a single part and get marked.

Clearly, at the end of the execution of Algorithm 7, all vertices must be marked. Therefore, $f(v)$ is well defined for all $v \in V$. Also, observe that when $f(v)$ is defined, the value of function f for all other vertices in $f(v)$ is also set to $f(v)$. Therefore, Claims 1 and 2 of the theorem hold for the function f computed by Algorithm 7.

Let us bound the probability that the number of edges between different parts is greater than $\varepsilon|V|$. By Lemma 39 with the lemma's ε set to ε/d, there exists a function $g : V \to \mathcal{P}(V)$ of the following properties:

1. For all $v \in V$, $v \in g(v)$.
2. For all $v \in V$, $|g(v)| \leq k$.
3. For all $v \in V$, $g(v)$ is connected.
4. Let $\mathcal{B}$ be the subset of V such that $v \in \mathcal{B}$ if and only if $g(v)$ is a $(k, \delta, 2(h+1))$-isolated neighborhood of v in G. The size of $\mathcal{B}$ is at least $(1 - \varepsilon/20d)|V|$.

Let us group the edges between different parts into two sets: the set of edges incident to at least one vertex in $\mathcal{B}^c = V \setminus \mathcal{B}$ and the set of edges with both endnodes in $\mathcal{B}$. Observe that the total number of edges in the first set is at most $\frac{\varepsilon}{20}|V|$. It remains to bound the number of edges in the second set. Consider a vertex $v \in \mathcal{B}$. Let Q_v be the set of vertices that cover v and let $m_v = |Q_v|$. As it is shown in Section 3.2.3, $m_v \leq k^{2h+3}$. Let $q_1, q_2, \cdots, q_{m_v} \in Q_v$ be the sequence of vertices that cover v in increasing order of r, i.e., $r_{q_1} \leq r_{q_2} \leq \cdots$. For each $k \in \{1, 2, \cdots, m_v\}$, let $S_v^{(k)} = \{S_{q_1}, S_{q_2}, \cdots, S_{q_k}\}$, where S_{q_i} is the isolated neighborhood found by Procedure Find-Isolated-Neighborhood starting from q_i. Note that, since r is random, $S_v^{(k)}$ and q_k are random variables for all $k \in \{1, 2, \cdots, m_v\}$.

Given $u, w \in \mathcal{B}$, we say u *marks* v if $v \in S_u$ and v is not marked before u is considered. Also, we say a vertex $u \in \mathcal{B}$ is *marking* if u marks some vertex in the graph. It is clear from the definition that, for any $k \in \{1, 2, \cdots, m_v\}$, q_k is marking if and only if $S_{q_k} \not\subseteq \bigcup_{i=1}^{k-1} S_{q_i}$. This implies that S_{q_k} must be a member of every cover of

$S_v^{(k)}$. By Corollary 42, there exists a cover $B_k \subseteq S_v^{(k)}$ such that $|B_k| \leq 2^{2(h+1)}$. Thus,

$$\begin{aligned} Pr\left[q_k \text{ is marking}\right] &= \sum_{S_v^{(k)}} Pr\left[q_k \text{ is marking} | S_v^{(k)}\right] Pr\left[S_v^{(k)}\right] \\ &\leq \sum_{S_v^{(k)}} Pr\left[S_{q_k} \in B_k | S_v^{(k)}\right] Pr\left[S_v^{(k)}\right] \\ &= \sum_{S_v^{(k)}} \frac{|B_k|}{|S_v^{(k)}|} Pr\left[S_v^{(k)}\right] \\ &\leq \frac{2^{2(h+1)}}{k} \end{aligned}$$

Let the number of marking vertices in Q_v be a_v. We have,

$$E[a_v] = \sum_{k=1}^{m_v} Pr[q_v \text{ is marking}] \leq \sum_{k=1}^{m_v} \frac{2^{2(h+1)}}{k} \approx 2^{2(h+1)} \log m_v \leq 2^{2(h+1)} \cdot (2h+3) \cdot \log k$$

Let $M \subseteq \mathcal{B}$ be the set of marking vertices in the graph, we have

$$\sum_{u \in M} |S_u| = \sum_{u \in \mathcal{B}} a_u$$

Thus,

$$E[\sum_{u \in M} |S_u|] \leq 2^{2(h+1)} \cdot (2h+3) \cdot \log k |\mathcal{B}|$$

