25

Space-Time Tradeoffs for Graph Properties

by

Yevgeniy Dodis

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

Master of Science in Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1998

Author ....
Department of Electrical Engineering and Computer Science
May 8, 1998

Certified by ....
Madhu Sudan
Professor
Thesis Supervisor

Accepted by ....
Arthur Smith
Chairman, Departmental Committee on Graduate Students

Space-Time Tradeoffs for Graph Properties

by

Yevgeniy Dodis

Submitted to the Department of Electrical Engineering and Computer Science
on May 8, 1998, in partial fulfillment of the
requirements for the degree of
Master of Science in Computer Science

Abstract

Given a graph property P, we study the tradeoff between the pre-processing space and the query time in the following scenario. We are given a graph G, a boolean function family $\mathcal{F}$, and a parameter s which indicates the amount of space available for storing a data structure D that contains information about G. The data structure D satisfies the constraint that the value of each cell c in D corresponds to an application of some function drawn from $\mathcal{F}$. Our queries are of the form: "Does the subgraph G_X induced by the vertex set X satisfy property P?" For various settings of $\mathcal{F}$ and s, this model unifies many well-studied problems. At one extreme, when the space s is unrestricted, we study the generalized decision tree complexity of evaluating P on the entire graph G itself; each tree node stores a function in $\mathcal{F}$ applied to some subset of edges. A special case of this model is the famous AKR conjecture (where $\mathcal{F}$, contains merely the identity function $g(x) = x$) which states that any non-trivial monotone graph property is evasive. At the other extreme, when the function family $\mathcal{F}$ is unrestricted, our problem is an example of the classical *static data structure problem* and we examine the cell probe complexity of our problem. We study graph properties across this broad spectrum of computational frameworks. A central thesis of our work is that "polynomial preprocessing space yields only a negligible (polylogarithmic) speedup". While proving such a result for an unrestricted $\mathcal{F}$ is unlikely, we provide formal evidence towards this thesis by establishing near-quadratic (optimal in many cases) lower bounds under a variety of natural restrictions. Our results are built upon a diverse range of techniques drawn from communication complexity, the probabilistic method and algebraic representations of boolean functions. We also study the problem from an algorithmic viewpoint and develop a framework for designing algorithms that efficiently answer queries using bounded space. We conclude with a study of space-time tradeoffs in an abstract setting of general interest that highlights certain structural issues underlying our problem.

Thesis Supervisor: Madhu Sudan
Title: Professor

Acknowledgments

The author would like to thank Sanjeev Khanna for collaborating on the research presented in this thesis. The author also thanks Shafi Goldwasser, Tom Leighton, Dan Spielman, Madhu Sudan, Avi Wigderson and Peter Winkler for their valuable comments and suggestions. Special thanks to my family for their continuing support.

Contents

List of Figures

Chapter 1

Introduction

We study space-time tradeoffs for graph properties. Specifically, given a graph property P, we study the tradeoff between the pre-processing space and the query time in the following scenario. We are given a graph G, a boolean function family $\mathcal{F}$, and a parameter s which indicates the amount of space available for storing a data structure $D = \{c_1, c_2, ..., c_s\}$ that contains information about G. The data structure D satisfies the constraint that the value of each cell c_i in D corresponds to an application of some $g \in \mathcal{F}$ to a subset of edge variables. Our queries are of the form "Does the subgraph G_X of G induced by the vertex set X satisfy the property P?" We refer to this problem as the *induced subgraph problem.* A given query q is answered by probing cells of D and the time t spent in answering q is the number of probes made. Our *goal* is to study the worst-case time $t = \mathcal{T}_{\mathcal{F},s}(P)$ needed to answer queries as a function of the family $\mathcal{F}$, space s, and the number n of vertices in the input graph G. Our framework elegantly unifies several well-studied questions concerning graph properties and data structures, hitherto studied in isolated settings. At one extreme, when the space s is unrestricted, the time $t = \mathcal{T}_{\mathcal{F}}(P)$ measures the decision tree complexity of evaluating P on the entire graph G itself; each tree node stores a function in $\mathcal{F}$ applied to some subset of edges. A well-known special case of this model is the famous AKR conjecture (where $\mathcal{F}$, contains merely the identity function

$g(x) = x$) which states that *any non-trivial monotone graph property is evasive.*[1] The conjecture is proven to within a constant factor i.e. every non-trivial monotone graph property is known to be almost evasive [13, 5].

At the other extreme, when the function family $\mathcal{F}$ is unrestricted, our problem becomes an example of the classical *static data structure problem* and the time $t = \mathcal{T}_s(P)$ measures the cell probe complexity of induced subgraph problem. The static data structure problem is defined in general for an arbitrary function $f : Y \times Q \mapsto \{0,1\}$ (f is the property P in our case), where the first input $y \in Y$ is static (the graph G), $|y| = m$, and the second input $q \in Q$ is a dynamic query (the induced subset X), $|q| = n$. The objective is to determine the worst case number of probes needed to compute $f(y, q)$ using a data structure with s cells; the problem has been well-studied in the literature ([1], [2], [8], [9], [10], [14]). Still, no explicitly defined function f is known for which $t = \omega(n)$ is proven when space $s = poly(n)$. Showing such a super linear bound for an NP function f (an NP graph property in our case) would *unconditionally* separate NP from the class of read-twice branching programs ([11]); a long-standing question in complexity theory. Notice that the case when both space s and family $\mathcal{F}$ are unrestricted is trivial as every query can be answered in unit time.

We study graph properties across this broad spectrum of computational frameworks, shedding some light on the combinatorial structure of many fundamental properties. Our techniques for obtaining the lower bounds rely on the probabilistic method, information-theoretic arguments such as the ones used in communication complexity, algebraic methods for representation of boolean functions and combinatorial arguments that exploit the structure of minimal "certificates" for a given property. The diverse nature of these techniques highlights various structural aspects of graph properties that are interesting in their own right. A thesis central to our work is that for any function family $\mathcal{F}$, the time $\mathcal{T}_{\mathcal{F},s}(P) = \Omega(n^2/polylog(s))$ for any evasive graph property P, that is, *polynomial preprocessing space yields only a neg-*

[1] A function $h : \{0,1\}^m \mapsto \{0,1\}$ is said to be evasive (almost evasive) if any decision tree for h has depth m ($\Omega(m)$).

ligible (poly-logarithmic) speedup. While proving such a result for unrestricted $\mathcal{F}$ is unlikely, as indicated in the preceding discussion, our work provides formal evidence towards this thesis under a variety of natural restrictions. In what follows, we describe more precisely our results, motivations and techniques, organized across this broad spectrum.

Restricted Space, Unrestricted Function Families: Since $\mathcal{F}$ is unrestricted, we denote time simply by $\mathcal{T}_s$. The best known lower bound for an explicitly defined function, as indicated above, is $\Omega(n/\log s)$. We match this bound for the induced subgraph problem when the underlying property is any non-trivial monotone graph property. We also show the same result for non-monotone property PARITY. The result is shown using the connection between static data structure problems and the communication complexity model, discovered by Miltersen [8], and essentially all known bounds can be viewed as an application of this technique [11]. In contrast, a simple counting argument shows that almost every function $f : \{0,1\}^m \times \{0,1\}^n \mapsto \{0,1\}$ requires $t = \Omega(m)$ even with exponential space $s = 2^{n-1}$ [8]. Thus, a central problem is to construct an explicit family of functions which is "hard" to speed-up. We believe that induced subgraph problem for evasive graph properties is a candidate function family towards this end.

We also study the *asymmetric communication complexity* [11] of our problem. In this setup, instead of measuring simply the total number of bits exchanged, we measure the number of bits sent by Alice and Bob individually. An $[A, B]$-protocol is a protocol where Alice sends at most A bits and Bob sends at most B bits. A lower bound of $[A, B]$ means that either Alice must send $\Omega(A)$ bits or Bob must send $\Omega(B)$ bits in order to compute the function. We show that when the underlying graph property is PARITY (of edges), there is a $[n^2/\log n, n]$ lower bound. We conjecture that the *asymmetric communication complexity* of induced subgraph problem for evasive P is $[n^2, n]$. Thus, if Bob does not send almost the entire subset X to Alice, Alice must send Bob almost the entire graph G.

Restricted Space, Restricted Function Families: Our interest here is to study space-time tradeoffs when c_i's are drawn from some general, yet not arbitrary, function family $\mathcal{F}$ i.e. the measure $\mathcal{T}_{\mathcal{F},s}$. We remark here that if $\mathcal{F}$ equals the family of all monotone functions, it is "equivalent" in power to the *unrestricted setting* above. This follows from the fact that any (non-monotone) function can be efficiently simulated by monotone functions using the idea of "slice functions" [4]. In general, many seemingly restrictive families can capture large classes of functions via efficient simulation. Observe that even when restricting to some simple function families $\mathcal{F}$, many evasive properties are now expressible as a single cell function that takes as input the entire graph. Consider the following evasive property P: Is G an empty graph? Let $\mathcal{F}$ be simply the family of OR functions. Clearly, a single OR can express the property on a given G. On the other hand, a cell storing an OR of all the edges in the graph is of no use in determining whether an induced subgraph G_X satisfies P. Intuitively speaking, the cells that are sensitive to "many" edges are useful for answering only very few queries, while "short" cells might be good for many queries but we need to read many of them to answer a "large" query. Indeed, we prove that if $\mathcal{F}$ is restricted to only AND and OR functions, $\mathcal{T}_{\mathcal{F},s}(P) = \Omega(n^2/\log^2 s)$ for any evasive property P. Moreover, for many natural properties we show that the bound is tight. We then non-trivially extend this result to α-CNF, α-DNF (for constant α) and symmetric function families and show that $\mathcal{T}_{\mathcal{F},s}(P) = \Omega(n^2/polylog(s))$ for any evasive proeprty P. We also study the following curious question: What is the time complexity of induced subgraph problem for a property P when the data structure can only contain answers about whether an induced subgraph of the input graph has property P. The interest in this question arises from the observation that indeed many properties can be efficiently computed on an entire graph by simply evaluating the property on various subgraphs. While any single query can now be answered in one probe, we show $\Omega(n^2/\log^2 s)$ bound for any non-trivial "uniform" monotone property. The central technique used in our results is a probabilistic argument which shows that for any data structuring strategy, there exists an input graph G such that (a) it "stabilizes" the value of any cell that is sensitive to many variables (where "many" will depend on space s), and (b) still leaves a large subset X "untouched" such that one can reveal the edges of G_X via an evasive strategy.[2] Since the evasive game is now only sensitive to cells with small number of variables, we get our desired bounds. A variation of this technique has been used by Hellerstein *et al* [4] to study graph reachability problem on bipartite graphs. An important distinguishing aspect of their problem is that output to a query comprises of n bits rather than a single bit of information.

Unrestricted Space, Restricted Function Families: At this end of the spectrum, our study essentially reduces to the following question: "how efficiently a graph property P (on the entire graph) can be expressed using primitives from a function family $\mathcal{F}$?" In other words, what is the decision tree complexity of P when the nodes of the tree store functions from $\mathcal{F}$? We denote this measure as $\mathcal{T}_{\mathcal{F}}(P)$. Hajnal *et al* [3] studied the measure $\mathcal{T}_{\text{OR}}$ for specific graph properties such as connectivity and bipartiteness. We derive lower bounds on the measure $\mathcal{T}_{\mathcal{F}}(P)$ when $\mathcal{F}$ is the family AND, OR or the family of small degree polynomials, and P is a non-trivial monotone property. Specifcally, we initiate our study in an algebraic framework where we determine the least degree of a multinomial over $\mathbb{Z}_2$ that expresses a given monotone property P. Building on ideas presented in the classic paper by Rivest and Vuillemin [13], we establish the following general result. For any *transitive function*[3] f on $m = p^k$ variables s.t. $f(\bar{0}) \neq f(\bar{1})$, the degree of the (unique) multilinear polynomial Q over $\mathbb{Z}_p$ that computes f, is m. An implication of this result is that every non-trivial monotone graph property requires a multinomial of degree $\Omega(n^2)$ over $\mathbb{Z}_2$, implying the AKR conjecture to within a constant factor. Thus if each function in a family $\mathcal{F}$ can be represented by a multinomial of degree at most d, we show $\mathcal{T}_{\mathcal{F}}(P) = \Omega(n^2/d)$, e.g. $\mathcal{T}_{\text{XOR}}(P) = \Omega(n^2)$ for the XOR family. On the other hand, this approach does not work for the two most natural extensions of the AKR setup, namely the AND and OR function families, since the degree of these functions can be as large as $\Omega(n^2)$. We develop general techniques for studying these families by essentially reducing their

[2] An answering strategy that forces any probing algorithm to read all the edges of the graph

[3] See Section 5.1.1 for definition

decision tree complexity to a certain measure of standard decision trees. We then develop several techniques to lower bound this measure and obtain $\Omega(n^2)$ bound for many properties. These techniques involve examining the combinatorial structure of "graph certificates" and design of general answering strategies for monotone graph properties. One of the techniques that we examine in detail is based on the analysis of two greedy strategies for playing decision games in the AKR setup, namely, answer "no" ("yes") unless forced to say "yes" ("no"). Our study of these strategies might be of independent interest. As an aside, we demonstrate the exponential gap between adaptive and oblivious complexities of computing the parity function using the family of threshold functions.

Upper Bound Results: So far we saw that near-quadratic lower bounds can be shown under certain restrictions — in line with our belief that polynomial preprocessing space does not yield significant speedups. We now approach the induced subgraph problem from an algorithmic viewpoint in an effort to improve upon the naive $O(n^2)$ bound. As our lower bound results might already suggest, it is unlikely to achieve speedups better than a *polylog*(s) factor. However, as we shall see, even achieving small speedups requires non-trivial new ideas. Our approach here is to develop canonical techniques for improving upon the trivial bounds. To begin with, we develop a representation scheme, called the *standard graph representation* which allows us to "efficiently"[4] perform useful operations on the graph induced by any query set. The basic idea of this respresentation is to partition the graph into small clusters and exhaustively store information within each cluster. To construct information about any given induced subgraph, we simply combine together relevant pieces from within each cluster. As an example, our representation scheme can be used to speed up construction of breadth-first (depth-first) forests — an integral part of many graph algorithms. We also develop a more efficient algorithm to compute the transitive closure of an induced subgraph; this allows us to efficiently compute properties such as connecivity and bipartiteness. Finally, we define the notion of "speedup preserving"

[4] Here "efficient" and "speedup" signify only modest *polylog*(s) factor improvements.

reductions for induced subgraph problem which enable us to transform an efficient algorithm for one property to one for another. An interesting aspect of our results is that simple function families such as AND, OR and threshold functions, seem to be all that one can use for a broad range of properties. We note here that a clustering based representation scheme has also been used by Hellerstein *et al* [4] for a static data structure problem on bipartite graphs.

The (s,t)-Spanning Set Problem: Finally, we examine space-time tradeoffs in the following abstract setting that is interesting in its own right and further highlights the combinatorial structure underlying our problem. Assume that we are given a set M of p^m elements with some operation $\star$ and M has some "basis" B of size m, i.e. every element of M is uniquely "expressible" using elements of B. B is the smallest set that can express every "query" element in $M-$ possibly using all m elements in the process. Now suppose that we are willing to store more than m elements in B and seek to express any element of M using at most $t << m$ elements drawn from B. If we find such B of size s, this B is called a *(s,t)-spanning set* for M. More generally, we will be interested in (s,t)-spanning set for some subset $W \subseteq M$. The question as usual is what is the optimal tradeoff between s and t. We start our study by establishing some general bounds. In particular, we show that $t = \Theta(m/\log_p s)$ when $W = M$, that is, only logarithmic speedup is possible using a polynomial size spanning set. Also, for any random $W \subseteq M$ with $|W| = p^n$, we show that almost certainly $t = \Theta(m/\log_p s)$ for $s = o(p^n)$, i.e. asymptotically we cannot do better than building a spanning set for entire M. Next we focus on the case when the set $M = \{0,1\}^m$ and is equipped with the union operation. We show that for any monotone function f, the non-deterministic as well as oblivious complexity of evaluating f using AND/ OR function families, can be characterized in terms of (s,t)-spanning set. This characterization is then used to obtain lower bounds on the non-deterministic and oblivious variants of $\mathcal{T}_{\text{AND},s}(P)/\mathcal{T}_{\text{OR},s}(P)$ and $\mathcal{T}_{\text{AND}}(P)/\mathcal{T}_{\text{OR}}(P)$ for a monotone property P. For many properties, including connectivity and bipartiteness, we show near-quadratic bounds even when non-determinism is allowed. As an aside, we obtain a separation between adaptive versus oblivious algorithms for AND/ OR-restricted data structures.

Organization: Chapter 2 gives notation and preliminaries, while chapters 3 through 7 correspond to the preceding paragraphs in order.

Chapter 2

Notation and Preliminaries

Let $f : Y \times Q \mapsto \{0,1\}$ be a function we are trying to compute, $y \in Y$, $q \in Q$, $|y| = m$, $|q| = n$ and $n \leq m$.

2.1 Generalized Cell Probe Model

The general cell probe model model of computation is the following. We pre-process the *static* input y by computing s pre-specified functions $g_1, \ldots, g_s : Y \mapsto \{0,1\}^b$. The results we store in s *cells* of our database D. Given a *dynamic* input q we compute $f(y,q)$ by adaptively reading the cells D trying to minimize the number of *probes* we make. Thus, the time w.r.t. to a given D is the worst case number of probes we have make over all dynamic inputs. Unless otherwise stated, we will assume that the cell size $b = 1$. Let $\mathcal{F}$ be some function family.

Definition 1 ($\mathcal{F}$-restricted Data Structure) *An* $\mathcal{F}$-restricted Data Structure *is a database D where every cell corresponds to an application of some function in $\mathcal{F}$ to an ordered subset of bits from the static input.*

We will talk about $\mathcal{F}$-restricted data structures for a variety of families of $\mathcal{F}$. Some examples of function families that we study include AND, OR, XOR and α-CNF/α-DNF (i.e. every clause in the formula has at most α literals) families. For instance, the family AND consists of functions $\{g_i\}_{i \geq 1}$ where g_i denotes conjunction of i variables. $\mathcal{F}$ consisting solely of the identity function is called *trivial.* We will also talk about the family of *threshold* or symmetric monotone functions. A threshold function $T_{p,q}$ (F_{p_q}) is a function on p variables that is true iff at least (less than) q out of p input variables are set to true. Clearly, $F_{p_q} = \overline{T_{p,q}}$, so we can w.l.o.g. restrict our attention to "upward" functions $T_{p,q}$ when talking about the family of threshold functions.

Thus, given a family $\mathcal{F}$ and space s, the task is to find the $\mathcal{F}$-restricted data structure with the smallest query time over all the dynamic inputs. This measure is denoted $\mathcal{T}_{\mathcal{F},s}(f)$.

Deterministic and Non-deterministic Computation: In the model described above the algorithm deciding which probe to make next is deterministic. We can also talk about non-deterministic models when the probing scheme is allowed to make guesses. Since the complexity measure is just the number of probes made, the only non-determinism is in guessing the right cells to read. There are 3 related non-deterministic models: verifying $f = 1$ (analog of NP), verifying $f = 0$ (analog of coNP) and guessing and verifying f (analog of NP$\cap$coNP). In verifying $f = 1$ (0) the probing scheme has to reject all the guesses when $f = 0$ (1) and accept at least once when $f = 1$ (0). The time complexity is the maximum number of cells read along any non-deterministic branch. Guessing and verifying f can be thought as making guesses and outputting 0, 1 or $failed$. When $f = z$, all non-failed branches should output z and there should be at least one such branch. It is easy to see that the time to guess and verify f is the maximum of the time to verify $f = 1$ and $f = 0$. In general, we will use $\mathcal{T}$ to refer to deterministic time measures and $\mathcal{NT}$ - to non-deterministic. When talking about verifying $f = 1$ (0) we use superscript ^{1} (^{0}), e.g. $\mathcal{NT}^1_{\mathcal{F},s}(f)$.

Oblivious Computation: Rather than letting the probing scheme determine the next cell to read based on the values of the previous cells (this computation is called *adaptive*), we can talk about oblivious computation. In this model, given a query q, the algorithm has to specify in advance all t cells that it will read from the database

D. The time, as usual, is the number of cells read. We remark that obliviousness is only a limitation for deterministic computation, since a non-deterministic probing algorithm can (obliviously) make all possible guesses and reject the ones that it did not like at the end. We use superscript ^{obl} when talking about oblivious time measures, e.g. $\mathcal{T}^{obl}_{\mathcal{F},s}(f)$.

Classical Data Structure Problem (Unrestricted $\mathcal{F}$): When the function family $\mathcal{F}$ is unrestricted, we can store absolutely arbitrary information about our static input and the problem of computing f becomes the *classical data structure problem* introduced by Yao [15]. This is the hardest case for proving strong lower bounds. We omit the subscript $_{\mathcal{F}}$ when the function family is unrestricted, e.g. $\mathcal{T}_s(f)$, $\mathcal{NT}^1_s(f)$.

2.2 Decision Tree Complexity (Unrestricted s)

When the space s is unrestricted, we can probe at any moment the value $g(y)$ for *any* $g \in \mathcal{F}$.