Note that, for each marking vertex u, the number of edges that have exactly one end in S_u is at most $d|S_u|\delta$. This is also a upper bound for the number of edges going out of the part that contains u in the partition. Therefore, the expected number of edges with both ends in $\mathcal{B}$ is at most

$$O\left(\delta d 2^{2(h+1)} \cdot (2h+3) \cdot \log k\right) |\mathcal{B}| \leq \frac{\varepsilon}{100}|V|$$

Thus, by Markov inequality, the probability that the number of edges in the second set is greater than $\frac{\varepsilon}{10}|V|$ is $1/10$. Therefore, the probability that the total number of edges in both set is less than $\varepsilon|V|$ is at least $9/10$, as required by Claim 3.

We now bound the size of each part in the partition. Observe that each part is either a single vertex or a subset of some $(k, \delta, 2(h+1))$-isolated neighborhood. Therefore, the size of each part is at most k, as stated by Claim 4 of the theorem.

We now show how Algorithm 7 can be simulated locally. Consider the local Algorithm 8 that given a query $q \in V$, computes $f[q]$. The local algorithm starts with computing the set of vertices that cover q. Then among the vertices in this set, it finds the vertex u with smallest r-value. Clearly, u is the vertex that marks q, and thus, $f[q]$ is a subset of S_u. Next the local algorithm considers each vertex in S_u and checks whether that vertex is also marked by u. If it is, the vertex should also be included in $f[q]$. Clearly local Algorithm 8 computes exactly the same set $f[q]$ as it was computed by the global Algorithm 7. Let us bound the number of queries to the input graph that Algorithm 8 makes to answer query q:

- As it is shown in Section 3.2.3, finding Q_q, the set of vertices that cover q, takes only dk^{4h+6} queries to the input graph.
- By Lemma 40, finding S_u only takes at most dk^{2h+3} queries to the input graph.
- Finally, for each $w \in S_u$, checking whether w is marked by u also takes dk^{4h+6} queries to the input graph. Therefore, it takes at most dk^{6h+9}) queries to find the subset of vertices in S_u that are marked by u.

Summarizing, the oracle only has to make at most $dk^{6h+9} = (\frac{d}{\epsilon})^{O(h)}$ queries to the input graph to answer a query about f. We now assume that a dictionary operation cost $O(\log n)$ for a collection of size n. The running time of the oracle to answer a query about f is then at most $\frac{d}{\epsilon})^{O(h)} \cdot \log Q)$, as stated by Claim 5 of the theorem.

Finally, the partition is computed such that it is independent of queries to the oracle. It is only a function of coin tosses that correspond to Step 5 of Algorithm 7[2].

■

[2]As a technical requirement to make the argument valid, we also assume that the loop in Step 6 of Procedure Find-Isolated-Neighborhood considers vertices of the input graph in an order that is independent of queries to the oracle.

3.3 Proof of Lemma 39

3.3.1 A Strong Partition for Forests

We now show that for any forest T, there exists a partition of T such that the cut-sizes of most of the parts in the partition are at most 2.

Lemma 43 *Let $T = (V_T, E_T)$ be a forest with maximum degree bounded by $d \geq 2$. Given $\varepsilon, \delta \in (0, 1/2)$ and let $k = \frac{240d^2}{\delta\varepsilon}$, there exists a partitioning function $f : V \to \mathcal{P}(V)$ of the following properties:*

1. *For all $v \in V$, $v \in f(v)$.*
2. *For all $v \in V$, and all $w \in f(v)$, $f(w) = f(v)$.*
3. *For all $v \in V$, $|f(w)| \leq k$.*
4. *For all $v \in V$, the subgraph of T inducted by $f(v)$ is connected.*
5. *Let $\mathcal{C}$ be the subset of V_T such that $w \in \mathcal{C}$ if and only if $f(w)$ is a $(k, \delta, 2)$-isolated neighborhood of w in T. The size of $\mathcal{C}$ is at least $(1 - \varepsilon/60)|V|$.*

Proof Consider the global partitioning Algorithm 9[3]. The algorithm consists of four parts:

1. **Step 3-7:** Small branches are shrunk into single nodes. For each node $v \in V_G$, $s[v]$ is the set of vertices in V_T that are contracted to v.
2. **Step 8-12:** In this part, the algorithm partitions the set of vertices that are contracted to nodes degree higher than 2. For each node of degree higher than 2, all the vertices in each branch that is shrunk to that node constitutes a single part and the node itself constitutes a part of size 1. Once all the node with degree higher than 2 are removed, the graph that remains only contains disjoints paths and isolated nodes.