Definition 2 (Decision Tree) *Let $h(y_1, \ldots, y_m) : \{0,1\}^m \mapsto \{0,1\}$ be any function and $\mathcal{F}$ be a family of functions from $\{0,1\}^m \mapsto \{0,1\}$. A* decision tree *$T$ for h with respect to $\mathcal{F}$ is a binary tree where the leaves are labeled by 0 or 1, each internal node is labeled by some function from $\mathcal{F}$ and has two outgoing edges labeled by 0 and 1. Given an input $y = y_1 \ldots y_m$ we traverse T starting from the root and being at an internal node labeled by $g \in \mathcal{F}$ we always take an outgoing edge labeled by the value of $g(y)$. For every input y the value of the leaf we reach has to be $h(y)$. The* (deterministic) decision tree complexity of h w.r.t. $\mathcal{F}$, *$\mathcal{T}_{\mathcal{F}}(h)$ is the minimum depth of a decision tree w.r.t. $\mathcal{F}$ that computes h.*

It is easy to see that when the space is unrestricted, the time to compute f is the decision tree complexity w.r.t. indicated family $\mathcal{F}$ of computing $f|_{q=q_0}$ for the *worst-case* query q_0. We will usually abuse the notation and simply talk about decision tree complexity of f itself, identifying f with its restriction to the worst-case query. Thus, we still use the notation $\mathcal{T}_{\mathcal{F}}(f)$ to measure time when the space is unrestricted. The reader is encouraged to view f as a function of a single variable y in such cases.

The non-deterministic and oblivious computation still make sense for unrestricted space: we can view the database as consisting of all possible functions in $\mathcal{F}$ applied to all possible subsets of input bits. We also omit the subscript $_s$ in such cases, e.g. $\mathcal{NT}_{\mathcal{F}}(f)$, $\mathcal{T}^{obl}_{\mathcal{F}}(f)$.

Simple Decision Trees, Certificates and Evasive Functions: A *simple* decision tree is a decision tree w.r.t. to trivial $\mathcal{F}$, i.e. every node is labeled by some input bit of y. The measures $\mathcal{T}_{\mathcal{F}}(\_), \mathcal{NT}_{\mathcal{F}}(\_), \mathcal{NT}^{z}_{\mathcal{F}}(\_)$ are denoted simply by $D(\_), N(\_), N^z(\_)$ in this case, and are called decision tree complexity, non-deterministic decision tree complexity, z-non-deterministic decision tree complexity (i.e. reference to $\mathcal{F}$ is omitted), where $z \in \{0, 1\}$. When we talk about a decision tree without mentioning $\mathcal{F}$ explicitly, we refer the trivial $\mathcal{F}$.

Definition 3 (Evasive Function) *A function $h(y_1, \ldots, y_m) : \{0,1\}^m \mapsto \{0,1\}$ is called* (almost) evasive, *if $D(f) = m$ $(\Omega(m))$.*

Evasive function has the property that any decision tree computing it has a long computational path, where an adversary can always force the probing scheme to ask about all the input bits of y. Such adversarial strategy is called an *evasive strategy* and the whole interaction between the adversary and any probing algorithm — an *evasive game.* Sometimes we loosely use the terms evasive strategy/game even for almost evasive functions, corresponding to the strategy/game that forces to read almost all the input bits. More generally, $D(h)$ is the value of the *decision game* between the algorithm that tries to compute h by asking the adversary about individual bits of y.

$N^z(h)$ also has an alternative natural meaning.

Definition 4 (Certificate) *A* z-certificate *C_z for h is a partial assignment that already fixes the value of h to z. If no smaller partial assignment is a z-certificate, then the C_z is called* min-z-certificate. *The size $|C_z|$ of C_z is the number of variables it sets.*

If we let $\mathcal{C}^z$ be the collection of all min-z-certificate for h, then it immediately follows that $N^z(h) = \max_{C_z \in \mathcal{C}^z} |C_z|$, i.e. the z-non-deterministic decision tree complexity of h is the size of the largest min-z-certificate of h, which is also the largest number of bits of y that is necessary to guess in order to verify that $h(y) = z$.

We will only talk about certificates for *upward monotone* f, in which case a min-z-certificate always consists of a (minimal) subset of variables set to z that force the function to z. We will often identify this certificate with the corresponding subset of variables, since all the variables in this subset are set to z anyway.

The following is a folklore result:

Theorem 1 *For any h, $D(h) \leq N^0(h)N^1(h) \leq (N(h))^2$.*

Thus, in the world of simple decision trees, "P = NP $\cap$ coNP".

2.3 Communication Complexity Model

The two-party communication complexity model was introduced in a seminal paper by Yao [16]. Again, we want to compute $f : Y \times Q \mapsto \{0,1\}$, but instead of static input $y \in D$ and dynamic input $q \in Q$, we have two parties Alice and Bob who are given y and q resp. They engage in a protocol, where they send each other some messages about their inputs, until both of them are able to compute $f(y,q)$. The protocol proceeds in rounds where at each round Alice sends a message to Bob and vice versa. There is no space limitation involved and the only measure of complexity is the total number of bits t that they sent to each other. The messages are arbitrary length and are sent in adaptive fashion one after another. The smallest worst-case number of bits sufficient to be exchanged in order to compute f, is the *deterministic communication complexity* $D^c(f)$. Here the trivial protocol would be for one player to send his entire input to the other one. If $|y| = m$, $|q| = n$, and $m >> n$, we clearly get $D^c(f) \leq n + 1$, corresponding to Bob sending Alice entire q and Alice replying with 1 bit answer $f(y,q)$. An equivalent way to view a protocol is to view it as a general decision tree where every internal node is labeled by an *arbitrary function* of either y

only (Alice's turn) or q only (Bob's turn), and has two outgoing edges labeled by 0 and 1. The leaves are labeled by 0 and 1. An execution is simply the traversal of this tree until we reach a leaf — the value of $f(y, q)$. The total number of bits exchanged worst-case is the depth of the tree. The depth of the smallest depth protocol tree for f is exactly $D^c(f)$. We can also define non-deterministic communication complexities, where Alice and Bob can make non-deterministic guesses. However, this measures are easier to define combinatorially.

Definition 5 *Let $z \in \{0, 1\}$. An input (y, q) s.t. $f(y, q) = z$ is called a z-*input*. A* rectangle *is any set $R = Y' \times Q'$, where $Y' \subseteq Y$, $Q' \subseteq Q$. R is z-*monochromatic *if it consists entirely of z-inputs. The z-*cover number *$C^{c,z}(f)$ is the smallest number of z-monochromatic rectangles covering (possibly with intersections) all z-inputs of f. We define z-non-deterministic communication complexity $N^{c,z} = \log C^{c,z}(f)$ and $N^{c,f} = \log(C^{c,0}(f) + C^{z,1}(f))$. Finally, we let the* answer matrix *M_f of f to be the $|Y| \times |Q|$ matrix with $M_f(y, q) = f(y, q)$.*

An important observation in communication complexity is the fact that after each round of communication the set of pair (y, q) that could produce the current communication transcript always forms a rectangle $R = Y' \times Q'$, where Y' is the set of y which are consistent with the messages that Alice sent to Bob and similarly for Q'. In particular, a leaf labeled by $z \in \{0, 1\}$ in the protocol tree must correspond to a z-monochromatic rectangle. This observation is a key to two basic and yet powerful techniques for showing lower bounds on communication complexity.

The *fooling-set* technique says that if one can construct a set of input pairs H s.t. for all $(y, q) \in H$, $f(y, q) = z$ for some fixed $z \in \{0, 1\}$, but for any distinct $(y_1, q_1), (y_2, q_2) \in H$ we have that at least one of $f(y_1, q_2)$ and $f(y_2, q_1)$ is different from z, then $D^c(f) \geq N^{c,z}(f) \geq \log |H|$. The reason is that no two points in F can be in the same z-monochromatic rectangle of f.

The second technique, provably more powerful but generally harder to analyze, is the *rank bound* technique. It examines the structure of the answer matrix M_f. For any field F (typically, we take the largest possible, say the reals), the rank bound states that $D^c(f) \geq \log(2 \cdot rank_F(M_f) - 1)$. The technique is so powerful that it is a major open problem in communication complexity of whether $D(f) = (\log(rank_F(M_f))^{O(1)}$ for any f.

An analog of Theorem 1 holds for communication complexity as well.

Theorem 2 *For any $f : Y \times Q \mapsto \{0,1\}$, $D^c(f) = O(N^{c,0}(f) N^{c,1}(f)) = O((N^c(f))^2)$.*

Asymmetric Communication Complexity: Miltersen *et al* [11] introduced a finer notion of asymmetric communication complexity. Instead of measuring the total number of bits sent by both players, we might measure the number of bits sent by each player. Assuming $n << m$, for example, it is true that $D^c(f) \leq n+1$ since Bob can send all his n bits to Alice. What, if Bob only sends $o(n)$ bits? Can we show that in this case Alice has to send significantly more than n bits? Define an $[A,B]$-protocol to be a protocol where Alice sends Bob at most A bits and Bob sends Alice at most B bits. An asymmetric complexity lower bound $[A,B]$ means that either Alice has to send $\Omega(A)$ bits, or Bob has to send $\Omega(B)$ bits in order to compute f.

One of the main techniques for showing lower bounds on the asymmetric complexity introduced by [11] is the *richness technique.* As in the rank bound, we look at the answer matrix M_f of f. If at least u rows of M_f contain v 1-entries each (this property is called (u,v)-richness), but there is no $(\frac{u}{2^{A+B}} \times \frac{v}{2^A})$ 1-monochromatic rectangle for f, then there is no $[A,B]$-protocol for computing f. For example, to show $[m,n]$ lower bound on asymmetric complexity, one needs to show that there are 2^{cm} inputs y s.t. for each of them there are at least 2^{dn} inputs q making $f(y,q) = 1$. However, there is no rectangle in M consisting of all 1-entries of dimension $2^{c'm} \times 2^{d'n}$, where $0 < c' < c \leq 1$, $0 < d' < d \leq 1$. For a much more detailed introduction to communication complexity we refer the reader to [6].

2.4 Graph Properties and the Induced Subgraph Problem

Let P be a graph property which means that P is a subset of n-vertex graphs invariant under relabeling of vertices, i.e. $P(G) = 1$ implies $P(G') = 1$, for any G' isomorphic to G. We usually denote a graph by $G = (V, E)$. Given $X \subseteq V$, we let $G_X = (X, E_X)$ be the subgraph of G induced by X and $Edges_X$ be the set of possible edges between vertices in X. The induced subgraph problem is a function $f_P(G, X) = P(G_X)$. Thus, the static input is the graph G of size $m = \binom{n}{2}$ and the dynamic input in $X \subseteq V$, it takes n bits to describe X, and the objective is to compute P on the induced subgraph G_X. We slightly abuse the notation and apply our time complexity measures to P rather than to f_P, e.g. $\mathcal{T}_{\mathcal{F},s}(P)$ rather than $\mathcal{T}_{\mathcal{F},s}(f_P)$.

We will more often than not talk about a very rich class *monotone graph properties*, which means that *addition* of edges cannot make the property go from true to false. There are two trivial monotone properties corresponding to the property being always true/false. We will exclude these 2 trivial properties from our consideration, so the term "monotone property" always refers to a "non-trivial monotone property", which means that P is false on the empty graph and true on the complete graph.

As we observed, when the space s is unrestricted, we talk about decision tree complexity of computing the property P on the worst dynamic input. For the induced subgraph problem this means that we compute the time to evaluate the property P on the entire graph G. Thus induced subgraphs come into play only when the space is bounded.

When $\mathcal{F}$ is trivial, we talk about simple decision tree complexity of a given graph property P. The famous Aandrea-Karp-Rosenberg conjecture (or the AKR conjecture) states that *every (non-trivial) monotone graph property P is evasive.* The conjecture is proven to within a constant factor by Rivest and Vuillemin [13], i.e. every every monotone property is almost evasive. Kahn *et al* [5] proved exact evasiveness for n being a prime power. A related result by Yao [17] states that *every (non-trivial) monotone bipartite property is evasive.* Here bipartite property means that it is defined only on bipartite graphs. Thus, the AKR conjecture provides the "base" case for our study of the generalized cell probe complexity of the induced subgraph problem, demonstrating $\mathcal{T}_{\mathcal{F}}(P) = \Omega(n^2)$ for the trivial $\mathcal{F}$ for nay monotone P.

Even though a lot of our results hold for general $f : Y \times Q \mapsto \{0,1\}$, we will apply them mainly to the induced subgraph problem. Thus, we will frequently refer to several common properties. These include CONNECTIVITY, BIPARTITE (G is bipartite), FOREST (G is acyclic), CLIQUE (G is a complete graph), NON-EMPTY (G has at least one edge), NO-ISOLATED-VERTEX (G has no isolated vertices), PARITY (the parity of the number of edges of the graph), MAJORITY (G has more edges than non-edges), k-CLIQUE (G has a k-clique). Except for PARITY, all mentioned properties are monotone or anti-monotone (i.e. their negation is monotone).

We use the notation $\mathcal{C}^z$ to refer to the set of all min-z-certificates of a graph property P on the entire vertex set. For example, for CONNECTIVITY a min-0-certificate is a (missing) complete bipartite graph (this ensures that G is disconnected), so $\mathcal{C}^0$ is the set of complete bipartite graphs, $N^0(P) = \Omega(n^2)$. $\mathcal{C}^1$ is the set of all trees, so $N^1(P) = n - 1$. For NO-ISOLATED-VERTEX, any $C_0 \in \mathcal{C}^0$ is a (missing) star ($N^0(P) = n - 1$), while $C_1 \in \mathcal{C}^1$ is a union of mini-stars of size at least two each that partition the n vertices of G. Such union of stars is the "minimal" way to ensure that there are no isolated vertices, since removal of any edge creates an isolated vertex. Thus, $N^1(P) = n - 1$ as well. For CLIQUE, $\mathcal{C}^1$ consists only of the complete graph ($N^0(P) = \binom{n}{2}$), while $\mathcal{C}^0$ consists of $\binom{n}{2}$ single edge graphs ($N^1(P) = 1$). For NON-BIPARTITE, $C_0 \in \mathcal{C}^0$ is a union of two (missing) cliques that partition the vertex sets ($N^0(P) = \Omega(n^2)$), while $C_1 \in \mathcal{C}^1$ is any odd cycle, $N^1(P) \leq n$.

Chapter 3

Unrestricted Function Families

3.1 Unconditional Lower Bounds

We start by giving lower bounds when the function family $\mathcal{F}$ is unrestricted, i.e. bounds on $\mathcal{T}_s(P)$. For no static data structure problem for an explicitly defined function, a bound better than $\Omega(\frac{n}{\log s})$ is known. Miltersen *et al* [11] showed that obtaining an $\omega(n)$ bound for a function in NP would separate NP from read-twice branching programs — a long standing open problem. In contrast, we have the following counting result of Miltersen [8]. For almost every random function, any scheme with space $s \leq 2^{n-1}$ requires $t = \Omega(m) >> \frac{n}{\log s}$; observe that only doubling the space to $s = 2^n$ yields $t = 1$. The result can be extended to non-deterministic computation as well.

Lemma 1 *[8] For a random function $f : \{0,1\}^m \times \{0,1\}^n \mapsto \{0,1\}$ and for any $s \leq 2^{n-1}$, with high probability $\mathcal{T}_s(f) \geq m - \mathrm{loglog}\, s - 1 >> m/2$ and $\mathcal{NT}_s(f) \geq \mathcal{NT}_s^1(f) \geq \frac{m-1}{\log s}$.*

Proof: We use a counting argument. The number of databases $c_1, \ldots, c_s : \{0,1\}^m \mapsto \{0,1\}$ we can make is $2^{2^m s}$. For each of 2^n queries we can make a separate decision tree of depth t with vertices labeled by some c_i. Number of such decision trees of depth t is at most s^{2^t}. Thus, the overall number of databases and computations we can have is $2^{2^m s}(s^{2^t})^{2^n}$. If we can compute all $2^{2^{m+n}}$ functions with space s and time

t, we must have

$$2^{2^m s}(s^{2^t})^{2^n} \geq 2^{2^{m+n}} \Longleftrightarrow 2^m s + 2^{t+n}\log s \geq 2^{m+n} \Longleftrightarrow \frac{s}{2^n} + 2^{t-m}\log s \geq 1 \qquad (3.1)$$

As $\frac{s}{2^n} \leq \frac{1}{2}$, the claim follows. The non-deterministic bound is identical except the number of non-deterministic decision trees is $2^{2^t\binom{s}{t}} \leq 2^{s^t}$ rather than s^{2^t}, since we can non-deterministically read any of the $\binom{s}{t}$ tuples of cells and have 2^t possible answers for the cells we read and 2 possible answers for each of these $2^t\binom{s}{t}$ traces. Proceeding as before, the non-deterministic bound follows. ■

So the best achievable lower bound with the current machinery is far from the correct bound in almost all cases. Moreover, as was observed in [11], essentially all lower bounds for the static data structure problem are obtainable via the connection between the cell probe model and the communication complexity model, discovered by [8]. For generality, we will talk about cell probe schemes with cell size b, i.e. each stored function $c_i : \{0,1\}^m \mapsto \{0,1\}^b$. When making a probe, we read all b bits at once. Given any cell probe scheme for computing f with cell size b, space s and query time t, we can construct a communication protocol computing f where there are t rounds of communication, and in each round Bob (the probing scheme) sends $\log s$ bits (the index of the cell to be read) and Alice (the database) sends b bits (the content of a cell). We simply view a probe as sending a $\log s$ bit index to the cell that we want to read and the value read as sending b bits back.

Lemma 2 *[8] Any cell probe scheme with space s, cell size b and time t for computing f yields a $[tb, t\log s]$-protocol for computing f. For the case of $b = 1$, we have $D^c(f) = O(t\log s)$. Hence, $\mathcal{T}_s(f) = \Omega(\frac{D^c(f)}{\log s})$ and $\mathcal{NT}_s(f) = \Omega(\frac{N^c(f)}{\log s})$*

Since $D^c(f) \leq n + 1$, the best possible data structure bound obtainable this way would be $t = \Omega(\frac{n}{\log s})$. We next show that such a lower bound can indeed be established for any non-trivial monotone graph property.

Theorem 3 *For any non-trivial monotone graph property P, $D^c(P) \geq N^c(P) = \Omega(n)$. Thus $\mathcal{T}_s(P) \geq \mathcal{NT}_s(P) = \Omega(\frac{n}{\log s})$*

Proof: We use the fooling-set method; fix a non-trivial monotone property P. Let $K[C,X]$ be the graph on the vertex set X that solely consists of a clique $C \subseteq X$. We will slightly abuse the notation and use $K[j,X]$ to denote an arbitrary member of the collection $\{K[C,X] \mid C \subseteq X, |C| = j\}$. Denote by $T(P)$ the least i such that any $(n/2)$-vertex graph consisting solely of an i-clique satisfies P. Fix V to be a set of n vertices. Now consider first the case $j = T(P) \geq n/4$. Take any $A \subseteq V$ with $|A| = \frac{n}{2} + j$ and let $B = V \backslash A$, so $|B| = \frac{n}{2} - j$. For any $C \subseteq A$ of size j, we define a fooling-set pair $\langle G = K[C,V], X = C \cup B \rangle$. We observe that $|X| = \frac{n}{2}$ and the size of our fooling-set F is $\binom{|A|}{j} = \binom{\frac{n}{2}+j}{j} \geq (1 + \frac{n}{2j})^j \geq 2^{n/4}$, as $j \geq n/4$. To show F is a fooling-set we check that $P(G_X) = P(K[C,X]) = 1$, as $|C| = j = T(P)$, and for any $C^1 \neq C^2$, $|C^1| = |C^2| = j$, we have $P(G^2_{X_1}) = P(K[C^1 \cap C^2, X_1]) = 0$, as $|C^1 \cap C^2| < j = T(P)$ (and similarly, $P(G^1_{X_2}) = 0$). Thus $N^{c,1}(P) \geq \log |F| \geq n/4$, if $j \geq n/4$. We also observe that our fooling set induces an identity submatrix so the rank bound could be used as well to get a bound on $D^c(P)$.

Otherwise, $j = T(P) < n/4$, and we look at the dual (non-trivial monotone) property $\tilde{P}$, which is true on G iff P is false on $\overline{G}$. Since on any set X of $n/2$ vertcies, $P(K[j,X]) = 1$ we have $\tilde{P}(\overline{K[j,X]}) = 0$ which by monotonicity implies that $\tilde{P}(K[n/2-j,X]) = 0$, so $T(\tilde{P}) > n/2 - j > n/4$, so we get $N^{c,0}(P) = N^{c,1}(\tilde{P}) \geq n/4$. Hence, $N^c(P) \geq \max(N^{c,0}(P), N^{c,1}(P)) = \Omega(n)$. ■

The above result can also be established for a non-monotone property PARITY using an idea quite similar to the one used in the known result for the inner product function [6]. Let M be the answer matrix for PARITY, i.e. $M(G,X) = P(G_X)$. Let $M' = M^T M$. We have that $rank(M) \geq rank(M')$. Let N be the total number of n-vertex graphs, i.e. $N = 2^{n(n-1)/2}$. $M'(X_1, X_2) = \sum_G P(G_{X_1}) P(G_{X_2}) = |\{G | P(G_{X_1}) = P(G_{X_2}) = 1\}|$. Assume both X_1 and X_2 are non-empty. If $X_1 = X_2 = X$ then exactly one half of the graphs will have parity 1 on X, so $M'(X,X) = N/2$. If $X_1 \neq X_2$ then we can assume w.l.o.g. that there is an edge $e \in Edges_{X_1} \backslash Edges_{X_2}$. $N/2$ graphs with parity 1 on X_2 can be partitioned into $N/4$ classes of size 2 where the two graphs in a class will differ in exactly the edge e. Exactly one of the graphs in each equivalence class will have parity 1 on X_1, so $M'(X_1, X_2) = N/4$. Since any $K \times K$ matrix with diagonal entries equal to a and off-diagonal entries equal to $b \neq a$ has full rank over the reals, we get $rank(M) \geq rank(M') \geq 2^n - 1$, so $D^c(\text{PARITY}) \geq n - 1$ and hence $\mathcal{T}_s(\text{PARITY}) = \Omega(\frac{n}{\log s})$.

3.2 Asymmetric Complexity

We examine next the *asymmetric* communication complexity of the induced subgraph problem. Recall that an $[A, B]$-protocol is a protocol where Alice sends at most A bits and Bob sends at most B bits. An asymmetric complexity lower bound $[A, B]$ means that either Alice has to send $\Omega(A)$ bits, or Bob has to send $\Omega(B)$ bits in order to compute the function. Assume we have a lower bound $[\frac{n^2}{r(n)}, n]$ $(r(n) \leq n)$ for some property P. We saw that a cell probe scheme with query time t, space s and cell size b produces a $[tb, t\log s]$-protocol. Thus, applying this to P we get $t \geq \Omega(\min(\frac{n^2}{r(n)b}, \frac{n}{\log s}))$. This minimum is $n/\log s$ provided $b \leq \frac{n \log s}{r(n)}$. Thus, even though we got the same bound on the number of *probes* needed, we can extend this lower bound to schemes with cell size up to $(n \log s)/r(n)$. We note that the actual number of *bits* read is $\Omega(n^2/r(n))$, which is much closer to n^2 especially if $r(n)$ is small. Even $r(n) = n$ (which we have by Theorem 3) lets us make $b = \log s \geq \log n$, but we would like to get $r(n) = 1$.

Conjecture 1 *For any evasive property P we have $[n^2, n]$ lower bound on asymmetric communication complexity for the induced subgraph problem.*

As the first step, we prove a slightly weaker $[\frac{n^2}{\log n}, n]$ lower bound for PARITY, which already implies that $t = \Omega(\frac{n}{\log s})$ even for $b = n$. We need the following lemma.

Lemma 3 *Let M be a vector space of all the n-vertex graphs under the $\oplus$ operation. Then any collection of k non-empty cliques has $\Omega(\log^2 k/\operatorname{loglog} k)$ linearly independent cliques.*

Proof: Let K_X denote a clique on X and let $\mathcal{C}$ be our collection of k cliques. We let $T(k)$ the *worst* possible rank of k non-empty cliques. We claim that for *any* $0 < \alpha < 1$

$$T(k) \geq \min(\log((1-\alpha)k) + T(\alpha k), \frac{1}{2\alpha}) \tag{3.2}$$

Assuming (3.2), we pick $\alpha = \frac{1}{\log^2 k}$ (we observe, it means different α at each level of recursion). It is easy to prove then by induction that $T(k) = \Omega(\log^2 k / \operatorname{loglog} k)$. To show (3.2), take any α. Assume first that there is a vertex $v \in V$ such that v belongs to $r \leq (1-\alpha)k$ cliques $K_{X_1}, \ldots, K_{X_r}$ in $\mathcal{C}$. Let H_i be a star from v to $X_i \backslash \{v\}$. All H_i are distinct since all X_i are. Any collection of r vectors has rank at least $\log r$. Say $H_1, \ldots, H_{\log r}$ are linearly independent. It is easy to see then that any collection $\mathcal{C}'$ of linearly independent cliques, each of which does not contain v, can be augmented by $K_{X_1}, \ldots, K_{X_{\log r}}$ to produce a still linearly independent collection. This is because each K_{X_i} contains H_i as a subgraph and no graph in $\mathcal{C}'$ contains v. Thus, a linear combination being 0 implies that no K_{X_i} are used, but this implies that no graphs in $\mathcal{C}'$ can be used as well. Thus, by inductive assumption, the number of linearly independent graphs would be at least $\log r + T(k-r)$, which is easily seen to be dominated by $\log((1-\alpha)k) + T(\alpha k)$, as $r \leq (1-\alpha)k$.

Otherwise, *every* $v \in V$ participates in more than $(1-\alpha)k$ cliques in $\mathcal{C}$. Thus every edge $e = (v, w)$ participated in more than $(1-2\alpha)k$ cliques in $\mathcal{C}$. Take any edge e_0 and let $\mathcal{C}_1$ to consist of all the cliques in $\mathcal{C}$ containing e_0. We then repeat the following process. Pick any clique $K_i \in \mathcal{C}_i$, which is not a complete graph (it exists if $|\mathcal{C}_i| > 1$) and any edge $e_i \notin K_i$. Let $\mathcal{C}_{i+1}$ contain all the cliques in $\mathcal{C}_i$ that contain e_i. Each time we eliminate at most $2\alpha k$ cliques by our assumption, so we can repeat at least $1/(2\alpha)$ times. We claim that K_i produced are linearly independent. For assume not and there are some $i_1, \ldots, i_j$ s.t. $\bigoplus K_{i_j} = 0$. Since *all* the cliques K_i contain e_0, we must have that j is *even* in order for e_0 to cancel. But then, $e_{i_1} \notin K_{i_1}$, while $e_{i_1} \in K_{i_2}, \ldots, K_{i_j}$. Since e_{i_1} has to cancel as well, we get that j has to be *odd*, a contradiction. Thus we obtain $1/(2\alpha)$ linearly independent cliques, and (3.2) follows.

■

Theorem 4 PARITY *has* $[\frac{n^2}{\log n}, n]$ *lower bound on asymmetric communication complexity.*

Proof: We use the richness technique described earlier. Let M be the answer matrix for PARITY, where the rows are the graphs and the columns are the subsets. First we show that PARITY is $(2^{\binom{n}{2}} - 1, 2^{n-2})$-rich. Take any non-empty graph $G = (V, E)$ and let $e = (a, b) \in E$. We claim that at least $1/4$ of the induced subgraphs have parity 1. Partition all the subsets of V into equivalence classes of size 4. Given any $X' \subseteq V \setminus \{a, b\}$ the equivalence class of X' consists of X', $X' \cup \{a\}$, $X' \cup \{b\}$, $X' \cup \{a, b\}$. We claim that at least one one the four induced subgraphs in each equivalence class has parity 1. Indeed, the $\oplus$ of the four parities counts every edge inside of $G_{X'}$ four times, every edge from a (b) to X' – two times, and edge e – one time, so the sum of four parities is 1, as $e \in E$. Thus, at least $1/4$ of the induced subgraphs have parity 1, if G is non-empty.

Now we have to bound the size of the largest 1-monochromatic rectangle in M. Assume it has size $2^a \times 2^{n/2}$. The $2^{n/2}$ subsets with parity 1 for their induced subgraphs define $2^{n/2}$ linear equations. By Lemma 3, there are at least $\gamma n^2/\log n$ *linear independent* equations in the system (for some $\gamma > 0$), so the number of solutions (graphs) can be at most $2^{\binom{n}{2} - \gamma n^2/\log n}$, so $a \leq \binom{n}{2} - \gamma n^2/\log n$. Thus PARITY does not have a $[\binom{n}{2} - (\binom{n}{2} - \gamma n^2/\log n), n - 2 - n/2] = [\gamma n^2/\log n, n/2 - 2]$ protocol. ■

We remark that improving the bound in Lemma 3 to $\Omega(\log^2 k)$ (easily seen to be the best possible) would give us $[n^2, n]$ bound for PARITY.

Chapter 4

Restricted Function Families with Restricted Space

We have thus far examined the inherent difficulties in obtaining unconditional super-linear lower bounds. In this section, we study the problem under some additional restrictions that would allow us to show much stronger lower bounds. Specifically, we seek to limit the power of functions that we are allowed to store in our data structure by talking about $\mathcal{F}$-restricted data structures and studying the $\mathcal{T}_{\mathcal{F},s}(P)$ measure. At one extreme, when we are allowed to store just the edges of the graph, it is the set-up of the AKR conjecture. For this setting, we immediately obtain an $\Omega(n^2)$ adaptive lower bound to compute any non-trivial monotone graph property even on a fixed query set, namely, the set V. Intuitively speaking, in this case the functions can not efficiently express the property even on a single query. At the other extreme, when the function family $\mathcal{F}$ is completely unrestricted, we obtain again the measure $\mathcal{T}_s(\_)$. Our goal here is to study $\mathcal{T}_{\mathcal{F},s}(\_)$ when $\mathcal{F}$ is some natural subclass of boolean functions whose expressive power lies somewhere in between these two extremes. As an elementary example, consider the OR family of functions. Any fixed query concerning (evasive) graph properties such as connectivity and bipartiteness, can be answered in $O(n \log n)$ probes using the OR family [3]. But can this family be used to significantly speed-up the induced subgraph problem for these and other evasive properties? The answer, as we show for many such function families, is "no".

We will show that irrespective of the efficiency in answering any single query, this and many other function families, can not yield better than a poly-logarithmic speed-up.

4.1 Stabilization Technique

The central technique used is what we call the *stabilization technique.* In order to explain the technique, we need two definitions.

Definition 6 (Gadget Graph) *An $\langle n, q(n)\rangle$-*gadget graph *$H(V, E)$ is a labeled clique on n vertices such that: (a) each edge is labeled* 0 *(missing),* 1 *(present), or $*$ (unspecified), and (b) there exists a subset $Q \subset V$ with $|Q| = q(n)$, such that Q induces a clique with each edge labeled $*$. We refer to Q as the* query set *of H.*

Definition 7 (Stabilizing Graph) *Given an $\mathcal{F}$-restricted data structure D of size s, a graph H is called an $\langle n, q(n), g(s)\rangle$-*stabilizing graph *for D if: (a) H is a $\langle n, q(n)\rangle$-gadget graph, and (b) every cell in D reduces to being a function of at most $g(s)$ edge variables on the partial assignment specified by H.*

Now suppose for a function family $\mathcal{F}$ we want to show $\mathcal{T}_{\mathcal{F},s}(P) = \Omega(q^2(n)/g(s))$ for every evasive property P. We start by showing existence of a $\langle n, q(n), g(s)\rangle$-stabilizing graph G_D for every $\mathcal{F}$-restricted data structure D. Thus when G_D is presented as the static input, every cell in D reduces to be a function of at most $g(s)$ edge variables. At the same time, we have access to a query set Q whose every edge is unspecified as yet. We present this set Q as the dynamic input to the scheme and play the evasive game for property P on the subgraph induced by Q. Since each cell probe can reveal at most $g(s)$ edge variables, we obtain the desired $\Omega(q^2(n)/g(s))$ lower bound. The following theorem summarizes this argument.

Theorem 5 *If every $\mathcal{F}$-restricted data structure of size s has a $\langle n, q(n), g(s)\rangle$-stabilizing graph, then for any evasive property P, we have $\mathcal{T}_{\mathcal{F},s}(P) = \Omega(q^2(n)/g(s))$.*

Thus the heart of our approach is showing the existence of a $\langle n, q(n), g(s)\rangle$-stabilizing graph with suitable parameters. In the remainder of this section, we show existence of such stabilizing graphs for many function families. The families that we study include AND, OR, α-CNF, α-DNF and symmetric monotone functions. For each of these families, we show the existence of stabilizing graphs with typically $q(n) = \Omega(n)$ and $g(s) = polylog(s)$. In other words, we show near-quadratic lower bounds for each of these families. We observe here that these parameters are essentially the best possible in general since many evasive graph properties can actually be sped-up by a *polylog*(s) factor using these function families. We also study the following question: What is the time complexity of induced subgraph problem for a property P when the data structure is restricted to contain only answers about which induced subgraphs of the input graph have property P. The interest in this question arises from the observation that indeed many properties can be efficiently computed on an entire graph by simply evaluating the property on various subgraphs. While any single query can now be answered in one probe, we show an $\Omega(n^2/\log^2 s)$ lower bound for the induced subgraph variant of the problem, for any non-trivial "uniform" monotone property. This bound is in fact tight for several evasive properties.

The common theme in obtaining the lower bounds for various families $\mathcal{F}$ is the use of the probabilistic method to show existence of stabilizing graphs. We start by establishing a simple lemma and its corollary that we invoke frequently in our analysis. The lemma shows that for any "large enough" set of edges, a random balanced partition of the vertex set yields a balanced partition of the set of edges. We need the following notation. Given a set $S \subset V$, we denote by $E_{S \times V}$ the set of edge variables with at least one end-point in S. Also, let $E(c_i)$ denote the set of edge variables that appear in the function stored in cell c_i of a given data structure D.

Lemma 4 *Let S be a random subset of V constructed by choosing every vertex in V with probability 1/2. Then there are constants η_1, η_2 such that for any set of edge variables Z such that $|Z| \geq \eta_1 \log^2 s$, we have* $\Pr[|Z \cap E_{S \times V}| \geq \eta_2 |Z|] \geq 1 - \log s/s^2$.

Proof: Let H denote the subgraph defined by the edges in Z and let $z = |Z|$. We choose $\eta_1 = 2^{12}$ and $\eta_2 = 1/16$. We denote by A_i the set of vertices in H with degree at least 2^i and at most $2^{i+1} - 1$. Define $J_1(J_2)$ to be the set of indices i such that

$|A_i| \geq \sqrt{z}/2$ $(2^i \geq \sqrt{z}/2)$. An edge (u, v) in H is called *bad* if $u \in A_{i_1}$ and $v \in A_{i_2}$ such that neither i_1 nor i_2 occurs either in J_1 or J_2. The total number of bad edges is upper bounded by the summation below:

$$\frac{1}{2}\left(\sum_{i \notin J_1, i \notin J_2} |A_i| 2^{i+1}\right) \leq \frac{1}{2}\left(\frac{\sqrt{z}}{2} \sum_{i=0}^{\log(\sqrt{z}/2)} 2^{i+1}\right) \leq \frac{z}{2}.$$

We will show that w.h.p. S gets a constant fraction of non-bad edges i.e. the edges in $Z \setminus Z_0$. Specifically, we show that (a) for every i such that $|A_i| \geq \sqrt{z}/2$, w.h.p. at least 1/4-fraction of these vertices belongs to S, and (b) for every i such that $2^i \geq \sqrt{z}/2$, w.h.p. at least 1/4-fraction of this neighborhood belongs to S.

To show (a) and (b), we use Chernoff bounds. Observe that since $\sqrt{z}/2 \geq 32 \log s$, using union bounds, we conclude that the probability that for *some* $i \in J_1 \cup J_2$, we fail to get the desired 1/4-fraction is bounded by $\log n(1/s^2) < \log s/s^2$. Thus, the number of edges guaranteed to be in S with probability at least $1 - \log s/s^2$ is at least

$$\frac{1}{2}\left(\frac{\sum_{i \in J_1 \cup J_2} |A_i| 2^i}{4}\right) \geq \frac{1}{2}\left(\frac{|Z \setminus Z_0|}{4}\right) \geq \frac{1}{8}\left(\frac{z}{2}\right) = \frac{z}{16}$$

■

The following is an immediate corollary; the existence of the desired S is shown by simply using the union bounds over all cells and observing that $|S| \leq n/2$ with probability at least 1/2.

Corollary 5 *For any data structure D, there exists a subset S of at most size $n/2$ such that for every c_i with $|E(c_i)| \geq \eta_1 \log^2 s$, we have $|E(c_i) \cap E_{S \times V}| \geq \eta_2 |E(c_i)|$ for some constants η_1 and η_2.*

Finally, we define the notion of a partial assignment that fixes the value of a function.

Definition 8 (Stabilizing Assignment) *A partial assignment to the variables of a boolean function is called* stabilizing *if it determines the value of the function to be true or false, irrespective of the value that is assigned to the unspecified variables.*

4.2 AND and OR Families

We start our study with AND and OR function families which in some sense constitute the most simple and natural extension of the AKR conjecture model.

Lemma 6 *Let $\mathcal{F}$ be the family of* AND *and* OR *functions. Then every $\mathcal{F}$-restricted data structure D has a $\langle n, n/2, O(\log^2 s)\rangle$-stabilizing graph.*

Proof: By Corollary 5, we know that there exists a set S of size at most $n/2$ such that $|E(c_i) \cap E_{S\times V}| \geq \log^2 s$ for every c_i with at least $\eta \log^2 s$ edges, for some constant η. Set each edge variable in $E_{S\times V}$ to be 0/1 uniformly at random. There is exactly one assignment that fails to stabilize any AND or OR function. Using the union bound, the probability that some c_i with $|E(c_i)| \geq \eta \log^2 s$ is not stabilized by the random assignment is $o(1)$. The result follows. ■

Combining Theorem 5 with Lemma 6, we obtain the following theorem.

Theorem 6 *For any evasive graph property P, $\mathcal{T}_{\{\text{AND},\text{OR}\},s}(P) = \Omega(n^2/\log^2 s)$.*

This result is the strongest possible general result; Chapter 6 shows that for many evasive properties, $\mathcal{T}_{\{\text{AND},\text{OR}\},s}(P) = O(n^2/\log^2 s)$ indeed.

4.3 α-CNF and α-DNF Families

We now extend the results of the previous section to a much richer class of functions, namely, the α-CNF and the α-DNF formulas for any constant α. The AND/OR families studied above correspond to the special case of $\alpha = 1$. Our approach relies on the following structural dichotomy of α-DNF formulas, which may be of independent interest.

Lemma 7 *Let f be an α-*DNF *formula on N variables and let $0 < r < 1$ be a positive real. Then*

- *either f has a decomposition of the form $l_0 f_0 + l_1 f_1 + \ldots + l_{p-1} f_{p-1}$ where l_i's are literals, f_i's are $(\alpha - 1)$-*DNF *formulas, and $p \leq ln(\alpha 2^\alpha N^\alpha)/r$, or*

- *f has at least $q = (1/2\alpha r)$ pairwise disjoint (i.e. no common variables) terms.*

Proof: Let k denote the number of terms in f; clearly k is at most $\alpha 2^\alpha N^\alpha$. A literal l in f is called *frequent* if it occurs in at least an r-fraction of the terms in the formula. Consider the sequence $\{f_i\}_{i \geq 0}$ of formulas derived from f in the following iterative manner: $f_0 = f$, and f_{i+1} is derived from f_i by finding a frequent literal l_i in f_i and deleting the set of all terms, say τ_i, in which l_i occurs. The process terminates when there are no more frequent variables; let f_p denote the final formula in this sequence. Since at each iteration we loose at least r-fraction of the terms, it is easy to see that $p \leq ln(k)/r \leq ln(\alpha 2^\alpha N^\alpha)/r$.

If f_p is an empty formula, then we are done. Otherwise, suppose that f_p contains q terms. We build a collection C of pairwise disjoint terms as follows. Pick any term Γ in f_p; each of its literals appears in at most rq terms. Thus there are at most $(2\alpha rq - 1)$ terms that share any variables with it. Add Γ to C, delete all terms that share a variable with it, and reiterate. Clearly, we can construct at least $1/2\alpha r$ pairwise disjoint terms in this manner. This completes the proof of the lemma. ■

Intuitively speaking, the lemma above says that either f has a certain "compact" decomposition or it has a "large" number of pairwise disjoint terms. The next lemma shows that it is easy to stabilize α-DNF formulas with large number of pairwise disjoint terms.

Lemma 8 *Let f be an α-DNF formula with $4\alpha^2 2^{4\alpha} \log^2 s$ terms such that each variable of f occurs in exactly one term in f. Let S be a subset of vertices constructed by choosing each vertex in V with probability $1/2$. Then assigning the value 0/1 uniformly at random to each edge variable in $S \times V$ stabilizes f with probability at least $1 - 1/s^2$.*

Proof: A set of vertices X is called the *hitting set* of a term t in f if every edge variable of t occurs in the set $X \times V$ of edge variables and that X is a minimal such set. Observe that a term can have many hitting sets, a fact critical to our analysis. Also observe that any hitting set contains at most α vertices since we are working with α-DNF formulas. We claim that f contains at least $k = 2^{2\alpha+1} \log s$ terms, say

$t_1, t_2, ..., t_k$ such that one can assign a hitting set X_i to each t_i with the property that $X_i \cap X_j = \emptyset$ for $i \neq j$. We build such a collection iteratively. Suppose we have picked i such terms so far and let Z denote the union of their hitting sets. The total number of vertices in the hitting sets corresponding to these terms can be at most αi. A term t in f is ruled out from being added to this collection if and only if it contains an edge variable whose both end points are contained in Z. Since X defines at most $(\alpha^2 i^2)/2$ such edge variables (and each variable occurs in 1 term), at most so many terms can be ruled out. Thus the process can not terminate before k iterations where $k + (k^2\alpha^2)/2 \geq 4\alpha^2 2^{4\alpha} \log^2 s$; implying that k must at least $2^{2\alpha+1} \log s$. Now f' is satisfied if $X_i \subset S$ for some $1 \leq i \leq k$ and the (at most) α edge variables corresponding to its term are assigned the unique satisfying assignment; let Φ_i denote this event. Observe that by our construction, $\Phi_1, ..., \Phi_k$ are all independent events and that $\Pr[\Phi_i] \geq (1/2^\alpha)^2$ since each of the two events comprising Φ_i has a probability of success at least $1/2^\alpha$. Thus $\Pr[\wedge_i \neg\Phi_i] \leq 1/s^2$, giving the desired result. ■

Theorem 7 *For any evasive property P and constant α, $\mathcal{T}_{\{\alpha-\mathsf{CNF},\alpha-\mathsf{DNF}\},s}(P) = \Omega(\frac{n^2}{\log^{\alpha-1} n \log^{2\alpha} s})$.*

Proof: First observe that it suffices to show the claimed lower bound for α-DNF formulas only, since any α-CNF formula can be implicitly stored as an α-DNF formula, namely, by storing its complement. By Theorem 5, it is enough to show the existence of a $\langle n, n/2, O(\log^{\alpha-1} n \log^{2\alpha} s)\rangle$-stabilizing graph for every α-DNF - restricted data structure D. Towards this end, we will show that if S is a random subset, constructed by choosing each vertex of V with probability 1/2, then setting each edge variables in $S \times V$ to 0/1 uniformly at random either stabilizes any α-DNF formula or reduces it to be a function of $O(\log^{\alpha-1} n \log^{2\alpha} s)$ edge variables, with probability $1 - o(1)$.