[3]Note that, since we only need to show the existence of a partition of required properties for T, it is not necessary to show how to simulate Algorithm 9 locally.

3. **Step 13-20:** In this part, the algorithm partitions the set of vertices that are contracted to nodes in the paths. For each end node of a path in the graph, if its weight is less $2/\delta$, the node is contracted to its only neighbor. Otherwise, if the node's weight is at least $2/\delta$, all the vertices in the branches that are shrunk to that node constitutes a single part.

4. **Step 21-22:** Finally, for each isolated node that remains in the graph, the set of vertices that are contracted to that node continues a single part.

Clearly, from the construction of function f, Claims 1, 2 and 4 of the Lemma 43 holds.

Let us bound the size of each part in the partition constructed by Algorithm 9. First, observe that for each part in the partition, all the vertices in that part are contracted to the same node. Also, observe that the weight of a node in T is a most $d \cdot k_1 = k$. Therefore, the size of each part in the partition is also bounded by k, as required by Claim 3.

Finally, we now show that the size $\mathcal{C}$ is at least $(1 - \varepsilon/60)|V|$.

- Let us bound the number of vertices in T which are in parts of cut-size greater than 2. Observe that for any $v \in V_T$, the cut-size of $f(v)$ is only greater than 2 if v is also a node of degree higher than 2 in 9 of Algorithm 9. Since after the first part of Algorithm 9, the weigh of each leaf node in G is at least k_1, the number of leaves in G at Step 8 is at most $\frac{1}{k_1}|V|$. This also implies that the number of nodes with degree higher that 2 in G at Step 8 is bounded by $\frac{1}{k_1}|V| \leq \frac{\varepsilon}{240}|V|$.

- We now bound the number of vertices in T which are in parts of conductance greater than δ. Given $v \in V_T$, observe that $\phi(f(v))$ is only greater than δ in one of the following cases:

  - v is a node of degree higher than 2 in Step 8 of Algorithm 9. There are at most $\frac{1}{k_1}|V| \leq \frac{\varepsilon}{240}|V|$ vertices of this type.

Algorithm 9: Stronger-Tree-Partitioning

```
1  k_1 := 240d/(δε)
2  G := T
   /* Phase 1:  Contract leaf node of weight less than k_1          */
3  forall the vertex v do s[v] := {v}
4  while there exists a node v of degree 1 such that |s[v]| < k_1 do
5  |  Let u be the neighbor of v.
6  |  s[u] := s[u] ∪ s[v]
7  |  Remove v from G.
   /* Phase 2:  Remove node with degree greater than 2              */
8  foreach node v of degree greater than 2 do
9  |  foreach w ∈ s[v] such that (w, v) ∈ E_T do
10 |  |  f(w) := s[w]
11 |  f(v) := {v}
12 |  Remove v from G.
   /* Phase 3:  Partitioning paths                                  */
13 while there exists a node v in G with degree 1 do
14 |  if |s[v]| ≥ 2/δ then
15 |  |  forall the w ∈ s[v] do f[w] := s[v_i]
16 |  |  Remove v from G.
17 |  else
18 |  |  Let u be the only neighbor of v in G.
19 |  |  s[u] := s[u] ∪ s[v]
20 |  |  Remove v from G
   /* Phase 4:  Partitioning isolated nodes                         */
21 while there exists an isolated node v do
22 |  forall the w ∈ s[v] do f[w] := s[v]  Remove v from G.
```

- v is contracted to a node u, and u is vertex of degree higher than 2 in Step 8. In this case, $\phi(f(v)) > \delta$ if and only if the size of the branch that was shrunk to u and contains v, is less than $1/\delta$. Since for each high degree node u, there are at most d such small branches. Therefore, the total number of vertices in T of this type is at most $\frac{dk_1}{\delta}|V| \leq \frac{\varepsilon}{240}|V|$.