Consider any α-DNF formula f in a given data structure scheme D. Define $r(\alpha) = 1/\alpha^3 2^{4\alpha+3} \log^2 s$. By Lemma 7, either f has $4\alpha^2 2^{4\alpha} \log^2 s$ pairwise disjoint terms or it has a representation of the form $l_0 f_0 + l_1 f_1 + ... + l_{p-1} f_{p-1}$ where $p \leq p_{\max} = ln(\alpha n^{2\alpha})/r(\alpha) = O(\log^{\alpha-1} n \log^{2\alpha} s)$. In case of the first scenario, we know by Lemma 8 that f will be stabilized "almost certainly". Otherwise, using the compact decomposition of f and induction on α, we argue that almost certainly f will have no more than $h(\alpha) = O(\log^{\alpha-1} n \log^{2\alpha} s)$ variables. The case $\alpha = 1$ follows from the preceding section. Assume inductively that any α'-DNF formula with $\alpha' \leq \alpha - 1$ almost certainly reduces to a function of at most $h(\alpha - 1)$ variables. Since each f_i in the decomposition of f is an $(\alpha - 1)$-DNF formula, by inductive assumption, it almost certainly contains no more than $h(\alpha - 1)$ distinct variables. Thus the following recurrence is satisfied:

$$h(\alpha) \leq p_{\max} h(\alpha - 1) + p_{\max} \leq (p_{\max})^{\alpha-1} h(1) + \sum_{i=1}^{\alpha-1} (p_{\max})^i.$$

Using $h(1) = O(\log^2 s)$, it is easy to verify that $h(\alpha) = O(\log^{\alpha-1} n \log^{2\alpha} s)$.

Probability Analysis: There are two distinct ways a failure can occur. One if the probability that S is larger than $n/2$, and other, if the α-DNF formula f is left with more than $h(\alpha)$ distinct variables. The probability of the first event is easily seen to be bounded by $1/2$. To analyze the second probability, denote by $P(\alpha)$ the probability that a given α-DNF formula does not reduce to a function of at most $h(\alpha)$ distinct variables. Then using Lemma 8, we have

$$P(\alpha) \leq p_{\max} P(\alpha - 1) + \frac{1}{s^2}$$

Scaling $h(1)$ by a suitably large constant, it is easy to see that $P(\alpha)$ can be bounded by $o(1/s)$ for any constant α. Since there are at most s α-DNF formulas to be considered, the overall probability of failure is bounded by $s[o(1/s)] + 1/2 < 1$. ■

4.4 Symmetric Monotone Functions

The symmetric nature of AND and OR functions enabled us to obtain strong lower bounds relatively easily. In contrast, the results of the preceding section illustrate the difficulty in obtaining strong lower bounds when $\mathcal{F}$ contains "unstructured" functions.

In this section, we continue on to general threshold functions and show how symmetry makes it easy to obtain strong lower bounds. Recall, a threshold function $T_{p,q}$ ($F_{p,q}$) is true if and only if at least (less than) q of its p inputs are set to true. Since $\overline{F_{p,q}} = T_{p,q}$, we focus on $T_{p,q}$ only from now on. Our main result here is a $\Omega(n^2/\log^2 s)$ lower bound when $\mathcal{F}$ only contains threshold functions with $q \leq (1-\epsilon)p$. Notice that in order to express such functions as α-DNF or α-CNF formulas we need $\alpha = \Omega(p)$ which can be $\Omega(n^2)$ in our set-up. Yet, due to the symmetric nature of these functions, we can obtain results that are much stronger than ones obtainable for arbitrary α-CNF and α-DNF formulas.

Theorem 8 *Let $\mathcal{F}$ be the family of all threshold functions $T_{p,q}$ with "threshold ratio" bounded away from 1, i.e. $q \leq (1-\epsilon)p$. Then for any evasive property P, $\mathcal{T}_{\mathcal{F},s}(P) = \Omega(n^2/\log^2 s)$.*

Proof: It suffices to show that there exists a $\langle n, \Omega(n), O(\log^2 s)\rangle$-stabilizing graph G for every $\mathcal{F}$-restricted data structure. We construct a sequence of sets $S_1 \subset V, S_2 \subset V \backslash S_1, \ldots, S_\eta \subset V \backslash \cup_{i=1}^{\eta-1} S_i$ by repeatedly applying Corollary 5; let $S = \cup_{i=1}^{\eta} S_i$, $|S| \leq (1-2^{-\eta})n$. At each round every formula which still has more than $\eta_1 \log^2 s$ edges gets a constant fraction of its edges inside $S_i \times V$. Here $\eta = O(\log(1/\epsilon))$ is a sufficiently large constant to ensure that every threshold function with $\Omega(\log^2 s)$ edge variables has a $(1-\epsilon)$-fraction of its edges in $S \times V$. The stabilizing graph G is created by inserting all edges in $S \times V$; this clearly stabilizes every threshold function with $\Omega(\log^2 s)$ edges, leaving a query set of size at least $n/2^\eta = \Omega(n)$. ■

4.5 Computing P Using P

Definition 9 (Type-0 / Type-1) *A monotone graph property P defined over the space of n-vertex graphs is called* type-0 (type-1) *if every 0-certificate (1-certificate) touches at least $n/2$ vertices. P is* uniform *if P is type-0 (type-1) for every n.*

For instance, CONNECTIVITY is both type-0 and type-1, while CLIQUE is type-1 but not type-0. It is easy to see that for each n, a monotone property is either of type-0 or type-1, since otherwise we can put two opposite certificates on two halves of V. Uniformity naturally requires that the type does not change excluding artificial properties like "Does G have at least $(n/2) + (-1)^n(n/3)$ non-isolated vertices". Suppose we are given a data structure scheme D such that each cell $c_i \in D$ corresponds to evaluating P on some induced subgraph G' of the static input G. While any fixed query can now be answered in a single probe we show this does not help much for the induced subgraph problem. We have already addressed this question for some specific graph properties, such as CLIQUE and MAJORITY. Now, we study this question for the class of "uniform" monotone graph properties We will not only allow to store answers about uniform P of type-0 (type-1) on subgraphs G_X induced by some $X \subseteq V$, but will also allow to fix any collection of edges of G_X to *true* (false) and store the answer to this restricted property. We let $\mathcal{P}_0$ ($\mathcal{P}_1$) be the class of all uniform properties of type-0 (type-1).

Theorem 9 *Let P be a type-0 (type-1) non-trivial uniform property and let $\mathcal{P}$ be the class $\mathcal{P}_0$ ($\mathcal{P}_1$). Then $\mathcal{T}_{\mathcal{P},s}(P) = \Omega(\frac{n^2}{\log^2 s})$.*

Proof: We show existence of a $\langle n, \Omega(n), \Omega(\log^2 s)\rangle$-stabilizing graph G as follows. Consider a set S constructed by picking at random vertices from V with probability 2/3 each. Using Chernoff bounds, it is easy to see that "almost certainly" any particular subgraph induced by $\Omega(\log s)$ vertices has half of its vertices in S. Now, if P (and hence $\mathcal{P}$) is type-0 then set to *true* all edge variables in $S \times V$ and set them to *false* otherwise. This stabilizes the answer on every $\Omega(\log s)$ size induced subgraph stored in the data structure — a "yes" if $\mathcal{P}$ is type-0 and a "no" otherwise, so all "surviving" function depend on $O(\log^2 s)$ edges. The results now follows by presenting $Q = V \setminus S$ as the query set and playing an evasive game on this query. ■

4.6 Small Degree Polynomials

The stabilization technique has its limitations as illustrated by the family of $\bigoplus$ functions. Given any partial assignment specifying k out of l variables of an $\bigoplus$ function, we get back a function that is sensitive to each one of the remaining $l - k$ variables. The implication of this fact is that stabilizing graphs, cannot reduce every "large" $\bigoplus$ function to an $\bigoplus$ function on "small" number of variables. On the other hand, functions like $\bigoplus$ have a very nice structural property, namely, they can be expressed as a small degree polynomial over $\mathbb{Z}_2$. We develop an algebraic approach to study such function families Section 5.1.

Chapter 5

Restricted Function Families with Unrestricted Space

We now turn our attention to the other extreme when the space s is unrestricted and only the function family $\mathcal{F}$ is restricted, that is, to the measure $\mathcal{T}_{\mathcal{F}}(\_)$. Since the space is not an issue, $\mathcal{T}_{\mathcal{F}}(f)$ measures the decision tree complexity of computing a function f w.r.t. $\mathcal{F}$. In case of induced subgraph problem, $\mathcal{T}_{\mathcal{F}}(P)$ is the decision tree complexity w.r.t. $\mathcal{F}$ of computing P on the entire vertex set V. When $\mathcal{F}$ contains merely the identity function, it is the setting of the AKR conjecture and $\Omega(n^2)$ lower bound is known for any non-trivial monotone P [13, 5]. In this section we examine what happens when we allow more powerful functions in $\mathcal{F}$ such as AND, OR (more generally, threshold functions), and XOR (more generally, low degree polynomials). Observe that since the space is not an issue, even these seemingly simple function families efficiently capture many evasive properties. For example, $\mathcal{T}_{\text{AND}}(\text{ALL-NEIGHBORS}) = n$, $\mathcal{T}_{\text{OR}}(\text{CONNECTIVITY}) = \Theta(n \log n)$ [3], $\mathcal{T}_{\text{XOR}}(\text{PARITY}) = 1$. Yet, we will show that for a large class of evasive properties, these families are no more powerful than the trivial identity function.

We will also examine the effect of obliviousness, i.e. when the algorithm has to specify in all $\mathcal{T}^{obl}_{\mathcal{F}}(P)$ functions from $\mathcal{F}$ in advance. As a motivating example we show that if $\mathcal{F}$ is a family of all threshold functions, we get $\mathcal{T}_{\mathcal{F}}(\text{PARITY}) = \log \binom{n}{2}$, while $\mathcal{T}^{obl}_{\mathcal{F}}(\text{PARITY}) = \binom{n}{2}$ (even $\mathcal{T}_{\mathcal{F},s}(\text{PARITY}) = O(\frac{n^2}{\log^2 s} \operatorname{loglog} s)$, see Corollary 26), justifying the fact that obliviousness is a severe limitation. Later in Chapter 7 we will study in detail $\mathcal{T}^{obl}_{\text{AND}}$ and $\mathcal{T}^{obl}_{\text{OR}}$ measures, where we will again get adaptive/oblivious separation for many properties.

5.1 Low Degree Polynomials

We start by examining the (non-monotone) family $\mathcal{F}_{\text{deg}\leq k}$ of all multivariate polynomials over $\mathbb{Z}_2$ of degree at most k; the special case $k = 1$ gives the XOR family. We obtain the strongest possible general bound by showing that for any monotone property P, we have $\mathcal{T}_{\mathcal{F}_{\text{deg}\leq k}}(P) = \Omega(\frac{n^2}{k})$, e.g. $\mathcal{T}_{\text{XOR}}(P) = \Omega(n^2)$. Thus, small degree polynomials do not bring any significant computational advantage. In particular, having access to 2^m possible XORs on the edges of a graph, do not bring any advantage over the setting when we have access only to the edges of the graph. Our approach for establishing this bound is to study the *degree* of a multi-linear polynomial q computing a boolean function f over some ring R with identity, i.e. $f(x) = q(x)$, $\forall x \in \{0,1\}^m$. Denote by $\deg_R(f)$ the degree of this q. When $R = \mathbb{Z}_r$, we simply denote it by $\deg_r(f)$, where $2 \leq r \leq \infty$ (here $\mathbb{Z} = \mathbb{Z}_\infty$). The following lemma justifies this definition and states some elementary facts that we will use.

Lemma 9

- *For any f there is a* unique *multi-linear polynomial over R computing it, and its degree is at most $D(f)$.*
- *Let the characteristic $char(R)$ of R be the smallest $i \in \mathbb{Z}$, $i > 1$ s.t. $i \times 1 = 0$ (if no such i exists, $char(R) = \infty$). Then $\deg_R(f) = \deg_{char(R)}(f)$, so the only interesting rings to consider are $\mathbb{Z}_r$ for $2 \leq r \leq \infty$.*
- *For any $2 \leq r, s \leq \infty$, $\deg_r(f) \leq \deg_{rs}(f)$.*

Proof: Given $A \subseteq [m]$ we denote by M_A a monomial $\prod_{i\in A} x_i$ and by 1_A, a 0/1 characteristic vector of A. Assume q_1 and q_2 are 2 distinct multi-linear polynomials computing f. Then $q = q_1 - q_2$ computes the zero function and has some monomial with a non-zero coefficient. Let M_A be any such monomial of *smallest degree* with coefficient $c \neq 0$. Then $q(1_A) = c \neq 0$, because 1_A sets all other monomials to 0, a contradiction.

To show the existence, we take any decision tree T for f (for example, a complete binary tree of depth m). Let $x_i(a) = x_i$ if $a = 1$ and $1 - x_i$ if $a = 0$. For every leaf l with the value of f equal to b_l and values $x_{i_j} = a_j$ $(1 \leq j \leq depth(l))$ along the path to l, we let $q_l(x) = b_l \cdot \prod x_{i_j}(a_j)$ and $q = \sum_l q_l$. Then q is a multi-linear polynomial that agrees everywhere with f, since given any x, the only contributing q_l corresponds to the leaf l arrived when traversing T on x, and $q_l = f(x)$. Since we started from an arbitrary T, we immediately get $\deg_R(f) \leq D(f)$.

Let $R' = \{i \times 1 | i \in \mathbb{Z}\}$. Then R' is a subring of R. Moreover, $R' \approx \mathbb{Z}_{char(R)}$. From the explicit construction of the polynomial from any decision tree for f, we see that all the coefficients of q are in fact in R', so $\deg_R(f) = \deg_{R'}(f) = \deg_{char(R)}(f)$.

If $q = \sum_{A \subseteq [m]} C_A M_A \in \mathbb{Z}_{rs}[x]$ computes f, then it is easy to see that $q' = \sum_{A \subseteq [m]} (C_A \bmod r) M_A \in \mathbb{Z}_r[x]$ also computes f, so $\deg_r(f) \leq \deg_{rs}(f)$. ■

Nisan and Szegedy [12] showed that $D(f) \leq 16 \deg_{\mathbb{Z}}^8(f)$ for any f. For monotone graph properties, we obtain a much stronger tight relation:

Theorem 10 *For any non-trivial monotone graph property P on n-vertex graphs, $\deg_2(P) = \Omega(n^2)$.*

The theorem is quite surprising, since *any* multi-linear polynomial over $\mathbb{Z}_2$ which is invariant under relabeling of vertices (including the ones of very small degree) computes some valid graph property. For example, (evasive) property PARITY is computed by a polynomial of degree 1. The theorem asserts that for a *monotone* property the degree has to be large even over $\mathbb{Z}_2$. It is worthwhile to note that the above theorem essentially gives a stronger version of the AKR conjecture. As a simple consequence, we get

Corollary 10 *For any non-trivial monotone graph property P, $\mathcal{T}_{\mathcal{F}_{\deg \leq k}}(P) = \Omega(\frac{n^2}{k})$. In particular, $\mathcal{T}_{\text{XOR}}(P) = \Omega(n^2)$.*

Proof: We only need to show that $\deg_2(P) \leq k\mathcal{T}_{\mathcal{F}_{\deg\leq k}}(P)$. Consider a decision tree T of depth d that computes f w.r.t. $\mathcal{F}_{\deg\leq k}$. We will use T to obtain a polynomial of degree at most kd that computes f. The construction is the same as in Lemma 9. For any polynomial p we let $p^0 = 1 - p$, $p^1 = p$, and let b_l be the value of leaf l. Now for each leaf l, construct a polynomial q_l as follows. Let $p_1, \ldots, p_d$ be the polynomials that were queried along the path from root to l and let a_i be the answer given by p_i. Define $q_l = b_l \prod p_i^{a_i}$, and let q be the sum of all such leaf polynomials. Clearly, q is a multinomial that computes f and has degree at most kd. ■

The bound is best possible in general, for example, $\mathcal{T}_{\mathcal{F}_{\deg\leq k}}(\text{CLIQUE}) = \Theta(\frac{n^2}{k})$. Thus, we only need to prove Theorem 10.

5.1.1 Transitive Functions and Their Degree

To establish Theorem 10, we follow the approach used by Rivest and Vuillemin [13] in resolving the AKR conjecture. Let S_m be the set of permutations on m elements. Given a monomial $e = cM_A$, $\sigma \in S_m$ and a subgroup $\Gamma \subseteq S_m$, we let $\sigma A = \{\sigma(i) | i \in A\}$, $\sigma e = cM_{\sigma A}$, $e\Gamma = \{\sigma e | \sigma \in \Gamma\}$, and $w(e) = |A| = \deg e$. $e\Gamma$ is called the *orbit* of e.

Definition 10 *For a function f on m inputs we let the* stabilizer subgroup $\Gamma(f) = \{\sigma \in S_m | \forall a \in \{0,1\}^m f(a) = f(\sigma a)\}$. *A group $\Gamma \subseteq S_m$ is* transitive *if $\forall i, j \in [m]$ $\exists \sigma \in \Gamma$ s.t. $\sigma(i) = j$. f is called* transitive *if $\Gamma(f)$ is transitive.*

Let p be a prime. Rivest and Vuillemin [13] proved that any transitive function f on $m = p^k$ variables s.t. $f(\vec{0}) \neq f(\vec{1})$ is evasive. We strengthen this result (Theorem 11) by following a similar, but more direct, argument which shows that in fact $\deg_p(f) = m$.

Lemma 11 *If q is a multi-linear polynomial over R computing f and e is a monomial in q, then q contains all the monomials in $e\Gamma(f)$.*

Proof: Let $e' = \sigma e$ for $\sigma \in \Gamma(f)$ and let $q'(x) = q(\sigma x)$. We have for any $a \in \{0,1\}^m$ that $q(a) = f(a) = f(\sigma a) = q(\sigma a) = q'(a)$. By the uniqueness of polynomial computing f (Lemma 9) we must have that $q(x) \equiv q'(x)$, and since q' has e', so does q. ■

Lemma 12 *If* Γ *is transitive, then for any monomial* e *we have*

$$w(e) \cdot |e\Gamma| = m \cdot |\{e' \in e\Gamma | e' \textit{ contains } x_1\}| \tag{5.1}$$

In particular, m *divides* $w(e) \cdot |e\Gamma|$.

Proof: Let M denote the $|e\Gamma|$ by m matrix whose rows are the characteristic vectors of monomials in $e\Gamma$. We count the number of ones in M by rows and by columns. The left side of (5.1) counts the ones by rows (each row has the same number of ones as $w(e) = w(\sigma e)$), the right — by columns, taking the number of ones in the first column. Transitivity of Γ ensures that each column contains the same number of ones (for any $i, j \in [m]$ there is $\sigma \in \Gamma$ s.t. $\sigma(i) = j$ which produces a bijection between all the monomials in $e\Gamma$ containing x_i and the ones containing x_j). ■

We are now ready to prove the main theorem.

Theorem 11 *If* f *is transitive,* $f(\vec{0}) \neq f(\vec{1})$ *and* $m = p^k$*, then* $\deg_p(f) = m$.

Proof: Let q be the (unique) multi-linear polynomial over $\mathbb{Z}_p$ computing f. Assume $\deg q < m$ and let e_0 be free term (zero degree monomial) of q. Then $f(\vec{0}) = e_0$. For any monomial e, we have by Lemma 12 that $m = p^k$ divides $w(e) \cdot |e\Gamma(f)|$, so unless $w(e) = 0$ or $w(e) = m$, we must have that p divides $|e\Gamma(f)|$. In particular, since by assumption q does not have a monomial of degree m, for any monomial e of q other than e_0 we must have $p \mid |e\Gamma(f)|$. We now apply Lemma 11 and divide the monomials of q into disjoint orbits of the form $e\Gamma(f)$. We then compute $f(\vec{1}) = q(\vec{1})$ on an orbit by orbit basis. If $e \neq e_0$ has coefficient $c(e) \in \mathbb{Z}_p$ then the orbit of e contributes $c(e) \cdot |e\Gamma(f)| \bmod p = 0 \bmod p$, as $p \mid |e\Gamma(f)|$. Hence we get that $f(\vec{1}) = q(\vec{1}) = e_0 = f(\vec{0})$, a contradiction. Thus, $\deg_p(f) = m$. ■

We observe that every condition in Theorem 11 is crucial. For example, it is easy to see that *MOD-p* function which is 0 iff the number of true variables is divisible by p has $\deg_p(f) = p - 1 << m$. But it does not satisfy $f(\vec{0}) \neq f(\vec{1})$ when $m = p^k$, and does satisfy it for any m not divisible by p. This example also shows that we cannot expect to bound $D(f)$ by some function of $\deg_p(f)$, since *MOD-p* is evasive (have to look at all the inputs given the all-zero input).

Rivest and Vuillemin [13] applied their analog of Theorem 11 to show the AKR conjecture by reducing it to evasiveness of certain non-trivial transitive function on 2^k edges of the graph (which was a certain restriction of the given graph property P). The identical reduction works in the case of degrees as well, as the degree of the restricted function can be at most the degree of the original one. This completes the proof of Theorem 10.