- v is contracted to a node u where u is the last node that remains after partitioning a path in G at Step 13 and the weight of u is smaller than $2/\delta$. Observe that before Step 8, u is not a leaf node in G, since otherwise u would be contracted another node. Therefore, u must be a neighbor of

a node with degree higher than 2 in Step 8. Since the number of nodes with degree greater than 2 at Step 8 is less than $\frac{1}{k_1}|V|$, there are at most $\frac{d}{k_1}|V|$ nodes likes u in G. This implies that the number of vertices like v is at most $\frac{d}{k_1\delta}|V| \leq \frac{\varepsilon}{240}$.

Summarizing, the number of of vertices in T which are in parts of conductance greater than δ is at most $\frac{\varepsilon}{80}|V|$.

By the Union bound, there are at least $(1-\varepsilon/60)|V|$ vertices in T which are in parts of conductance at most δ and of cut-size at most 2. Combining this with the Claim 3 of the lemma, we have $|\mathcal{C}| \geq (1-\varepsilon/60)|V|$, as stated in Claim 5. ■

3.3.2 Some Properties of Tree Decomposition

Definition 44 *Let $(\mathcal{X}, \mathcal{T})$ be a tree decomposition of $G = (V, E)$, where $\mathcal{X} = (X_1, X_2, \cdots, X_m)$ is a family of subsets of V and $\mathcal{T}$ is a tree whose nodes are the subset X_i's.*

1. *We say $(\mathcal{X}, \mathcal{T})$ is* edge-overlapping *if for any $X_i, X_j \in \mathcal{X}$ such that $(X_i, X_j) \in \mathcal{T}$, $X_i \cap X_j \neq \emptyset$.*

2. *We say $(\mathcal{X}, \mathcal{T})$ is* minimal *if for any $X_i \in X$ and any $x_i \in X_i$, removing x_i from X_i will make $(\mathcal{X}, \mathcal{T})$ no longer a tree decomposition of G.*

3. *We say $(\mathcal{X}, \mathcal{T})$ is* non-repeated *if for any $(X_i, X_j) \in \mathcal{T}$, $X_i \neq X_j$.*

Let $\mathcal{T}_1$ be a subtree of $\mathcal{T}$. We write $\mathcal{X}_{|\mathcal{T}_1}$ to denote the set of nodes in $\mathcal{T}_1$.

Let $\mathcal{S}$ be a subset of $\mathcal{X}$ and let v be a vertex in V. We write $V_{|\mathcal{S}}$ to denote $\bigcup_{Z \in \mathcal{S}} Z$. Similarly, let $\mathcal{T}_1$ be a subtree of $\mathcal{T}$, we write $V_{|\mathcal{T}_1}$ to denote $\bigcup_{Z \in \mathcal{X}_{|\mathcal{T}_1}} Z$.

Let v be a vertex in V, we say v *appears* in a subset $\mathcal{S} \subseteq \mathcal{X}$ if $v \in V_{|\mathcal{S}}$, and v *appears* in subtree $\mathcal{T}_1$ of $\mathcal{T}$ if $v \in V_{|\mathcal{T}_1}$.

Let $e = (u, v) \in E$ be an edge in G, we say a subset $\mathcal{S} \subseteq \mathcal{X}$ *witnesses* e if there exists a node $Z \in \mathcal{S}$ such that Z contains both u and v. Similarly, we say a subtree $\mathcal{T}_1$ of $\mathcal{T}$ *witnesses* e if there exists a node $Z \in \mathcal{X}_{|\mathcal{T}_1}$ such that Z contains both u and v.

Lemma 45 *Given a graph $G = (V, E)$ with treewidth h. There is a tree decomposition of G of width h which is edge-overlapping, minimal and non-repeated.*

Proof Let $(\mathcal{X}, \mathcal{T})$ be a tree decomposition of G of width h. Starting from $(\mathcal{X}, \mathcal{T})$, we can obtain a edge-overlapping, minimal and non-repeated tree decomposition of G by repeatedly applying the following operations.

1. If there exists a pair of nodes $X_i, X_j \in \mathcal{X}$ such that $(X_i, X_j) \in \mathcal{T}$ and $X_i = X_j$, remove X_j and connect all neighbors of X_j to X_i.