5.2 "Small Threshold" Family

The techniques of the preceding section do not apply to perhaps the most natural extensions of the setup of AKR conjecture, namely, how efficiently can graph properties be computed when we are allowed to store an AND (OR) of any subset of edges. The breakdown occurs because polynomial representations of AND and OR functions require up-to $\Omega(n^2)$ degree. In fact, many monotone properties have $\mathcal{T}_{\text{AND}}(P)$ much smaller than n^2, for instance, $\mathcal{T}_{\text{AND}}(\text{CLIQUE}) = 1$ and $\mathcal{T}_{\text{AND}}(\text{ALL-NEIGHBORS}) = n$. On the other hand, there are many other properties for which we have $\mathcal{T}_{\text{AND}}(P) = \Omega(n^2)$; examples include (as we will show) CONNECTIVITY and BIPARTITE. The goal of this subsection is to develop characterizations of properties for which the AND and OR function families do not bring additional computational advantage. Our approach is to reduce the question of lower bounding $\mathcal{T}_{\text{AND}}(\_)$ and $\mathcal{T}_{\text{OR}}(\_)$ to a certain measure of *simple* decision tree complexity. We then develop several general techniques to bound this measure from below. Since the technique we describe holds for any function f, we present our results here in this more general framework. Assume throughout that z ranges over $\{0, 1\}$.

Definition 11 *Given a decision tree T, let $d(T)$ denote the depth of T. Define the* z-depth *of T, $d_z(T)$ to be the maximum number of edges labeled by z on a root-leaf path. Given a function f, its* simple z-decision tree complexity *is the minimum of*

$d_z(T)$ over all decision trees T computing f. For a function family $\mathcal{F}$, $D(\mathcal{F})$ (resp. $D_z(f)$) is the maximum of $D(g)$ (resp. $D_z(g)$) over $g \in \mathcal{F}$.

The standard relation $\mathcal{T}_{\mathcal{F}}(f) \geq D(f)/D(\mathcal{F})$ can be extended as follows

Lemma 13 *For any function family $\mathcal{F}$, $\mathcal{T}_{\mathcal{F}}(f) \geq \frac{D_z(f)}{D_z(\mathcal{F})}$ where $z \in \{0,1\}$.*

Proof: Consider an optimal decision tree T w.r.t. $\mathcal{F}$ that computes f in depth $\mathcal{T}_{\mathcal{F}}(f)$. Replace every node v of T labeled by some $g \in \mathcal{F}$ by the optimal simple decision tree of z-depth $D_z(g) \leq D_z(\mathcal{F})$. The resulting simple decision tree computes f and has z-depth at most $\mathcal{T}_{\mathcal{F}}(f)D_z(\mathcal{F})$. Hence, $D_z(f) \leq \mathcal{T}_{\mathcal{F}}(f)D_z(\mathcal{F})$. ■

The measures D_0, D_1 will allow us to get strong lower bounds not only for AND and OR, but more generally, for threshold functions with a "small threshold".

Corollary 14 *Let $\mathcal{F}_{k,1} = \{T_{p,q} \mid q \leq k\}$, $\mathcal{F}_{k,0} = \{T_{p,q} \mid p - q < k\}$. Then $\mathcal{T}_{\mathcal{F}_{k,z}}(f) \geq \frac{D_z(f)}{k}$ where $z \in \{0,1\}$. In particular, the case $k = 1$ yields $\mathcal{T}_{\text{AND}}(f) \geq D_0(f)$, $\mathcal{T}_{\text{OR}}(f) \geq D_1(f)$.*

Proof: For any $T_{p,q} \in \mathcal{F}_{k,0}$, $D_0(T_{p,q}) = q - p + 1 \leq k$, so $D_0(\mathcal{F}_{k,0}) = k$. Similarly, $D_1(F_{k,1}) = k$. ■

The next lemma shows that for AND and OR function families, the converse is also almost true.

Lemma 15 *For any f, $\mathcal{T}_{\text{AND}}(f) \leq D_0(f)\log m$, $\mathcal{T}_{\text{OR}}(f) \leq D_1(f)\log m$. For* monotone *$f$, $\mathcal{T}_{\text{AND}}(f) \leq (D_0(f) - 1)\log m + 1$, $\mathcal{T}_{\text{OR}}(f) \leq (D_1(f) - 1)\log m + 1$.*

Proof: We focus on AND functions; an analogous argument applies to OR functions. Given a simple decision tree T_k for f of 0-depth k, we show how to build an AND decision tree T'_k for f of depth at most $k \log m$ ($(k-1)\log m + 1$ for monotone f). We first look at the all 1 path in T_k. We make a binary search tree using ANDs in T'_k of depth at most $\log m$ that would either tell us that all the variables along the path are true, or will give the *first* place where some x_i is false. In the first case, we just output the answer, in the second, we only have to deal with a simple decision tree of 0-depth $(k-1)$ (the left subtree of x_i), so by induction we get that $d(T'_k) \leq (k-1)\log m + d(T'_1)$. For general f, we can only bound $d(T'_1)$ by $\log m$ by again doing a binary search. For monotone f, the only simple decision tree of 0-depth 1 is the one computing some AND function, so a single AND will suffice. ■

Thus, to get optimal/near-optimal lower bounds on $\mathcal{T}_{\text{AND}}$ or $\mathcal{T}_{\text{OR}}$, it suffices to develop techniques to get good bounds on D_z. For this it is useful to look at an equivalent definition of $D_z(f)$ as a value of the simple decision game where the adversary A tries to maximize the number of z-answers that he is able to give (rather then the number of questions he is asked). So to show $D_z \geq b$ we only have to give a strategy for A allowing him to always give at least b answers equal to z. In the remainder of this section, we study three technqiues that will allow us to lower bound D_z in this manner. We will only concentrate on *monotone* f (and graph properties P) in our study.

5.2.1 "Dense Certificate" Technique

This is the most elementary technique — it uses the simple fact that $D_z(f)$ must be at least as large as any minimal z-certificate of f.

Lemma 16 *For any monotone function f,*

$$\mathcal{T}_{\text{AND}}(f) \geq D_0(f) \geq N^0(f) = \mathcal{NT}^0_{\text{AND}}(f) \quad \text{and} \quad \mathcal{T}_{\text{OR}}(f) \geq D_1(f) \geq N^1(f) = \mathcal{NT}^1_{\text{OR}}(f).$$

Proof: The answering strategy consistent with the largest min-z-certificate of f would force at least $N^z(f)$ z-answers before deciding $f = z$. Since our answering strategy is non-adaptive, it's gives a non-deterministic lower bound. ■

We can apply this to various monotone graph properties: $\mathcal{T}_{\text{AND}}(\text{CONNECTIVITY}) \geq N^0(P) = n^2/4$, $\mathcal{T}_{\text{OR}}(k\text{-CLIQUE}) \geq N^1(P) \geq \binom{k}{2} = \Omega(n^2)$ if $k = \Omega(n)$. For some properties, however, the bound is too weak. For example, $P =$NO-ISOLATED-VERTEX has $N^0(P) = n - 1$, even though we will see that $D_0(P) = \Omega(n^2)$. This is not surprising, since we already observed that the bound is non-deterministic.

5.2.2 Evasive Path Technique

The next answering strategy we examine applies to evasive f only. Since f is evasive, there is an evasive strategy A that would always force to ask all m inputs of f. Using A as our answering startegy, we get that $D_z(f)$ is greater than the least number of z answers produced by A over all $m!$ question orderings. We apply this to various monotone graph properties. An important point to note here is that the AKR conjecture is proven only for n being a prime power. This, however, is sufficient for the induced subgraph problem since we can take the largest subset $X \subseteq V$ with a prime power of elements and work on X.

For $P =$NON-BIPARTITE, every min-0-certificate has $\binom{a}{2}+\binom{n-a}{2} \geq 2\binom{n/2}{2}$ edges and thus $D_0(P) = \Omega(n^2)$. For $P =$NO-ISOLATED-VERTEX, it is readily seen that the graph G produced by A after $m-1$ questions has to be acyclic and hence $D_0(P) \geq \binom{n}{2} - n$. To show that G cannot have a cycle, we can look at the last edge e of this cycle queried. Querying of e is useless for NO-ISOLATED-VERTEX, as we already know that none of the vertices of the cycle is isolated. Thus an optimal querying strategy will never query e and decide P in at most $m - 1$ queries, contradicting the evasiveness of P under A.

Even though potentially powerful, the evasive path technique is not easy to use since it is hard to explicitly characterize evasive strategies. Thus, for instance, while we know every non-trivial monotone graph property is evasive when n is a prime power, for only a handful of properties we have an explict evasive answering strategy. In such cases, we have to find some structural property or invariant of the evasive strategy (like G is acyclic) that would let us conclude something without actually knowing the strategy. For many properties, however, this argument is hard to find (e.g. D_1(TRIANGLE), where TRIANGLE is true if G has a 3-clique).

5.2.3 "YES/NO unless cannot" Technique

We now examine in detail two simple yet quite powerful answering strategies that would often produce optimal bounds for a monotone function f. We find the exact measure of how well they perform (including when they are evasive strategies) and then translate the results into lower bounds for $D_0(f)$ and $D_1(f)$. The symmetric strategies in question which we call "YES/NO unless cannot" are, naturally, "YES unless cannot" and "NO unless cannot". As the name suggests, "NO unless cannot" would try to answer "no" unless it is forced to say "yes", i.e. answering "no" in conjunction with all the previous answers would force the monotone function to 0. Only in such cases we answer "yes". By definition, at the end of the game under "NO unless cannot" strategy, the answer is always going to be that f is 1. The picture is symmetric for the case of "YES unless cannot".

We begin with the notion of a hitting set.

Definition 12 (hitting set) *For a family $\mathcal{H}$ of subsets we call a set H a* hitting set *for $\mathcal{H}$ if H intersects ("hits") all members of $\mathcal{H}$.*

Let us view any 0-certificate (1-certificate) of a monotone f as a subset of variable set to 0 (1). In this view, any min-0-certificate for f is a minimal hitting set for $\mathcal{C}^1$ (all min-1-certificates for f) and vice versa. We next define a concept related to certificates that would be central to our characterization of the efficacy of "YES/NO unless cannot" strategies.

Definition 13 *For a monotone function f and a min-1-certificate C_1 of f we call a subset of variables L* leave-1-certificate for C_1 *if it is a hitting set for $\mathcal{C}^1 \backslash \{C_1\}$ and $L \cap C_1 = \emptyset$. L is* minimal leave-1-certificate for C_1 *if no proper subset of L is a leave-1-certificate. We let leave(C_1) be the (minimal) leave-1-certificate for C_1 of smallest size. We similarly define the concepts of* leave-0-certificate for C_0, minimal leave-0-certificate for C_0, *leave(C_0) for a min-0-certificate C_0.*

If we set the variables of some leave-1-certificate L for C_1 to *false*, we "eliminate" all the min-1-certificates except C_1. So we almost force the value of f to be 0 — the only way to force f to be 1 now is by setting all the variables in C_1 to be true. The subset *leave*(C_1) is a minimum collection of variables that would "kill" all min-1-certificates except C_1. We contrast a minimal leave-1-certificate L for C_1 with a min-0-certificate C_0: C_0 hits all the min-1-certificates while L hits all the min-1-certificates except for a given C_1. Naturally, adding any variable of C_1 to L is a 0-certificate, so $\min_{C_0} |C_0| \leq \min_{C_1} |leave(C_1)| + 1$. For example, for CONNECTIVITY we have $\min_{C_0 \in \mathcal{C}^0} |C_0| = n - 1$ (missing star), but for any spanning tree C_1, the only leave-1-certificate L for C_1 is all edges outside of C_1. Indeed, if L does not contain a single edge $e \notin C_1$, then we can add e to C_1, delete some other edge e' of the created cycle getting another spanning tree $C' \neq C$ that is not hit by L. Hence, $|leave(C_1)| = \binom{n}{2} - (n - 1)$.

Theorem 12 *For any monotone function f, if we use the answering strategy "NO unless cannot", then*

- *for any $C_1 \in \mathcal{C}^1$ there is a questioning (probing) strategy that asks $|leave(C_1)| + |C_1|$ questions and gets exactly $|leave(C_1)|$ "no" asnwers.*
- *for any questioning (probing) strategy there is $C_1 \in \mathcal{C}^1$ s.t. the number of questions is at least $|leave(C_1)| + |C_1|$, the number of "no" answers is at least $|leave(C_1)|$ and the number of "yes" answers is at least $|C_1|$.*

In particular, the smallest number of questions possible is $\min_{C_1}(|leave(C_1)| + |C_1|)$, *and the minimum number of* 0 *answers possible is* $\min_{C_1} |leave(C_1)|$. *A similar result holds for "YES unless cannot".*

Proof: Take any C_1. Ask about all the variables in $leave(C_1)$, getting all "no" answers. Then ask about the variables in C_1, getting all the "yes" answers as C_1 is the only min-1-certificate left. For the converse we start from an arbitrary question startegy. Let $x_1 \ldots x_j$ be the variables that were answered to be 1. Since once the variable is forced to 1 (i.e. setting it to 0 forces f to 0), it remains forced to 1, we can safely ask the questions about $x_1 \ldots x_j$ after we get all the 0 answers. Since at the end we decide that $f = 1$, it means that $x_1 \ldots x_j$ contain some min-1-certificate C_1 of f. So we can only reduce the number of questions and ask about the variables in this C_1 after we get all our 0 answers. Now, why would all the variables of C_1 be forced to 1? It must be the case that the 0 answers contain a minimal leave-1-certificate L

for C_1, since if at least one min-1-certificate $C_1' \neq C_1$ is not hit, any variable x_i with $i \in C_1 \backslash C_1'$ is not forced to 1, as C_1' is still a possibility. Moreover, we do not need to ask the "no" questions outside of L, since L already forces all the variables in C_1 to 1. ∎

The following is an immediate corollary.

Corollary 17 *For any monotone f, $\mathcal{T}_{\text{AND}}(f) \geq D_0(f) \geq L^0(f)$ and $\mathcal{T}_{\text{OR}}(f) \geq D_1(f) \geq L^1(f)$ where $L^0(f) = \min_{C_1 \in \mathcal{C}^1} |leave(C_1)|$ and $L^1(f) = \min_{C_0 \in \mathcal{C}^0} |leave(C_0)|$.*

Theorem 12 also allows us to characterize the class of functions for which "YES/NO unless cannot" gives an evasive strategy.

Corollary 18 *For any monotone function f*

- *"NO unless cannot" is an evasive strategy iff $\forall x$ s.t. $f(x) = 1$ and $\forall i$ s.t. $x_i = 0$ $\exists j$ s.t. $x_j = 1$ and making $x' = x$ except swapping $x_i = 1$, $x_j = 0$, still leaves $f(x') = 1$.*
- *"YES unless cannot" is an evasive strategy iff $\forall x$ s.t. $f(x) = 0$ and $\forall i$ s.t. $x_i = 1$ $\exists j$ s.t. $x_j = 0$ and making $x' = x$ except swapping $x_i = 0$, $x_j = 1$, still leaves $f(x') = 0$.*

Proof: We focus on "NO unless cannot"; an analogous argument holds for "YES unless cannot". Assume "NO unless cannot" is evasive for f. Clearly, it is sufficient to look only at x being some min-1-certificate for f. So let $x = x_{C_1}$ (meaning $x_k = 1$ iff $k \in C_1$) and let $x_i = 0$. By Theorem 12, the optimal question strategy takes time $\min_{C_1' \in \mathcal{C}^1}(|leave(C_1')| + |C_1'|)$. Hence, by evasiveness, for every min-1-certificate C_1' we have $|leave(C_1')| + |C_1'| = m$, including C_1. As $i \in \bar{C}_1 = L$, there is a min-0-certificate $C_1' \subseteq C_1 \cup \{i\}$ with some $j \in C_1 \backslash C_1'$. This j satisfies the claim, as min-1-certificate $C_1' \subseteq C_1 \cup \{i\} \backslash \{j\}$. The converse is similar. ∎

As an example, we see that "NO unless cannot" is evasive for CONNECTIVITY, since when adding an edge to a spanning tree we can always remove some other edge of the cycle and still have a spanning tree left. We might hope that for any non-trivial monotone property either "NO unless cannot" or "YES unless cannot" forces $\Omega(n^2)$ questions. However, it is not hard to see that for $\sqrt{n}$–CLIQUE both $\min_{C_1 \in \mathcal{C}^1}(|leave(C_1)| + |C_1|)$ and $\min_{C_0 \in \mathcal{C}^0}(|leave(C_0)| + |C_0|)$ are $O(n^{3/2})$.

As a non-trivial illustration of Corollary 17, we apply it to NO-ISOLATED-VERTEX. Take any C_1, that is a union of stars on subsets $X_1, \dots, X_k$ partitioning V. We claim that at most $\binom{k}{2}$ edges outside of C_1 do not have have to be in $leave(C_1)$. Thus, $|leave(C_1)| \geq \binom{n}{2} - (n-k) - \binom{k}{2} \geq 3n^2/8 - 3n/4 = \Omega(n^2)$, since $k \leq n/2$.

To show the claim, we let c_i be the vertex that is the center of the star on X_i (if $|X_i| = 2$ both vertices are the centers). We look at every edge $e = (a, b)$ (where $a \in X_i$, $b \in X_j$) outside of C_1 and argue which e have to be in $leave(C_1)$. The argument has a format that "if this $e \notin leave(C_1)$ then there is some other min-1-certificate $C_1' \subseteq C_1 \cup \{e\}$ which is a contradiction". This reduces to the case analysis. (see Figure 5-1).

- Case 1: $i = j$, $|X_i| > 3$ (then both $a \neq c_i$, $b \neq c_i$). We delete a, b form X_i and let them form a separate component using e. As we had $|X_i| > 3$, we still leave at least 2 vertices in X_i.

- Case 2: $i = j$, $|X_i| = 3$ (then both $a \neq c_i$, $b \neq c_i$). Leave X_i the same but make a (or b) a new center instead of the third vertex c_i, thus adding only e as a new edge.

- Case 3: $i \neq j$, $|X_i| > 2$, $|X_j| > 2$, $a \neq c_i$, $b \neq c_j$. We delete a and b from X_i and X_j, resp., and make a new component X_{k+1} on a and b with edge e. As we had $|X_i| > 2$, $|X_j| > 2$ we still leave at least 2 vertices in X_i and X_j.

- Case 4: $i \neq j$, $|X_i| > 2$, $a \neq c_i$, $b = c_j$ (when $|X_j| = 2$ the condition $b = c_j$ is satisfied). We simply switch a from X_i to X_j where we add e to connect $c_j = b$ to a. As we had $|X_i| > 2$, we still leave at least 2 vertices in X_i.

- Case 5: $i \neq j$, $|X_i| > 2$, $|X_j| > 2$, $a = c_i$, $b = c_j$. We don't know what to do with such edges.

- Case 6: $i \neq j$, $|X_j| = 2$, $a = c_i$ (includes case $|X_i| = 2$). Let $X_j = \{b, b'\}$. Again, $e = (a, b)$ does not have to be in $leave(C_1)$, but at least one of (a, b) and (a, b') does, as otherwise we can move both b and b' to X_i.
- Case 7: $i \neq j$, $|X_i| = |X_j| = 2$. Let $X_i = \{a, a'\}$, $X_j = \{b, b'\}$. Again, $e = (a, b)$ does not have to be in $leave(C_1)$, but at least one of (a, b), (a', b') does, as otherwise (a, b) and (a', b') can form two new stars. Together with Case 6, this shows that at most 1 out of 4 edges between X_i and X_j may be missing in $leave(C_1)$.

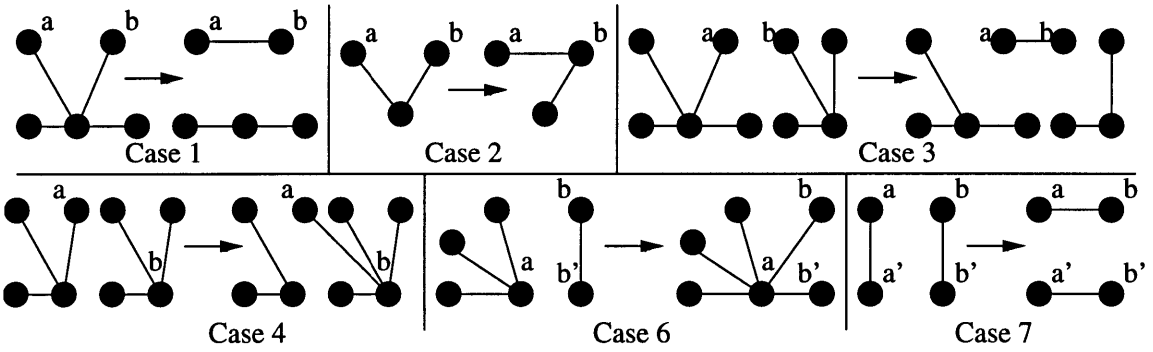

Figure 5-1: Various Cases for NO-ISOLATED-VERTEX

This shows that at most one edge between every pair of X_i and X_j $(i \neq j)$ can be missing from $leave(C_1)$, so at most $\binom{k}{2}$ edges overall.

Remark 1 *The "Dense Certificate" and "YES/NO unless cannot" techniques are in general orthogonal. We saw that for* P =NO-ISOLATED-VERTEX, $L^0(P) = \Omega(n^2)$, $N^0(P) = n - 1$. *For* $P = \sqrt{n}$-CLIQUE, *on the other hand,* $L^0(P) = O(n^{3/2})$, *while* $N^0(P) \geq \binom{n-\sqrt{n}}{2} = \Omega(n^2)$.

5.3 Oblivious versus Adaptive Bounds

We conclude by briefly examining the limiting effects of obliviousness. It is clear that for any f and $\mathcal{F}$, $\mathcal{T}^{obl}_{\mathcal{F}}(f) \leq 2^{\mathcal{T}_{\mathcal{F}}(f)}$, because we can take *all* the functions in an optimal decision tree for f w.r.t. $\mathcal{F}$ as our oblivious collection of functions. We now show that for certain scenarios this is the best that one can do, i.e. there is an exponential gap between adaptive and oblivious decision tree complexities.