2. If there exists a pair of nodes $X_i, X_j \in \mathcal{X}$ such that $(X_i, X_j) \in \mathcal{T}$ and $X_i \cap X_j = \emptyset$, remove the edge (X_i, X_j) from $\mathcal{T}$.

3. If there exists a node $X_i \in X$ and a vertex $x_i \in X_i$ such that $(\mathcal{X}, \mathcal{T})$ is still a tree decomposition of G after removing x_i from X_i, then remove x_i from X_i.

■

Lemma 46 *Given a graph $G = (V, E)$ and $(\mathcal{X}, \mathcal{T})$ is a non-repeated tree decomposition of G of width h. Let $\mathcal{T}_1$ be a subtree of $\mathcal{T}$.*

$$\frac{|V_{|\mathcal{T}_1}|}{h+1} \leq |\mathcal{X}_{|\mathcal{T}_1}| \leq |V_{|\mathcal{T}_1}|$$

Proof The first inequality is straightforward. Since each vertex in $V_{|\mathcal{T}_1}$ appears at least once in $\mathcal{X}_{|\mathcal{T}_1}$ and each set in $\mathcal{X}_{|\mathcal{T}_1}$ contains at most $h+1$ vertices, $|\mathcal{X}_{|\mathcal{T}_1}| \geq \frac{|V_{|\mathcal{T}_1}|}{h+1}$.

The second inequality can be proved by induction on the size of $\mathcal{X}(\mathcal{T}_1)$.

- **Base case** $\mathcal{T}_1$ is an isolated node X_0 in $\mathcal{T}$. Thus, $|\mathcal{X}_{|\mathcal{T}_1}| = 1 \leq |V_{|\mathcal{T}_1}| = |X_0|$.

- **Inductive case** $\mathcal{T}_1$ is a connected component of size $k > 1$ in $\mathcal{T}$. Let $X_0 \in \mathcal{X}_{|\mathcal{T}_1}$ be a leaf node in $\mathcal{T}_1$. By the induction hypothesis,

$$|\mathcal{X}_{|\mathcal{T}_1} \setminus \{X_0\}| \leq |\bigcup_{Z \in \mathcal{X}_{|\mathcal{T}_1} \setminus \{X_0\}} Z| \tag{3.1}$$

and let X_1 be X_0's neighbor in $\mathcal{T}_1$. Since $(\mathcal{X}, \mathcal{T})$ is non-repeated, there must be some vertex $v \in V$ such that $v \in X_0 \setminus X_1$. By the definition of tree decomposition, the set $\{X \in \mathcal{X} : v \in X\}$ is a connected component of $\mathcal{T}$. Therefore, for every $Z \in \mathcal{X}_{|\mathcal{T}_1} \setminus \{X_0\}$, $v \notin Z$. This implies that,

$$|\bigcup_{Z \in \mathcal{X}_{|\mathcal{T}_1} \setminus \{X_0\}} Z| \leq |V_{|\mathcal{T}_1} \setminus \{v\}| = |V_{|\mathcal{T}_1}| - 1 \tag{3.2}$$

From (3.1) and (3.2), we have $|\mathcal{X}_{|\mathcal{T}_1}| \leq |V_{|\mathcal{T}_1}|$. ■

Lemma 47 *Given a graph $G = (V, E)$ with maximum degree bounded by d and $(\mathcal{X}, \mathcal{T})$ is a edge-overlapping and minimal tree decomposition of G of width h. The maximum degree of $\mathcal{T}$ is bounded by $(h+1) \cdot d$.*

Proof Let $X_0 \in \mathcal{X}$ be a node in $\mathcal{T}$. Consider a neighbor X_1 of X_0. Since $(\mathcal{X}, \mathcal{T})$ is edge-overlapping, there must exist some $x \in V$ such that $x \in X_0 \cap X_1$. In addition, if we consider $\mathcal{T}$ as a tree rooted at X_0, there is an unique subtree of X_0 which contains X_1. Since $(\mathcal{X}, \mathcal{T})$ is minimal, this subtree must witness some edge $(x, y) \in E$, $y \in V$, that is not witnessed by any other subtree of X_0,. Therefore, we can correspond each subtree of X_0 to an unique edge of a vertex in X_0. Since there are at most $h+1$ vertices in X_0 and each of them has at most d edges, X_0 can only have at most $d(h+1)$ neighbors. ■