Lemma 19 *Let $\mathcal{F}$ be the family of all the threshold functions. Let f be the parity function on m variables. Then $\mathcal{T}_{\mathcal{F}}(f) = \log m$ while $\mathcal{T}_{\mathcal{F}}^{obl}(f) = m$.*

Proof: For the adaptive bound, we only need to use threshold functions of the form $T_{m,\_}$. We simply perform a binary search and count exactly how many variables are set to true. Taking this number mod 2 yields the result in $\log m$ questions. The lower bound follows from the oblivious bound below and $\mathcal{T}_{\mathcal{F}}(f) \geq \log \mathcal{T}_{\mathcal{F}}^{obl}(f)$.

For the oblivious bound we proceed by an induction on m. Let $g_1, \ldots, g_k$ be the threshold functions used by the oblivious algorithm. We claim that at least one of these functions must have threshold 1. For otherwise, these functions would not let us distinguish between the all zero input (of parity 0) and any input with a single one (of parity 1). Assume, g_k is the function with threshold 1. Take any variable in g_k, say it is x_m, and fix its value to true. This fixes g_k to true and makes it redundant in obtaining any further information. The functions $g_1, \ldots, g_{k-1}$ now project down to threshold functions $h_1, \ldots, h_{k-1}$. Now to compute $x_1 \bigoplus \ldots \bigoplus x_{m-1} \bigoplus x_m = (x_1 \bigoplus \ldots \bigoplus x_{m-1}) \bigoplus 1$ we need to compute $x_1 \bigoplus \ldots \bigoplus x_{m-1}$ using $h_1, \ldots, h_{k-1}$. By the inductive assumption, $k - 1 \geq m - 1$, so $k \geq m$. ■

Chapter 6

Upper Bounds Techniques and Results

Until now we have focused on obtaining lower bounds for induced subgraph problem under a variety of different measures. We have seen that despite the inherent difficulties in showing super-linear lower bounds in general, it is possible to show near-quadratic lower bounds under certain restrictions — in line with our belief that polynomial preprocessing space does not yield significant speedups. We now approach the induced subgraph problem from an algorithmic viewpoint. For which properties can we improve upon the trivial $O(n^2)$ upper bound? As our lower bound results might already suggest, it is unlikely to achieve speedups better than a $polylog(s)$ factor. However, as we shall see, for many fundamental properties such as connectivity and bipartiteness, even achieving small speedups requires non-trivial new ideas. Our approach here is to develop canonical techniques for improving upon the trivial bounds. To begin with, we develop a representation scheme, called the *standard graph representation* which allows us to "efficiently" [1] perform useful operations on the graph induced by any query set. The basic idea of this representation is to partition the graph into small clusters and to exhaustively store some information within each cluster. To construct information about any given induced subgraph, we sim-

[1] In this section, words such as "efficient" and "speedup" signify only a modest $polylog(s)$ factor improvements.

ply combine together relevant pieces from within each cluster. A clustering based representation scheme has also been used by Hellerstein *et al* [4] for a static data structure problem on bipartite graphs. We show that several classes of graph properties can be sped up as an immediate consequence of this representation scheme. To capture some other properties such as connectivity and bipartiteness, we develop an efficient algorithm to compute transitive closure of an induced subgraph. Finally, we define the notion of "speedup preserving" reductions for induced subgraph problem which enable us to transform an efficient algorithm for one property to one for another. An interesting aspect of our results is that simple function families such as OR, AND and threshold functions, seem to be all that one can use for a broad range of properties. In the remainder of this section, we assume that the space s is super-linear ($\Omega(m^{1+\epsilon})$) in the size m of the static input, and that it is sub-exponential (2^{n^δ}, $\delta < 1/3$) in the size n of the dynamic input. For example, $\log \frac{s}{m} = \Theta(logs)$, $O(n^2/\log^2 s + n \log s) = O(n^2/\log^2 s)$.

6.1 Standard Representations

As a first step towards designing our algorithms, we develop a useful representation scheme for graphs.

6.1.1 Standard Variable Representation (SVR)

Suppose we are given a set U of m boolean variables as a static input and a set $B \subseteq U$ of k boolean variables as dynamic input. Let $A \subseteq B$ denote the subset of variables in B which are set to true, $|A| = a$. We wish to determine A or perhaps some element $v \in A$, in an efficient manner. Our goal is to design a data structure using space s that allows us to do that. Let us first do so for a fixed dynamic input B.

Definition 14 (Simple Standard Variable Representation (SVR))
An OR-*restricted data structure scheme is called* simple SVR *of B if its cells correspond to the nodes of a binary tree constructed in the following manner. The root*

corresponds to an OR *of all k variables of B. The two subtrees attached to the root correspond to recursing this construction on a partition of variables into two equal halves.*

Lemma 20 *Simple SVR can determine A in time $t \leq 2a(1 + \log \frac{k}{a}) + 1$ using space $s = 2k - 1$.*

Proof: Traverse the underlying tree in a depth-first manner stopping any time we see a node with value 0. In the worst case we end up examining completely the first $\log a$ levels and then traversing at most a paths down, each being of length $\log \frac{k}{a}$. An easy computation gives the claimed bound; the space bound follows from the fact that the tree has $2k - 1$ nodes. ■

The following corollary is an easy consequence.

Corollary 21 *Using simple SVR we can*

1. *find some true variable in A (if $A \neq \emptyset$) in at most $1 + \log k$ steps;*

2. *for any fixed j determine either that $j > a$ or find some j true variables in time $O(j \log k)$;*

3. *if $a = o(k)$, we can completely determine A in $o(k)$ time.*

Now we would like to extend this approach to efficiently determine $A = A(B)$ for arbitrary dynamic input B, given our space constraints.

Definition 15 (Standard Variable Representation (SVR))
An OR*-restricted data structure is called a* SVR of U *if its cells correspond to the following representation. Partition U arbitrarily into $l = m/p$ clusters $C_1, \ldots, C_l$ of size $p = \log(s/m) = \Theta(\log s)$ each. For every non-empty subset S of any cluster, we store an* OR *of variables in S. Total space used is bounded by $l2^p < s$.*

Now given any subset $B \subseteq U$, we can use SVR to determine its subset of true variables $A = A(B)$ as follows. First determine for each $i \in [l]$, the i_{th} *projection* $P_i = C_i \cap B$. Define $\Gamma_m(B) = \min(m/p, |B|) = O(\min(m/\log s, |B|))$; it clearly upper bounds the number of non-empty projections. Since we have a simple SVR for each P_i, we can use the preceding algorithm to determine the set A_i of true variables in P_i. If we let $a_i = |A \cap P_i|$, the total time taken to determine each A_i is $O(1 + a_i \log p)$. Thus $A(B)$ can be determined in time $\Gamma_m(B) + O(a \operatorname{loglog} s)$. The lemma below summarizes these observations.

Lemma 22 *With space s, SVR can be used*

1. *to determine for any fixed j either that $j > a$ or find some j true variables in B in time $\Gamma_m(B) + O(j \operatorname{loglog} s)$, which is $O(\frac{m}{\log s})$ when $j = O(\frac{m}{\log s \operatorname{loglog} s})$. In particular, we can determine whether $A(B) = \emptyset$ in $\Gamma_m(B) = O(\frac{m}{\log s})$* oblivious *probes;*

2. *to determine $A(B)$ (and a) in time $\Gamma_m(B) + O(a \operatorname{loglog} s)$, which is $O(\frac{m}{\log s})$ when $a = O(\frac{m}{\log s \operatorname{loglog} s})$.*

6.1.2 Standard Graph Representation (SGR)

We now turn our attention to how these representation schemes may be applied to efficiently storing a graph G for the induced subgraph problem. Using $m = \binom{n}{2}$, we can directly use SVR as our representation scheme. But observe that our concern is only subsets of edges generated by induced subgraphs. This observation allows us to design what we call *standard graph representation (SGR)* which would allow us to save a $\log s$ factor over the previous scheme. The basic idea is that now instead of partition edges into clusters, we partition vertices into clusters. As a building block, we need the following representation scheme that applies to a fixed bipartite graph.

Simple SGR for bipartite graphs: Let $H(U, W, E_H)$ be a bipartite graph. We now construct a binary tree in a manner analogous to the simple SVR. The root contain an OR of edges in E_H. If either $|U|$ or $|W|$ is greater than one, say U, we split it into two equal-sized partitions U_1 and U_2, and recursively proceed on the graphs $H_1(U_1, W, E_{H_1})$ and $H_2(U_2, W, E_{H_2})$ where $E_{H_i} = (E_H) \cap (U_i \times W)$ for $i = \{1, 2\}$.

The two subtrees constructed above are attached to the root. Clearly, the total space used is $2|U||W| - 1$.

SGR for general graphs: We now show how to construct SGR of a graph $G(V, E)$ using space s. Partition V into $l = n/p$ clusters $V_1, ..., V_l$ where each V_i contains $p = (1/2)\log(s/n^2) = \Theta(\log s)$ vertices. First, for every subset U of each cluster, store a simple SVR for the edge variables in the graph induced by U. Next consider any pair of clusters, say V_i and V_j. For every $U_i \subseteq V_i$ and $W_j \subseteq V_j$, store a simple SGR for the bipartite graph induced by U_i and W_j. The total space used is $s = p^2 2^p + 2^{2p} n^2/p^2 < s$.

As an illustration of how SGR can be used, consider the NON-EMPTY property. Let $X_i = X \cap V_i$ be the i_{th} projection and let $\Gamma(X) = \min(|X|, n/p)$ denote the upper bound on the number of non-empty projections. For every non-empty projection X_i, we use the simple SVR to check for the presence of an edge inside E_{X_i} in 1 probe. Next, for every pair (X_i, X_j) of non-empty projections, we do the same using the simple SGR stored for the bipartite graph induced by X_i and X_j. Total time taken is bounded by $\Gamma(X) + \binom{\Gamma(X)}{2} = O(n^2/\log^2 s)$.

The following lemma summarizes some properties of SGR.

Lemma 23 *Let X be a subset of V, $G_X = (X, E_X)$ be the subgraph of G induced by X and $a = |E_X|$. Using SGR we can*

1. *for any fixed j determine either that $j > a$ or find some j edges in E_X in time $O(\Gamma^2(X) + j \operatorname{loglog} s + \Gamma(X)\log^2 s)$, which is $O(\frac{n^2}{\log^2 s})$ when $j = O(\frac{n^2}{\log^2 s \operatorname{loglog} s})$.*

2. *determine E_X (and a) in time $O(\Gamma^2(X) + a \operatorname{loglog} s + \Gamma(X)\log^2 s)$, which is $O(\frac{n^2}{\log^2 s})$ when $a = O(\frac{n^2}{\log^2 s \operatorname{loglog} s})$.*

6.1.3 Some Extensions

We conclude our discussion of these representations by indicating some useful extensions.

Representing Vertex Neighborhoods: In some applications, we will be interested in studying the neighborhood of a vertex v within a given query set X; let $N_X(v)$ denote this neighborhood. Observe that the standard graph representation contains almost all the information that we need to efficiently determine $N_X(v)$. The only information that we lack concerns the neighborhood of v within its cluster C_v. So we augment SGR as follows: for each vertex v and subset $S \subseteq C_v$, store an OR of edges between v and S. It is easy to see that this augmentation requires only a slight increase in space, namely $n2^p = o(s)$. We refer to this neighborhood information as the *vertex SVR* of v.

Generalizing to arbitrary functions: Instead of storing an OR of some variables in our representation, we can apply some function $g : \{0,1\}^* \mapsto R$ to this subset of variables and store the result. The resulting (simple) SVR (SGR) is called a (simple) SVR (SGR) w.r.t. g (if we do not specify g, we assume it is OR). We observe that we do not require g to be boolean, so the output of g might take more than 1 bit to describe (however, it is less than the number of variables to which this g is applied). For example, g can be integer addition, where both SGR and SVR will need cell size $O(\log\log s)$. It is easy to see that the space (overall number of bits) required by a SGR (SVR) w.r.t. g is still bounded by s, however. When talking about a "probe" we mean reading the whole output of g, but the time is still the overall number of bits read. The useful examples of functions g aside from OR and addition are AND and XOR.

6.2 Optimality of SGR/SVR

In this section, we try to give some intuition on why SGR and SVR only use clusters of size $O(\log s)$. For that, let us look at a class of representations where some function $g : \{0,1\}^* \mapsto R$ is applied to certain s subsets of the static input. And let us identify every cell in such a representation with the subset of variables to which g is applied. SGR and SVR are two examples of such representations. We address two natural conditions on such representations and show that SGR and SVR are optimal in terms of meeting these conditions.

We already noticed that *SGR (SVR) allows us to "store" s subsets and to partition every induced subgraph (every subset) of our static input into at most $O(\frac{n^2}{\log^2 s})$ $(O(m/\log s))$ disjoint subsets that are "stored" in our representation.* Such partitioning of the induced subgraph (subset) into disjoint "clusters" seems quite useful. For example, this seems to be the most natural way to compute splittable functions (see Theorem 13) over induced subgraphs (over subsets). We would like to address the question of whether SGR (SVR) produces the most efficient way to make such a partition, given the space constraint s. In other words, *what is the minimal t s.t. one can find s subsets of the static input allowing to partition every induced subgraph (every subset) into at most t disjoint subsets.* This is the (s,t)-spanning set question studied in Chapter 7, where we can even relax the requirement that the subsets are disjoint (i.e. we look at covers rather than partitions). We show in Theorem 17 (with $p = 2$ and the union operation) that to represent every subset of an m-element set we must have $t = \Omega(m/\log s)$, so *SVR is optimal* in this respect. Similarly, by Corollary 32, to represent every induced subgraph we need $t = \Omega(\frac{n^2}{\log^2 s})$, so *SGR is optimal* in this respect as well.

We also observe another property of SGR (SVR). *Together with every subgraph H of G (subset B of U) that it "stores", it also stores all distinct non-empty induced subgraphs of H (all non-empty subsets of B).* This also seems like a useful thing to do. For example, when partitioning the induced subgraphs into a union of "stored" subsets, we had one such partition $E_1, \ldots, E_t$ $(t = O(\frac{n^2}{\log^2 s}))$ for the whole edge set, and every induced subgraph had a partition where we simply projected each E_i onto this induced subgraph. Of course, we would like to have large subgraphs E_i so that t would be small. Thus we examine *what is the largest size of H (B) s.t. there are less than s distinct non-empty induced subgraphs H' of H (subsets B' of B).* The subset question is very easy, the number of B' is $2^{|B|} - 1$, so the largest B has size $|B| = O(\log s)$. Thus, to cover any subset of $\Omega(m)$ size by such subsets B, one would need $\Omega(m/\log s)$ "stored" subsets, and SVR exactly achieves that.

For the SGR we show

Proposition 24 *If H has $2k$ non-isolated vertices than it has at least $2^k - 1$ distinct non-empty induced subgraphs.*

Proof: It is easy to see that every graph with $2k$ non-isolated vertices has a vertex cover F of size at most k. Then any subset of these $2k$ vertices that properly contains F forms a non-empty induced subgraph, where all these subgraphs are distinct, since every $a \notin F$ has an edge to F. The number of such subsets of vertices is $2^{2k-|F|} - 1 \geq 2^k - 1$. ■

Hence, H can touch at most $O(\log s)$ vertices, so it has $O(\log^2 s)$ edges. Thus, to cover any subgraph induced by a subset of size $\Omega(n)$ one would need $\Omega(\frac{n^2}{\log^2 s})$ "stored" subgraphs, and SGR achieves this.

6.3 Applications of SGR

SGR and its variants are extremely useful in getting non-trivial upper bounds for various evasive graph properties. We now present some generic examples as well as some problem-specific techniques.

6.3.1 Computing Splittable Functions

Definition 16 (Splittable Function) *A function $g : \{0,1\}^* \mapsto R$ is called* splittable *if for any m and $x \in \{0,1\}^m$, for any partition of the m variables of x into subsets $A_1, \ldots, A_k$ we have that $g(x) = h((g(x_{A_1}), \ldots, g(x_{A_k})), (A_1, \ldots, A_k))$, for some function h.*

In other words, we can reconstruct $g(x)$ by knowing the values of g on an arbitrary partition of x. Some examples of splittable functions are AND, OR, XOR, addition.

Theorem 13 *Let g be a splittable function.*

- *Using SVR w.r.t. g and given a static input x of length m, we can compute g applied to an arbitrary subset B of variables of x in $\Gamma_m(B) = O(m/\log s)$* oblivious *probes.*

- *Using SGR w.r.t. g and given a graph G as a static input, we can compute g applied to a subgraph induced by X in* $\Gamma^2(X) = O(\frac{n^2}{\log^2 s})$ oblivious *probes.*

Proof: SVR w.r.t. g stores g applied to at most $\Gamma_m(B)$ non-empty projections of x. Reading the value of g applied to those projections and using splittability of g, the result follows. SGR w.r.t. g stores g applied to at most $\Gamma(X)$ subgraphs induced by non-empty projections of X and to at most $\binom{\Gamma(X)}{2}$ bipartite subgraphs induced by pairs of non-empty projections of X. This partitions G_X into at most $\Gamma^2(X)$ subgraphs. Reading the value of g applied to these subgraphs and using splittability of g, the result follows as well. Notice, both schemes are oblivious, since the projections are known in advance. ■

Corollary 25
$\mathcal{T}^{obl}_{\text{OR},s}(\text{NON-EMPTY})$, $\mathcal{T}^{obl}_{\text{AND},s}(\text{CLIQUE})$, $\mathcal{T}^{obl}_{\text{XOR},s}(\text{PARITY})$ *are all* $O(\frac{n^2}{\log^2 s})$.
$\mathcal{T}^{obl}_{\text{OR},s}(\text{NO-ISOLATED-VERTEX})$ *and* $\mathcal{T}^{obl}_{\text{AND},s}(\text{ALL-NEIGHBORS})$ *are* $O(\frac{n^2}{\log s})$.

Proof: Bounds for NON-EMPTY, CLIQUE and PARITY are immediate using splittability of OR, AND and XOR. For NO-ISOLATED-VERTEX, we check for every vertex of $v \in X$ that it is not isolated using the vertex SVR of v w.r.t. OR in $O(\frac{n}{\log s})$ probes, a total of $O(\frac{n^2}{\log s})$ oblivious probes. Similarly, for ALL-NEIGHBORS. ■

Corollary 26 *Using the family of threshold functions we can (adaptively) compute the number of edges $|E_X|$ for any X in* $O(\frac{n^2}{\log^2 s}\ \text{loglog}\ s)$ *time. In particular,* MAJORITY *and* PARITY *can be solved in this time using threshold functions.*

Proof: Storing $k+1$ threshold functions $T_{k,\_}$ and performing a binary search, we can compute the number of true variables among k variables $x_1, \ldots, x_k$ in $1+\log k$ probes. On the other hand, we can compute $|E_X|$ in $O(\frac{n^2}{\log^2 s})$ oblivious probes of $O(\text{loglog}\, s)$ bits each, by using SGR w.r.t. addition. Each probe returns the number of edges in a small subgraph of $O(\log^2 s)$ edges. We simply replace each $O(\text{loglog}\, s)$-bit cell of the SGR containing the number of edges with $O(\log^2 s)$ threshold functions. This still uses space at most s as is easily seen. When we need the number of edges in some subgraph, we perform the binary search using threshold functions in $O(\text{loglog}\, s)$ steps. ■

It is interesting to observe that by Lemma 19, for the family $\mathcal{F}$ of threshold functions, $\mathcal{T}^{obl}_{\mathcal{F}}(\text{PARITY}) = \binom{n}{2}$, so adaptivity is crucial.

6.3.2 Edge-Separated Properties

Definition 17 *A property P is called $g(n)$-*edge separated *if $|E| \geq g(n)$ fixes the property to true or false.*

For example, TREE, FOREST, k-MATCHING, k-VERTEX COVER (for constant k) are $O(n)$-edge separated properties. Using Lemma 23, we can either recognize that $|E_X|$ is more than $g(n)$ or obtain E_X in time $O(n^2/\log^2 s + g(n)\,\text{loglog}\, s)$. This clearly suffices to determine any $g(n)$-edge separated property. Thus we have the following theorem.

Theorem 14 *For any $g(n)$-edge separated P, $\mathcal{T}_{\text{OR},s}(P) = O(\frac{n^2}{\log^2 s} + g(n)\,\text{loglog}\, s)$.*

6.3.3 Computing BFS/DFS Forest

Computing BFS/DFS forest is often an integral part of many graph algorithms. Here we show how they can be sped up using SGR.

Theorem 15 *Using SGR (w.r.t. OR) we can compute a BFS or DFS forest of G_X in time $O(\frac{n^2}{\log s})$.*

Proof: We just follow the standard BFS/DFS algorithm of greedily discovering new vertices. Let us concentrate for BFS, for concreteness. Having a current set of vertices $Y \subseteq X$ in the forest and the frontier F, we look at each $v \in F$ and find all its neighbors to $X \backslash Y$. Using the vertex SVR of v, this takes at most $\frac{n}{\log s} + n_v\,\text{loglog}\, s$ steps, where n_v is the number of new neighbors discovered. Summing over all the vertices and using the fact that $\sum n_v \leq |X| \leq n$ we get $O(\frac{n^2}{\log s})$ algorithm. Similarly, for DFS. ■

6.3.4 Transitive Closure Computation

The ability to efficiently compute transitive closure of a subgraph induced by a subset of vertices is a useful primitive for many graph properties (e.g. connectivity). And among other things, BFS/DFS forest gives us the transitive closure of G_X. On the other hand, if we do not need the extra structure of the BFS/DFS forest, it turns out that we can find the transitive closure faster by searching not vertex by vertex but cluster by cluster!

Theorem 16 *For any $X \subseteq V$, we can compute some spanning forest (and thus the transitive closure) of G_X in time $O(\frac{n^2}{\log^2 s} \operatorname{loglog} s)$.*

Proof: Let $c \leq n/\log s$ be the number of non-empty projections of X, which we call $X_1, \ldots, X_c$. We let G_i be the subgraphs induced by X_i and H_{ij} - the bipartite graph induced by X_i and X_j. First, we read one be one all the edges inside G_i, at most $cp^2 = O(n \log s) = o(n^2/\log^2 s)$ of them. Then we find connected components (including their spanning trees) of each G_i, initializing our spanning forest F to be the union of all these spanning trees $T_1, \ldots, T_t$. The only possible connections between T_k's that might contract our forest are now inside H_{ij}, $i \neq j$. In the procedure below, we will continue to find an edge between two currently distinct trees in F (such edge is called a *contracting edge*), thus contracting those two trees together, until no more contracting edges exist. We will look for contracting edges by examining edges in all H_{ij}'s in the order $i = 1 \ldots (c-1)$, $j = (i+1) \ldots c$. We continue examining the current H_{ij} until it has no new contracting edges. We strees that we update F whenever a new contracting edge is found, so at most $n-1$ contractions are possible. The correctness of this algorithm is clear, we only have to describe how to look for a contracting edge in H_{ij} given the current state of F. Assuming for a second that we know how to find a contracting edge in H_{ij} (if it exists) in $O(\operatorname{loglog} s)$ steps, it is easy to justify the running time. There will be at most $n-1$ succesful searches overall and at most one unsuccessful search for every H_{ij}, a total of $O(n + c^2) = O(\frac{n^2}{\log^2 s})$ searches, proving $O(\frac{n^2}{\log^2 s} \operatorname{loglog} s)$ running time.

Thus, it remains to describe how to find a contracting edge. Assume w.l.o.g. that $i = 1$, $j = 2$. We observe that for every $Y \subseteq X_1$ and $Z \subseteq X_2$, our SGR stores the information on whether there is an edge between Y and Z. Let F consist of trees $T_1, \ldots, T_t$, and C_k (D_k) be the set of vertices of T_k inside X_1 (X_2). Also let $L = \{k | C_k \neq \emptyset\}$ be the set on non-empty trees in X_1, $R = \{k | D_k \neq \emptyset\}$ - in X_2. Clearly, both L and R have at most $O(\log s)$ trees. The set of contracting edges is exactly the set of edges between C_{k_1} and D_{k_2} for $k_1 \neq k_2$, $k_1 \in L$, $k_2 \in R$. We observe that there might or might not be an edge between C_k and D_k, but we really do not care since they are not contracting. We can clearly "identify" all the vertices of non-empty C_k (D_k) into a "single vertex", since they are parts of the same tree already. Thus, every probe we are going to make can be thought as testing 2 *disjoint* subsets $A \subseteq L$ and $B \subseteq R$, corresponding to making $Y = Y_A = \cup_{k \in A} C_k$, $Z = Z_B = \cup_{k \in B} D_k$. We require $A \cap B = \emptyset$, since we do not want to include the same tree on both sides. We call such a probe $[A, B]$. Once we get any such probe returning true, we can use the simple SGR for the bipartite graph (Y_A, Z_B), make a binary search and find the contracting edge in $O(\text{loglog}\, s)$ steps. So we can just describe how to partition all possible contracting edges into few complete bipartite graphs (Y_A, Z_B), $A \cap B = \emptyset$.

First, we make two probes $[L, R \backslash L]$ and $[L \backslash R, R]$ which test for a contracting edge between the trees present in X_1 but not X_2 and all the trees in X_2, and vice versa. If any of them is true, we are done. Otherwise, if we let $U = L \cap R$ (notice $|U| = O(\log s)$), the *only* possible contracting edges are exactly between "points" C_{k_1} and D_{k_2} for $k_1, k_2 \in U$, $k_1 \neq k_2$. Thus, the queries we make will be of the form $[A, U \backslash A]$, where $A \subseteq U$, and we need to find a sequence of queries $A_1, \ldots, A_q$, $A_i \subseteq U$, s.t. $A_i \times (U \backslash A_i)$ cover every point (k_1, k_2) for $k_1 \neq k_2$, $k_1, k_2 \in U$. Hence, our problem reduces to finding a small q s.t. there are sets $A_1, \ldots, A_q \subseteq U$ s.t. for any $k_1, k_2 \in U$, $k_1 \neq k_2$, there is A_i s.t. $k_1 \in A_i$, but $k_2 \notin A_i$. Such a collection $A_1, \ldots, A_q$ is called a *separating family of U*.

Lemma 27 *A set U of size r has a separating family of size $2\lceil \log r \rceil = O(\log r)$.*

Proof: Let $\beta = \lceil \log r \rceil$. We can write any $z \in U$ in binary $z = z(1), \ldots, z(\beta)$. We let $K_i = \{z | z(i) = 0\}$, $N_i = \{z | z(i) = 1\}$, $i \in [\beta]$. We claim that $K_1, \ldots, K_\beta, N_1, \ldots, N_\beta$ is a separating family for M. Take any $a \neq b$. There is a bit position i where they differ. Assume $a(i) = 0$, $b(i) = 1$. Then K_i separates a from b, as $a \in K_i$, $b \notin K_i$. Similarly, when $a(i) = 1$, $b(i) = 0$ we get that N_i separates a from b, as $a \in N_i$, $b \notin N_i$. ■

In our case $r = |U| = O(\log s)$, so there is a separating family for U of size $O(\log\log s)$. Testing $[A, U \backslash A]$ for A in the separating family will complete testing for a contracting edge in $O(\log\log s)$ steps. ■

Corollary 28

$\mathcal{T}_{\mathrm{OR},s}$(CONNECTIVITY), $\mathcal{T}_{\mathrm{OR},s}$(NO-ISOLATED-VERTEX), $\mathcal{T}_{\mathrm{AND},s}$(ALL-NEIGHBORS) are all $O(\frac{n^2}{\log^2 s} \log\log s)$.

Proof: Transitive closure of G_X is clearly sufficient for deciding CONNECTIVITY and NO-ISOLATED-VERTEX. To deal with ALL-NEIGHBORS, we compute transitive closure of $\overline{G_X}$ using AND instead of OR. ■

We observe by Corollary 25 that $\mathcal{T}^{obl}_{\mathrm{OR},s}$(NO-ISOLATED-VERTEX) $= O(\frac{n^2}{\log s})$. We will come back to that in Chapter 7.

6.3.5 Speedup-Preserving Reductions

While the class of properties we were able to speed-up might seem too small, we now develop the notion of speedup-preserving reductions will allow us to capture many properties via the above framework. Speedup preserving reduction allows us to apply efficient algorithms developed for one property to the other.

Definition 18 (Speedup-Preserving Reduction (S-Reduction)) *A property P is said to be* S-reducible *to a property P', denoted $P \propto_S P'$, if there are maps g, h and a constant c such that*

- *g takes as input a graph $G(V, E)$ and outputs a graph $G'(V', E')$ with $|V'| \leq c|V|$, and*

- $h : 2^V \to 2^{V'}$ *is such that such that for any* $X \subseteq V$, $P(G_X) \iff P'(G'_{h(X)})$ *(or* $P(G_X) \iff \overline{P'(G'_{h(X)})}$*).*

If $P \propto_S P'$, then it is readily seen (as $|V'| = O(|V|)$) that $\mathcal{T}_s(P) = O(\mathcal{T}_s(P'))$.

As an example of an S-reduction, we sketch a reduction from graph bipartiteness to graph connectivity. For any static input G, we construct a transform as follows. Define a graph $G' = (V', E')$, where $V' = V \times \{0, 1\}$, and E' only contains edges that connect the "0-side" to the "1-side" as specified by $E'([v, 0], [w, 1]) = E(v, w)$; notice that G' is bipartite. The map h is given by $h(X) = X \times \{0, 1\}$. We claim now that G_X is not bipartite (has an odd cycle) if and only if for some $[v, 0] \in V'$, there is a path connecting $[v, 0]$ to $[v, 1]$. To see this, consider an odd cycle $(v_1, \ldots, v_{2k+1})$ in G_X. Then $G'_{h(X)}$ contains the path

$$[v_1, 0] \to [v_2, 1] \ldots \to [v_{2k+1}, 0] \to [v_1, 1].$$

Conversely, stripping the second component from any path from $[v, 0]$ to $[v, 1]$ produces an odd cycle in G_X. Thus graph bipartiteness S-reduces to graph connectivity. Moreover, since edges of G' are just some edges of G, we can say that $\mathcal{T}_{\text{OR},s}(\text{NON-BIPARTITE}) = O(\mathcal{T}_{\text{OR},s}(\text{CONNECTIVITY})) = O(\frac{n^2}{\log^2 s} \operatorname{loglog} s)$. It is interesting to observe that the two "more direct" algorithms for bipartiteness run in $O(\frac{n^2}{\log s})$ time: one using DFS forest and then trying to detect an odd cycle by checking back edges, the other - using BFS forest and checking for an edge between two vertices on the same level.

Chapter 7

(s, t)-Spanning Set

In this section we study an abstract problem that highlights the issue of space-time tradeoffs and is interesting in its own right. Here is the approximate description of the problem. Suppose we are given a set M with some operation $\star$ and M has some "spanning set" B of size m i.e. every element of M is "expressible" using some elements of B. Minimal such B is usually called a *basis*. What a basis allows us to accomplish is to "store" fewest possible elements needed to express every possible "query" element, by possibly using all m elements of B. Assume now that we are willing to store more than m elements in B but seek to express any element of M using at most t elements of B, where $t << m$. If we find such a B of size s, we call it a (s, t)-spanning set for M. More generally, we can talk about an (s, t)-spanning set for some subset $W \subseteq M$. The question is what is the optimal tradeoff between s and t. After obtaining some general bounds for the (s, t)-spanning set question, we find how a particular version of it connects to non-deterministic and oblivious cell probe complexity under the families of AND and OR functions. This will allow us to give *purely combinatorial* descriptions of such measures as $\mathcal{NT}^{0}_{\mathrm{OR},s}(f)$, $\mathcal{T}^{obl}_{\mathrm{OR},s}(f)$, $\mathcal{T}^{obl}_{\mathrm{OR}}(f)$ (similarly, for the AND family). We then apply these results to the induced subgraph problem.

7.1 Basic Setup and General Results

Let F be some set equipped with two operations: addition $+$ and multiplication $*$. Assume that addition is commutative, associative and has an additive identity 0, while multiplication is associative and has an identity element $u \in F$. We also assume the left distributive law $a * (b + c) = (a * b) + (b * c)$ holds. We let $M = F^m$ be the m^{th} power of F, i.e. $M = \{(a_1, \dots, a_m) | a_i \in F\}$. We define addition $(a_1, \dots, a_m) + (b_1, \dots, b_m) = (a_1 + b_1, \dots, a_m + b_m)$ and multiplication by a scalar $a * (b_1, \dots, b_m) = (a * b_1, \dots, a * b_m)$. When F is a field, for example, M is just an m-dimensional vector space over F, but we will not restrict ourselves to this case. We say that $v \in M$ is *expressible* in terms of $v_1, \dots, v_k \in M$, if there are $a_1, \dots, a_k \in F$ s.t. $v = a_1 * v_1 + \dots + a_k * v_k$. We observe that M has a *basis* B of m elements, i.e. every $v \in M$ is uniquely expressible in terms of elements of B. An example of B is the canonical basis, where "unit" vector $u_i \in M$ has 0 in all positions other than i where it has u, and $v = (a_1, \dots, a_m) = \sum a_i * u_i$.

Definition 19 *An (s, t)-*spanning set *for $W \subseteq M$ is a collection S of s elements of M with the property that for any $v \in W$, v is expressible in terms of at most t elements of S (then we call v t-*expressible *in terms of S). When $s = t$ we call S simply a* spanning set *of W. S is* minimum (s, t)-spanning set *for W if s is optimal given t.*

The canonical basis of M is a (minimum) (m, m)-spanning set for M. Let $|F| = p$, so $|M| = p^m$. We start with a sharp bound for s when $W = M$.

Theorem 17 *Any minimum (s, t)-spanning set for M satisfies:*

- *If $m\frac{p-1}{p} < t < m$ and F is a group under addition, then $s = m + 1$.*
- *If $1 \leq t \leq m\frac{p-1}{p}$ then*

$$\frac{1}{e(p-1)} tp^{m/t} \leq s \leq t(p^{m/t} - 1)$$

In particular, $s = \Theta(tp^{m/t})$, $t = \Theta(\frac{m}{\log_p s})$. For F being a field, the upper bound can be improved to $s \leq t(p^{m/t} - 1)/(p - 1)$.

Proof: Let $m\frac{p-1}{p} < t < m$ and F be a group under addition. Let $S = \{u_1, \ldots, u_m, b = u_1 + \ldots + u_m\}$, $|S| = m + 1$. Take any $v \in M$, $v = (a_1, \ldots, a_m)$. Since $|F| = p$, at least $\frac{m}{p}$ of a_i's are the same, say, $a_1 = a_2 = \ldots = a_{m/p} = a$. Then $v = a * b + (a_{m/p+1} - a) * u_{m/p+1} + \ldots + (a_{m/p+1} - a) * u_{m/p+1}$ (we can subtract a as F is a group). Since $t < m$, it is impossible to have $|S| = m$, so S in minimum indeed. Let $1 \leq t \leq m\frac{p-1}{p}$. For the upper bound we construct the following (s,t)-spanning set. Divide canonical basis B into t blocks of size $\frac{m}{t}$ each. For each block store in S all possible vectors expressible in terms of the elements of the block (except u). This gives a total of $t(p^{m/t} - 1)$ elements in S. Then clearly every element of M is (uniquely) expressible in terms of at most t of elements in S: simply take the normal representation of v in terms of B and replace all basis vectors of each block by a single vector. In the case when F is a field, we can split each block into equivalence classes consisting of $p - 1$ vectors obtainable from each other by a non-zero scalar multiplication, and store only 1 vector from each equivalence class.

For the lower bound, using the known inequality for the partial binomial sum (see [7], pp. 55), the total possible number of elements expressible in terms of t out of some s elements is at most (all logarithms below are base p) :

$$\sum_{k=0}^{k=t} (p-1)^k \binom{s}{k} \leq p^{t\log(p-1)+s\log s - t\log t - (s-t)\log(s-t)} = A \tag{7.1}$$

We must have $A \geq |M| = p^m$, since we want to express all elements of M. Hence we need

$$\log A = t\log(p-1) + s\log s - t\log t - (s-t)\log(s-t) \geq m \tag{7.2}$$

Assume, $s = at \cdot p^{m/t}$, where $a = \frac{1}{e(p-1)}$. We will show then that (7.2) is not satisfied. Indeed,

$$\begin{aligned} \log A &= t\log(p-1) + atp^{m/t}\log(atp^{m/t}) - t\log t - \\ &\quad t(ap^{m/t} - 1)(\log t + \log(ap^{m/t} - 1)) \\ &= t\log(p-1) + p^{m/t}at(\frac{m}{t} + \log a) + p^{m/t}at\log t - t\log t(ap^{m/t}) - \end{aligned}$$

$$\begin{aligned}
& t(ap^{m/t}-1)\log(ap^{m/t}-1)) \\
= \;& t\log(p-1)+p^{m/t}(am+at\log a)+t\log(ap^{m/t}-1)- \\
& tap^{m/t}\log(ap^{m/t}-1) \\
= \;& t\log(p-1)+t\log(ap^{m/t}-1)+p^{m/t}(am+at\log a-at(\frac{m}{t}+\log a)- \\
& at\log(1-\frac{1}{ap^{m/t}})) \\
< \;& t\log(p-1)+t(\log a+\frac{m}{t})-p^{m/t}at\log(1-\frac{1}{ap^{m/t}}) \\
< \;& t\log(p-1)+m+t\log a+p^{m/t}(at\frac{1}{ap^{m/t}})\log e \\
= \;& m+t\log(p-1)+t\log a+t\log e=m+t\log(ae(p-1))=m
\end{aligned}$$

■

Hence, we cannot save more than $\log s$ factor in time by allowing more space. Notice, in the lower bound proof, the only fact we used about M was that $|M|=p^m$. So we get in the same way

Corollary 29 *For $t \le m\frac{p-1}{p}$, any (s,t)-spanning set for $W \subseteq M$ with $n=\log_p|W|$ satisfies $s \ge \frac{1}{e(p-1)}tp^{n/t}$, i.e. $t=\Omega(\frac{n}{\log_p s})$.*

The bound in Corollary 29 is the best possible generic lower bound for W, since we can take W to be the subspace spanned by $u_1,\dots,u_n$ and apply to it the upper bound in Theorem 17. On the other hand, there are 2 trivial (s,t)-spanning sets for any W: for $t=1$ we can take $s=|W|=p^n$ by storing all the elements in W, or we can use the (s,t)-spanning set for the whole M as described in Theorem 17, getting $s=\Theta(tp^{m/t})$. What we show is that for a vast majority of $W \subseteq M$, $\log_p|W|=n$, one of these two extremes is essentially optimal, so the lower bound of $t=\Omega(n/\log s)$ is unachievable when $n << m$.

Theorem 18 *For a random $W \subseteq M$, $|W|=p^n$, with high probability, for any (s,t)-spanning set for W,*

- *If $t=O(\frac{m}{n})$, then $s=\Omega(|W|)=\Omega(p^n)$ (i.e. to get "small" t the trivial scheme with $s=|W|$, $t=1$ is nearly optimal).*

- *If $t = \Omega(\frac{m}{n})$ then $s = \Omega(tp^{\Omega(m)/t})$ (i.e. it is nearly optimal to take the (s,t)-spanning set for the whole M). Thus $t = \Omega(\frac{m}{\log_p s})$.*

Proof: When $n > \frac{m}{2}$, the bound in Corollary 29 already gives us the desired result for all W, so let $n \leq \frac{m}{2}$. There are $\binom{p^m}{s}$ possible collections S of s vectors, each of them is capable of t-representing at most A vectors, as defined in (7.1). Those A vectors contain at most $\binom{A}{p^n}$ possible subsets W of size p^n. Hence, using space s, the total number of subsets W with an (s,t)-spanning set is at most $\binom{p^m}{s}\binom{A}{p^n} \leq p^{ms}A^{p^n}$. On the other hand, the total number of W's is $\binom{p^m}{p^n} \geq (\frac{p^m}{p^n})^{p^n} \geq p^{\frac{m}{2}p^n}$, as $n \leq \frac{m}{2}$. Thus, if space s would be sufficient for all W of size p^n, we must have $p^{ms}A^{p^n} \geq p^{\frac{m}{2}p^n}$, i.e. $ms + p^n \log_p A \geq \frac{m}{2}p^n$ or $\frac{s}{p^n} + \frac{log_p A}{m} \geq \frac{1}{2}$. Hence, either $s = \Omega(p^n) = \Omega(|W|)$, or $\log_p A = \Omega(m)$. In the second case, carrying out the same computation as in the proof of Theorem 17, we would get $s = \Omega(tp^{\Omega(m)/t})$, as required. The threshold between these two space bounds is $t = \Theta(\frac{m}{n})$. High probability part follows easily from above.
■

The result should not come as a surprise, since if W is irregular and unstructured, we should not expect to have an "efficient" (s,t)-spanning set for W, other than 2 extremes: store W directly, or store an (s,t)-spanning set for the whole M. This raises a question of getting optimal (s,t)-spanning set for some explicit M and W. We will do it soon in connection with graph properties.

7.2 Computing Monotone Functions Using AND/OR Families

We now describe how (s,t)-spanning set gives a combinatorial characterization of non-deterministic and oblivious computations for evaluating a monotone function f, under AND/OR-restricted data structures. From here on, we focus on (s,t)-spanning set for $M = F^m$ where $F = \{0,1\}$ (i.e. $p = 2$), "addition" is the OR operation and "multiplication" is the AND operation. Unwinding the definition, M is simply the power set of $[m]$ with the *union operation* on it. An (s,t)-spanning set for some

$W \subseteq M$ is a collection S of s subsets of $[m]$ such that every set $A \in W$ is a union of at most t sets in S; we say that S *(s,t)-covers* W. When $s = t$ we simply say that S *covers* W. Let $f : Y \times Q \mapsto \{0,1\}$ be a *monotone* (in y for any fixed q) function that we wish to compute using an AND-restricted data structure $D = \{c_1, \ldots, c_s\}$. Let $E(c_i)$ be the set of variables that occur in c_i, and $S(D) = \{E(c_1), \ldots, E(c_s)\}$ be the collection of subsets of $\{0,1\}^m$ resulting from D.