Proof [Proof of Lemma 39]

By Lemma 45, there exists a a edge-overlapping, minimal and non-repeated tree decomposition $(\mathcal{X}, \mathcal{T})$ of G of width h. By Lemma 47, $\mathcal{T}$ is a tree with maximum degree bounded by $d \cdot (h+1)$. By Lemma 43, with the lemma's ε set to $\varepsilon/(h+1)$, d set to $d(h+1)$ and δ set to $\frac{\delta\varepsilon}{60d(h+1)}$, there exists a partitioning function $f : \mathcal{X} \to \mathcal{P}(\mathcal{X})$ such that:

1. For all $X \in \mathcal{X}$, $X \in f(X)$.
2. For all $X \in \mathcal{X}$, and all $Y \in f(X)$, $f(Y) = f(X)$.

3. For all $X \in \mathcal{X}$, $|f(X)| \leq \frac{14400d^3(h+1)^4}{\delta\varepsilon^2}$.

4. For all $X \in \mathcal{X}$, the subgraph of $\mathcal{T}$ inducted by $f(X)$ is connected.

5. The size of $\mathcal{C}$, the subset of nodes in $\mathcal{X}$ such that for every $X \in \mathcal{C}$, $f(X)$ is a $(\frac{14400d^3(h+1)^4}{\delta\varepsilon^2}, \frac{\delta\varepsilon}{60d(h+1)}, 2)$-isolated neighborhood of X in T, is at least $(1 - \varepsilon/60(h+1))|\mathcal{X}|$.

Let $\mathcal{P}$ be the partition represented by the function f. To simplify the notation, we say a part P in $\mathcal{P}$ is *good* if P is a $(\frac{14400d^3(h+1)^4}{\delta\varepsilon^2}, \frac{\delta}{60d(h+1)}, 2)$-isolated neighborhood. Otherwise, P is *bad*. Let $\mathcal{P}_{good}$ be the set of good parts in $\mathcal{P}$. Also, let $\mathcal{P}_{bad} = \mathcal{P} \setminus \mathcal{P}_{good}$ be the set of bad parts in $\mathcal{P}$. Let $A_{bad} = \bigcup_{P \in \mathcal{P}_{bad}} V_{|P}$, i.e., A_{bad} is the set of vertices in V that appear in at least one bad part in $\mathcal{P}$. Let $A_{good-neighbor} = \bigcup_{P \in \mathcal{P}_{good}} V_{|N(P)}$ is the set of vertices in V that appear in the neighbor of at least one good part[4]. Finally, let $A_{good} = V \setminus (A_{bad} \cup A_{good-neighbor})$ is the set of vertices in V such that for every $v \in A_{good}$, v only appears in exactly one part in $\mathcal{P}$ and that part is good.

Let us construct the function g as follows. For each $v \in V$, if $v \notin A_{good}$, set $g(v)$ to $\{v\}$. Otherwise, if $v \in A_{good}$, set $g(v)$ to the connected component that contains v, in the subgraph of G induced by A_{good}.

It is clear from the construction that the Claims 1 and 3 of Lemma 39 hold for the function g.

Let us bound the size of $g(v)$.

- If $v \notin A_{good}$, then it is clear from the definition of g, $|g(v)| = 1$.

- Otherwise, if $v \in A_{good}$, let P be the part in $\mathcal{P}$ that v appears. It is clear from the construction of g that $V_{|N(P)} \cap A_{good} = \emptyset$. Therefore, $g(v) \subseteq V_{|f(P)|}$. Thus, $|g(v)| \leq |V_{|N(P)}| \leq (h+1)|f(P)| \leq \frac{14400d^3(h+1)^5}{\delta\varepsilon^2}$.

In both cases, the size of $g(v)$ is bounded by $\frac{14400d^3(h+1)^5}{\delta\varepsilon^2}$, as stated by Claim 2 of the lemma.

[4]Recall that $N(P)$ denotes the set nodes in $\mathcal{T}$ that are not in P and adjacent to at least one node in P.