Definition 20 *For a function $g : \{0,1\}^m \mapsto \{0,1\}$, let $\mathcal{C}^1[g]$ be the set of all min-1-certificates for g viewed as subsets of $[m]$. Given a query $q_0 \in Q$, we let $\mathcal{C}^1(q_0) = \mathcal{C}^1[f|_{q=q_0}]$ and $W_{\text{AND}} = \bigcup_{q \in Q} \mathcal{C}^1(q)$. Analogous definitions can be made for* OR *case by replacing min-1-certificates with min-0-certificates.*

Lemma 30 *Let D be any* AND*-restricted (*OR*-restricted) data structure scheme of size s, and let $f(y,q)$ and $g(y)$ be two monotone functions. Then*

- *D can be used to* non-deterministically *check $f(y,q) = 1$ (0) in t probes for any query $q \in Q$ if and only if $S(D)$ (s,t)-covers W_{AND} (W_{OR}).*
- *D* obliviously *computes $g(y)$ if and only if $S(D)$ covers $\mathcal{C}^1[g]$ $(\mathcal{C}^0[g])$.*

Proof: Suppose D can be used to non-deterministically verify $f(y,q) = 1$. Take any $q_0 \in Q$, $C_1 \in \mathcal{C}^1(q)$ and let $y_0 = y_{C_1}$ (i.e. just the variables in C_1 are set to true), so $f(y_0, q_0) = 1$. Assume $c_1, \ldots, c_t$ are the cells we guessed that let us verify $f(y_0, q_0) = 1$. The value of c_i is 1 if and only if $E(c_i) \subseteq C_1$. Assume such c_i's are $c_1, \ldots, c_k$, $k \leq t$. Let $A = \cup_{i=1}^k E(c_i)$, so $A \subseteq C_1$ and $y = y_A$ is consistent with the answers we got. If $A \neq C$ then $f(y_A, q_0) = 0$, so we could not have concluded that $f = 1$, so $A = C$, and $S(D)$ is an (s,t)-spanning set for W_{AND}. Conversely, if $S(D)$ is an (s,t)-spanning set for W_{AND}, given any q_0 we can guess $C_1 \in \mathcal{C}^1(q_0)$ that makes $f = 1$ (if one exist) and read the at most t cells that cover it in $S(D)$. We accept if and only if we get all 1 answers. Clearly, this correctly non-deterministically verifies that $f = 1$. The oblivious case is essentially the same as above. ■

As an immediate corollary we get the desired combinatorial description of non-deterministic and oblivious cell probe complexities.

Theorem 19 *For any monotone function f,*

- $\mathcal{NT}^1_{\text{AND},s}(f)$ *($\mathcal{NT}^0_{\text{OR},s}(f)$) is a smallest t s.t. there is an (s,t)-spanning set for W_{AND} (W_{OR}).*
- $\mathcal{T}^{obl}_{\text{AND},s}(f)$ *($\mathcal{T}^{obl}_{\text{OR},s}(f)$) is a smallest t s.t. there are s sets containing a spanning set of size t for every $\mathcal{C}^1(q)$ ($\mathcal{C}^0(q)$) where $q \in Q$.*
- $\mathcal{T}^{obl}_{\text{AND}}(g)$ *($\mathcal{T}^{obl}_{\text{OR}}(g)$) is the size of the minimum spanning set for $\mathcal{C}^1[g]$ ($\mathcal{C}^0[g]$).*

An interpretation of this theorem is that when the computation needs to verify all min-1-certificates (min-0-certificates), then the amount of work needed to be done can be characterized combinatorially.

7.3 Applications To Graph Properties

We now apply Theorem 19 to the induced subgraph problem. We already saw in Theorem 6 that for any evasive property, $\mathcal{T}_{\text{OR}}(P) = \Omega(n^2/\log^2 s)$; this was shown using the stabilization technique. The same technique also yields $\mathcal{NT}^0_{\text{OR}}(P) = \Omega(N^0(P)/\log^2 s)$ for any property P. For any non-trivial monotone property, we now re-establish this result via a very different approach, namely, by using (s,t)-spanning set. One advantage of this approach over the stabilization technique is that it applies to arbitrary functions and not only induced subgraph problem. We will also show $\mathcal{T}^{obl}_{\text{OR}}(P) = \Omega(n^2)$ bounds for many natural properties P. Some of these properties, e.g. CONNECTIVITY, have $\mathcal{T}_{\text{OR}}(P) = \Theta(n \log n)$ [3] or even $\mathcal{T}_{\text{OR},s}(P) = O(\frac{n^2}{\log^2 s} \text{loglog}\, s)$. Thus, such properties are speedable adaptively using OR but not obliviously. Hence, we once again obtain a separation between adaptive and oblivious computation, this time for AND/OR-restricted function families. We conclude the section by showing that for $P =$ NO-ISOLATED-VERTEX we get $\mathcal{T}^{obl}_{\text{OR},s}(P) = \Theta(\frac{n^2}{\log s})$ while $\mathcal{T}_{\text{OR},s}(P) = O(\frac{n^2}{\log^2 s} \text{loglog}\, s)$. So this property is speedable both adaptively and obliviously, but one does better adaptively. This will involve both stabilization technique and estimating the size of a certain spanning set. In the sequel we will also develop general techniques on how to show bounds on the size of the (s,t)-spanning set under the union operation. In particular, we show how to get tight bounds on the (s,t)-spanning set for the set of graphs isomorphic to a given one, which is of independent interest. The bounds we obtain are tight and are generally better than the counting results of Corollary 29.

We now have $m = \binom{n}{2}$ and M is the set of all n-vertex graphs. In this case $\mathcal{C}^z(X)$ is a collection of min-z-certificates for P on X, $W_{\text{AND}} = \bigcup_X \mathcal{C}^1(X)$, $W_{\text{OR}} = \bigcup_X \mathcal{C}^0(X)$. We let $\mathcal{C}^z = \mathcal{C}^z(V)$.

Definition 21 *For a graph $G = (V,E)$ we let $I(G)$ be the set of all graphs isomorphic to G (under relabeling of vertices of V). We call the set of non-isolated vertices in G an* active set *and its cardinality - a* touching number *of G.*

Because of invariance of any graph property under relabeling of vertices, we have that if $G \in W_{\text{AND}}$, then $I(G) \subseteq W_{\text{AND}}$ (same for W_{OR}). Also, $G \in \mathcal{C}^z$ implies $I(G) \subseteq \mathcal{C}^z$. Above observations suggest that we should develop technique on finding an optimal (s,t)-spanning set for $W = I(G)$. We also observe the special property of the union operation: *the only graphs H that can be useful in representing G are subgraphs of G.* Thus, we expect that a "large" graph should be useful for (is a subgraph of) only for a very small fraction of graphs in $I(G)$. As an example, we let G be an $\frac{n}{2}$-clique. Then any graph H touching k vertices is a subgraph of exactly all the $\frac{n}{2}$-cliques whose active set contains the active set set of H, which forms a $\binom{n-k}{n/2-k}/\binom{n}{n/2} \leq 2^{-k}$ fraction of $\frac{n}{2}$-cliques. In fact, similar bound turns out to be true for a much larger class of graphs, but not for all, as is illustrated by the following example. Let G be the complement of an $\frac{n}{2}$-clique. Then the "star" on n-vertices with touching number n is nevertheless useful for $\binom{n-1}{n/2-1}/\binom{n}{n/2} = \frac{1}{2}$ fraction of graphs in $W = I(G)$! In fact, n star graphs form an $(n, n/2)$-spanning set for $I(G)$. As we will see, the problem with this G is the fact that it has vertices of large degree.

Lemma 31 *If G has maximum degree at most cn, for $c < 1$, then any graph H touching k vertices occurs as a subgraph in at most $2^{-\Omega(k)}$ fraction of graphs in $I(G)$.*

Proof: Let p be the fraction in question. Let H' be a graph on k (ordered) vertices equal to H restricted to its k non-isolated vertices. As H' has no isolated vertices, it has $l \geq k/2$ edges $e_1 \dots e_l$ in its spanning forest F. We can assume w.l.o.g. that H'

(and H) has no other edges except for $e_1, \ldots, e_l$, as the fewer edges H has, the more graphs it can be a subgraph of. We also choose a convenient order on e_i's and on vertices in H' by ordering them in the *depth first search order*: each time we traverse an edge of F in the DFS, we add this edge to the list of edges and add the new endpoint to the list of vertices. When we go to a new component, we just add the first vertex of this component to the list of vertices. Let $Aut(G) \subseteq S_n$ be the set of automorphisms of G ($\pi \in S_n$ s.t. $\pi(G) = G$). Consider first the case $k \leq \frac{1-c}{2}n$. We have:

$$\begin{aligned}
p &= \Pr_{G' \in I(G)}(H \subseteq G') = \frac{|\{G' \in I(G) | H \subseteq G'\}|}{|I(G)|} = \frac{|\{G' \in I(G) | H \subseteq G'\}| \cdot |Aut(G)|}{|I(G)| \cdot |Aut(G)|} \\
&= \frac{|\{\pi \in S_n | H \subseteq \pi(G)\}|}{n!} = \Pr_{\pi \in S_n}(H \subseteq \pi(G)) = \Pr_{\pi \in S_n}(\pi^{-1}(H) \subseteq G) \\
&= \Pr_{\pi \in S_n}(\pi(H) \subseteq G) = \Pr_{X \subseteq V, |X|=k}(H' \subseteq G_X) = \Pr_{X \subseteq V, |X|=k}(e_1, \ldots, e_l \in E_X) \\
&= \prod_{i=1}^{l} \Pr_{X \subseteq V, |X|=k}(e_i \in E_X | e_1, \ldots, e_{i-1} \in E_X) \leq (\max_{a \in V} \frac{degree(a)}{n-k+1})^l \\
&\leq (\frac{cn}{n-k})^l \leq (\frac{2c}{1+c})^{k/2} = 2^{-\Omega(k)}
\end{aligned}$$

Let us elaborate on the above sequence of inequalities. Picking a random $G' \in I(G)$ and testing it has H as its subgraph, is equivalent to picking a random isomorphism π of vertices and testing that H is a subgraph of $\pi(G)$, since $|\{G' \in I(G) | H \subseteq G'\}| \cdot |Aut(G)| = |\{\pi | H \subseteq \pi(G)\}|$, and $|I(G)| \cdot |Aut(G)| = |S_n| = n!$. But $H \subseteq \pi(G)$ $\Longleftrightarrow$ $\pi^{-1}(H) \subseteq G$. Since π is random, so is π^{-1}, and applying a random permutation to H touching k vertices is the same as choosing k (*ordered*) random vertices in V forming a subset X, looking at the induced subgraph G_X, and testing $H' \subseteq G_X$, i.e. all $e_i \in E_X$, which we then rewrite using the conditional probability. Next comes the crucial observation. Since X is ordered, we can view a random choice of X as choosing the k vertices of X one by one in the same DFS order as the vertices of H' occur, and whenever H' had a new edge e_i in its forest, we check that we get a corresponding edge in G. Thus, the conditional probability estimates the following. Assume we already chose some $r < k$ vertices in V and we know the presence of some edges between them. Let a be some special vertex among those r vertices. We now choose a random b out of the remaining $n - r \geq n - k + 1$ vertices and ask the probability that (a, b) form an edge in G. If we did not know anything about the previous edges, this probability would be clearly bounded by $\frac{degree(a)}{n-r} \leq \frac{degree(a)}{n-k+1}$. But the *presence* of some edges adjacent to a among the r previously selected vertices only decreases the needed probability, so our probability is indeed bounded by $\frac{degree(a)}{n-k+1}$. Since we do not know what a is, we have to take the worst possible choice of $a \in V$. The remaining computations are clear, they use the facts that $k \leq \frac{1-c}{2}n$, $l \geq k/2$, and $c < 1$.

To get the desired bound for arbitrary $k > \frac{1-c}{2}n$, we simply let $\tilde{H}$ be the subgraph of H induced by the first $(1-c)n/2$ vertices in the DFS traversal of H. Clearly, $\tilde{H}$ touches f or $f - 1$ vertices and is a subgraph of H, where $f = \frac{(1-c)n}{2} \geq \frac{(1-c)k}{2}$. Since a subgraph of H can be a subgraph of only more graphs in $I(G)$, we get by the preceding analysis that $\tilde{H}$, and thus H, is useful for at most $2^{-\Omega(f)} \leq 2^{-\Omega((1-c)k/2)} = 2^{-\Omega(k)}$ fraction of graphs in $I(G)$. ■

The above lemma serves as a powerful tool for obtaining strong bounds for (s, t)-spanning set when G satisfies a simple restriction.

Theorem 20 *If $G = (V, E)$ has maximum degree at most cn, for $c < 1$, then*

- *any (s, t)-spanning set S for $W = I(G)$ has $t = \Omega(\frac{|E|}{\log^2 s})$.*
- *any spanning set S for $W = I(G)$ has $s = t = \Omega(|E|)$.*

Proof: Take any $G' \in I(G)$. Since it has $|E|$ edges and is the union of at most t graphs in S, some graph $H \in S$ has at least $|E|/t$ edges, and thus it touches at least $\sqrt{|E|/t}$ vertices. By Lemma 31, such a graph H is useful for at most $2^{-\Omega(\sqrt{|E|/t})}$ fraction of graphs in $I(G)$. Deleting all these graphs from $I(G)$ and repeating the process we get $H' \in S$ that is different from H and is useful for at most $2^{-\Omega(\sqrt{|E|/t})}$ fraction of graphs in $I(G)$. Continuing this way, $|S| = s \geq 2^{\Omega(\sqrt{|E|/t})}$, i.e. $t = \Omega(\frac{|E|}{\log^2 s})$. For a smallest spanning set, one way to get some results is to use the above bound with $s = t$. Assuming $|E| = \Omega(n^\epsilon)$, we get $s = t = \Omega(\frac{|E|}{\log^2 n})$. However, this bounds seems non-optimal, as in the proof we did not count all "small" graphs in S, and this is too wasteful when $s = t$. In fact, we will show that $s = t = \Omega(|E|)$. Let t_e be the number of graphs in some spanning set S for W which have e edges, so $t = \sum_{e=1}^{m} t_e$. Pick a random $G' \in I(G)$. For any H having e edges, since it touches at least $\sqrt{e}$ vertices and using Lemma 31, we have $\Pr(H \text{ is useful for } G') = 2^{-\Omega(\sqrt{e})}$, so $E(\text{number of edges of } G' \text{ that } H \text{ covers}) = e \cdot \Pr(H \text{ is useful for } G') = O(e2^{-\Omega(\sqrt{e})}) = O(1)$. Summing over all H in our spanning set and counting covered edges with repetitions, $E(\text{number of edges of } G' \text{ covered by all } H) = \sum_H E(\text{number of edges of } G' \text{ that } H \text{ covers}) = \sum_k t_k O(1) = O(\sum_k t_k) = O(t)$. On the other hand, the above expectation has to be at least $|E|$, because if it is less than $|E|$, there is some $G' \in I(G)$, not all of whose $|E|$ edges are covered by some t sets in our spanning set, which is a contradiction. Hence $|E| \geq O(t)$, so $t = \Omega(|E|)$. ■

A useful corollary is as follows.

Corollary 32 *The minimum (s,t)-spanning set for the collection of 2^n clique graphs has $t = \Theta(\frac{n^2}{\log^2 s})$.*

Proof: The lower bound comes from Theorem 20 by applying it to an $\frac{n}{2}$-clique while the upper bound follows from Corollary 25. ■

We are now ready to combine Theorem 19 and Theorem 20 in order to obtain our main result.

Theorem 21 *For any monotone property P,*

- $\mathcal{NT}^1_{\text{AND},s}(P) = \Omega(\frac{N^1(P)}{\log^2 s})$ *and* $\mathcal{NT}^0_{\text{OR},s}(P) = \Omega(\frac{N^0(P)}{\log^2 s})$.
  Thus, $\mathcal{NT}^1_{\text{AND},s}(P) = \mathcal{NT}^0_{\text{OR},s}(P) = \Omega(n^2/\log^2 s)$ *when* $N^1(P) = N^0(P) = \Omega(n^2)$.

- *If there is a min-1-certificate $C_1 \in \mathcal{C}^1$ with maximum degree cn and $\Omega(n^2)$ edges, then $\mathcal{T}^{obl}_{\text{AND}}(P) = \Omega(n^2)$. Similarly, if there is a min-0-certificate $C_0 \in \mathcal{C}^0$ s.t. $\overline{C_0}$ is of maximum degree cn and has $\Omega(n^2)$ edges, then $\mathcal{T}^{obl}_{\text{AND}}(P) = \Omega(n^2)$.*

Proof: Let $N^1(P) = r(n)$. Take any X of size $n/2$ and let C_1 be the min-1-certificate for P on X with the largest number of edges (equal $r(n/2) = \Omega(r(n)) = \Omega(N^1(P)))$[1].

[1] We assume that P is "regular", so above holds.

Since G has maximum degree at most $n/2$ (it is a certificate on a subset size $n/2$), we can apply Theorem 20 and conclude that any (s,t)-spanning set for $I(G) \subseteq W_{\text{AND}}$ has $t = \Omega(\frac{N^1(P)}{\log^2 s})$. The result follows by Theorem 19. The oblivious result is an immediate corollary of Theorems 19 and 20. ■

As an example, on the non-deterministic front, we get $\mathcal{NT}^0_{\text{OR},s}(P) = \Omega(\frac{n^2}{\log^2 s})$ for such P as CONNECTIVITY, NON-BIPARTITE, NON-EMPTY and NOT-FOREST. On the side of oblivious bounds, we can apply it to $P =$ CONNECTIVITY, since a (missing) complete bipartite graph on $n/2$ vertices has maximum degree $n/2$ and $n^2/4$ edges and hence $\mathcal{T}^{obl}_{\text{OR}}(P) = \Omega(n^2)$. As $\mathcal{T}_{\text{OR}}(P) = \Theta(n \log n)$, this is an example of limitation of obliviousness for the OR family. An identical result also holds for BIPARTITE.

We conclude by showing how to apply Theorem 19 to get an optomal bound on $\mathcal{T}^{obl}_{\text{OR},s}(P)$ for P being NO-ISOLATED-VERTEX.

Lemma 33 *For P=*NO-ISOLATED-VERTEX*, $\mathcal{T}^{obl}_{\text{OR},s}(P) = \Theta(\frac{n^2}{\log s})$, while $\mathcal{T}_{\text{OR},s}(P) = O(\frac{n^2}{\log^2 s} \operatorname{loglog} s)$.*

Proof: The oblivious $O(\frac{n^2}{\log s})$ upper bound follows from Corollary 25. To show the lower bound we first apply stabilization. We claim that there exists a subset X of size $\frac{n}{2}$ s.t. if we set all the edges outside of X to 1, we fix to 1 all the ORs whose edges touch $\Omega(\log s)$ vertices of V. This is a slightly stronger version of Lemma 6. If we pick X by picking every vertex of V with probability $1/2$, any H touching $k \geq 2\log s$ vertices will be stabilized with probability $1 - 1/s^2$, as only when all k vertices touched by H fall inside the query subset X, the OR defined by H is not going to be stabilized. Using the union bound and the fact that $|X| \geq n/2$ with probability at least $1/2$, the needed X exists. We work on X as our query set and apply Theorem 19 to X. $\mathcal{C}^0(X)$ is $\frac{n}{2}$ (missing) stars on X (each star connects $v \in X$ to $X \backslash \{v\}$). Any H in the spanning set S for $\mathcal{C}^0(X)$ must be a subgraph of a star on X (otherwise it is useless), and since H touches $O(\log s)$ vertices of a star graph, it must have $O(\log s)$ edges, i.e. $|H| = O(\log s)$. Since the union of all the stars in $\mathcal{C}^0(X)$ is $Edges_X$ and S covers all the graphs in $\mathcal{C}^0(X)$, the union of graphs in S covers $\binom{|X|}{2} = \Omega(n^2)$ edges of $Edges_X$. Since each graph in S has $O(\log s)$ edges, $|S| = t = \Omega(\frac{n^2}{\log s})$. ■

Bibliography

[1] M. Ajtai. A lower bound for finding predecessors in Yao's cell probe model. In *Combinatorica*, 8:235–247, 1988.

[2] P. Elias, R.A. Flower. The complexity of some simple retrieval problems. In *J. Assoc. Comput. Mach.*, 22:367–379, 1975.

[3] A. Hajnal, W. Maass and G. Turan. On the communication complexity of graph properties. In *Proc. 20th ASM Symp. on Theory of Computing (STOC)*, pp. 186–191, 1988.

[4] L. Hellerstein, P. Klein, R. Wilber. On the Time-Space Complexity of Reachability Queries for Preprocessed Graphs. In *Information Processing Letters*, 27:261–267, 1990.

[5] J. Kahn, M. Saks, D. Sturtevant. A topological approach to evasiveness. In *Proc. 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pp. 31–39, 1983.

[6] E. Kushilevitz, N. Nisan. Communication Complexity. *Cambridge University Press*, 1997.

[7] J. Van Lint. Introduction to Coding Theory. *Springer Verlag*, 1992.

[8] P.B. Miltersen. The bit probe complexity measure revisited. In *Proc. 10th Symp. on Theoretical Aspects of Computer Science (STACS)*, pp. 662–671, 1993.

[9] P.B. Miltersen. Lower bounds for union-split-find related problems on random access machines. In *Proc. 26th ASM Symp. on Theory of Computing (STOC)*, pp. 625–634, 1994.

[10] P.B. Miltersen. On cell probe complexity of polynomial evaluation. In *Theoretical Computer Science*, 143:167–174, 1995.

[11] P. Miltersen, N. Nisan, S. Safra, A. Wigderson. On Data Structures and Asymmetric Communication Comlexity. In In *Proc. 27th ASM Symp. on Theory of Computing (STOC)*, pp. 103–111, 1995.

[12] N. Nisan, M. Szegedy. On the Degree of Boolean Functions as Real Polynomials. In *Computational Complexity*, 4:301–313, 1994.

[13] R. Rivest, J. Vuillemin. On recognizing graph properties from adjecency matrices. In *Theoretical Computer Science*, 3:371–384, 1976.

[14] B. Xiao. New bounds in cell probe model. Ph.D. thesis, UC San Diego, 1992.

[15] A.C. Yao. Should tables be sorted. In *J. Assoc. Comput. Mach.*, 28:615–628, 1981.

[16] A.C. Yao. Some complexity questions related to distributed computing. In *Proc. 11th ASM Symp. on Theory of Computing (STOC)*, pp. 209–213, 1979.

[17] A.C. Yao. Monotone bipartite graph properties are evasive. In *SIAM Journal on Computing*, 17(3):517–520, 1988.