Finally, we now show that $|\mathcal{B}| \geq \frac{\varepsilon}{30}|V|$. Consider a vertex $v \in A_{good}$. Let P be the part in $\mathcal{P}$ that contains v. Observe that every edge going out of $g(v)$ must end up in $V_{|N(P)}$. Therefore,

$$\eta(g(v)) \leq V_{|N(P)} \leq (h+1)|N(P)| \leq 2(h+1) \tag{3.3}$$

Let $A_{high-conductance} \subseteq A_{good}$ be the set of vertices in A_{good} such that $v \in A_{high-conductance}$ if and only if $\phi(g(v)) > \delta$. Also, let $A_{low-conductance} = A_{good} \setminus A_{high-conductance}$.

- Observe that for each bad part P in $\mathcal{P}$, all nodes in P are in $\mathcal{C}^c$,

$$|A_{bad}| \leq |\bigcup_{Z \in \mathcal{C}^c} Z| \leq (h+1)|\mathcal{C}^c| \leq \frac{\varepsilon}{60}|\mathcal{X}| \leq \frac{\varepsilon}{60}|V|$$

- Observe that for each good part P in $\mathcal{P}$, $|N(P)| \leq \frac{\delta\varepsilon}{60d(h+1)}|P|$. Thus,

$$|V_{|N(P)}| \leq (h+1)|N(P)| \leq \frac{\delta\varepsilon}{60d}|P|$$

  This implies that $|A_{good-neighbor}| \leq \frac{\delta\varepsilon}{60d}|V| \leq \frac{\varepsilon}{60}|V|$.

- Since every edge going out of $g(v)$ must end up in $A_{good-neighbor}$, the total of edges going out of $g(v)$ for all $v \in A_{good}$ is at most $d|A_{good-neighbor}|$. Thus, the size of $A_{high-conductance}$ must be bounded by

$$\frac{d}{\delta}|A_{good-neighbor}| \leq \frac{\varepsilon}{60}|V|$$

From (3.3), it can be deduced every vertex in $A_{low-conductance}$ is also in $\mathcal{B}$. Therefore,

$$|\mathcal{B}| \geq |A_{low-conductance}| \geq |V| - |A_{bad}| - |A_{good-neighbor}| - |A_{high-conductance}| \geq \frac{\varepsilon}{20}|V|$$

■

Bibliography

[1] Noga Alon, Paul D. Seymour, and Robin Thomas. A separator theorem for graphs with an excluded minor and its applications. In *STOC*, pages 293–299, 1990.

[2] Reid Andersen and Yuval Peres. Finding sparse cuts locally using evolving sets. *CoRR*, abs/0811.3779, 2008.

[3] A. Atserias. On digraph coloring problems and treewidth duality. *European Journal of Combinatorics*, 29(4):796–820, 2008.

[4] Mihai Badoiu, Artur Czumaj, Piotr Indyk, and Christian Sohler. Facility location in sublinear time. In *ICALP*, pages 866–877, 2005.

[5] Itai Benjamini, Oded Schramm, and Asaf Shapira. Every minor-closed property of sparse graphs is testable. In *STOC*, pages 393–402, 2008.

[6] Manuel Blum, Michael Luby, and Ronitt Rubinfeld. Self-testing/correcting with applications to numerical problems. In *STOC*, pages 73–83, 1990.

[7] H.L. Bodlaender. A tourist guide through treewidth. *Developments in Theoretical Computer Science*, page 1, 1994.

[8] H.L. Bodlaender and A.M.C.A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 2007.

[9] Bernard Chazelle, Ronitt Rubinfeld, and Luca Trevisan. Approximating the minimum spanning tree weight in sublinear time. In *ICALP*, pages 190–200, 2001.

[10] Artur Czumaj, Asaf Shapira, and Christian Sohler. Testing hereditary properties of nonexpanding bounded-degree graphs. *SIAM J. Comput.*, 38(6):2499–2510, 2009.

[11] Artur Czumaj and Christian Sohler. Estimating the weight of metric minimum spanning trees in sublinear-time. In *STOC*, pages 175–183, 2004.

[12] Andrzej Czygrinow and Michal Hańćkowiak. Distributed algorithm for better approximation of the maximum matching. In *COCOON*, pages 242–251, 2003.

[13] Andrzej Czygrinow, Michal Hańćkowiak, and Edyta Szymańska. A fast distributed algorithm for approximating the maximum matching. In *ESA*, pages 252–263, 2004.

[14] Andrzej Czygrinow, Michal Hanckowiak, and Wojciech Wawrzyniak. Fast distributed approximations in planar graphs. In *DISC*, pages 78–92, 2008.

[15] Gábor Elek. L^2-spectral invariants and convergent sequences of finite graphs. *Journal of Functional Analysis*, 254(10):2667 – 2689, 2008.

[16] Eldar Fischer. The art of uninformed decisions: A primer to property testing. *Science*, 75:97–126, 2001.

[17] M. R. Garey and David S. Johnson. The rectilinear Steiner tree problem is NP complete. *SIAM Journal of Applied Mathematics*, 32:826–834, 1977.

[18] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[19] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

[20] Oded Goldreich. Combinatorial property testing (a survey). In *In: Randomization Methods in Algorithm Design*, pages 45–60. American Mathematical Society, 1998.

[21] Oded Goldreich and Dana Ron. Property testing in bounded degree graphs. *Algorithmica*, 32(2):302–343, 2002.

[22] Avinatan Hassidim, Jonathan A. Kelner, Huy N. Nguyen, and Krzysztof Onak. Local graph partitions for approximation and testing. In *FOCS*, pages 22–31, 2009.

[23] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[24] Piotr Indyk. Sublinear time algorithms for metric space problems. In *STOC*, pages 428–434, 1999.

[25] David S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. Syst. Sci.*, 9(3):256–278, 1974.

[26] Kyomin Jung and Devavrat Shah. Algorithmically efficient networks. Manuscript, `http://web.kaist.ac.kr/~kyomin/Algorithmically_Efficient_Networks.pdf`. Last accessed on May 19, 2010.

[27] Richard M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[28] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *SODA '06: Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 980–989, New York, NY, USA, 2006. ACM.

[29] Richard J. Lipton and Robert Endre Tarjan. A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36:177–189, 1979.

[30] Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM J. Comput.*, 9(3):615–627, 1980.

[31] László Lovász. On the ratio of optimal integral and fractional covers. *Discrete Mathematics*, 13:383–390, 1975.

[32] Sharon Marko and Dana Ron. Distance approximation in bounded-degree and general sparse graphs. In *APPROX-RANDOM*, pages 475–486, 2006.

[33] Sharon Marko and Dana Ron. Approximating the distance to properties in bounded-degree and general sparse graphs. *ACM Transactions on Algorithms*, 5(2), 2009.

[34] Huy N. Nguyen and Krzysztof Onak. Constant-time approximation algorithms via local improvements. In *FOCS*, pages 327–336, 2008.

[35] Michal Parnas and Dana Ron. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci.*, 381(1-3):183–196, 2007.

[36] Seth Pettie and Peter Sanders. A simpler linear time 2/3-epsilon approximation for maximum weight matching. *Inf. Process. Lett.*, 91(6):271–276, 2004.

[37] N. Robertson and P.D. Seymour. Graph minors. III. Planar tree-width. *Journal of Combinatorial Theory, Series B*, 36(1):49–64, 1984.

[38] N. Robertson and P.D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

[39] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63(1):65 – 110, 1995.

[40] Neil Robertson and Paul D. Seymour. Graph minors. XX. Wagner's conjecture. *Journal of Combinatorial Theory, Series B*, 92(2):325 – 357, 2004. Special Issue Dedicated to Professor W.T. Tutte.

[41] Dana Ron. Property testing. In *Handbook of Randomized Computing, Vol. II*, pages 597–649. Kluwer Academic Publishers, 2000.

[42] Ronitt Rubinfeld. Sublinear time algorithms. *SIGACT News*, 34:2003, 2003.

[43] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly-linear time graph partitioning. arXiv:0809.3232v1, 2008.

[44] Yuichi Yoshida and Hiro Ito. Testing outerplanarity of bounded degree graphs. In *RANDOM*, pages 393–402, 2010.

[45] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum~matchings. In *STOC*, pages 225–234, 2009.

[46] Yuichi Yoshida, Masaki Yamamoto, and Hiro Ito. An improved constant-time approximation algorithm for maximum~matchings. In *STOC*, pages 225–234, 2009.

[47] David Zuckerman. On unapproximable versions of NP-complete problems. *SIAM J. Comput.*, 25(6):1293–1304, 1996.