Deterministic Amplification of Space Bounded Randomized Algorithms

A thesis submitted in fulfillment
of requirements for the degree of
Master of Science in Computer Science

by
Ziv Bar-Yossef

supervised by
Professor Avi Wigderson

Department of Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel

December 20, 1999

Contents

Preface

This thesis was prepared in fulfillment of the requirements for the degree of Master of Science in the Computer Science department of the Hebrew University in Jerusalem.

I worked under the guidance of Professor Avi Wigderson. I am grateful to Avi for his support, care and ideas throughout the year we worked together. I have gained much knowledge from his difficult courses, from the discussions in his office and in the Espresso Bar in the Math building and especially from the masses of papers he asked me to read.

I would like to thank Oded Goldreich (whom I have not met yet...) for the mutual idea with Avi, that lead to this thesis and to the paper we wrote together ([BYGW98]). His comments and thoughts about the paper organization were of great importance.

I also warmly thank the people I talked to about this thesis and about my previous work: Noam Nisan, Nati Linial, Michael Ben-Or, Ran Raz, David Barrington, Pierre McKenzie, Eli Ben-Sasson, Ronen Shaltiel, Amir Shpilka, Roy Armoni, Dorit Aharonov, Dana Pe'er, Ido Bergman and Avner Magen.

Finally, I wish to thank Shuki Sagiv, Yaron Kanza, Yakov Kogan and Shmulik London for the great pleasure in working with them during the recent year and a half.

Ziv Bar-Yossef

Jerusalem, Israel
November 1998

Abstract

We consider deterministic amplification, that is reducing the error probability of a randomized algorithm while adding a minimal number of random bits. We outline the connection of deterministic amplification to extractors and survey known methods of deterministic amplification for polynomial time randomized algorithms.

Based on [BYGW98], we initiate the study of deterministic amplification of space bounded randomized algorithms that use a one-way random tape. The straightforward implementations of known amplification methods cannot be used for such algorithms, since they consume too much space. We present a new implementation of the AKS ([AKS87, CW89, IZ89]) method, that enables to amplify an S space algorithm that uses r random bits and errs with probability ϵ to an $O(kS)$ space algorithm that uses $r + O(k)$ random bits and errs with probability $\epsilon^{\Omega(k)}$.

This method can be used to reduce the error probability of BPL algorithms below any constant, with only a constant addition of new random bits. This is weaker than the exponential reduction that can be achieved for BPP algorithms by methods that use only $O(r)$ random bits. However, we prove that any black box amplification method that uses $O(r)$ random bits and makes at most p parallel simulations reduces the error to at most $\epsilon^{O(p)}$. Hence, in BPL, where p should be a constant, the error cannot be reduced to less than a constant. This means that our method is optimal with respect to black box amplification methods, that use $O(r)$ random bits.

The new implementation is based on first explicit constructions of constant space *online extractors* and *online expanders*. These are extractors and expanders, for which neighborhoods can be computed in a constant space by a Turing machine with a one-way input tape.

Chapter 1

Introduction

1.1 Deterministic Amplification

A *probabilistic algorithm* for a language $L \subseteq \{0,1\}^*$ is an algorithm that may use random bits during its execution and determines in the end whether the input belongs to the language or not. We consider *Monte-Carlo* algorithms, which are probabilistic algorithms that are allowed to err on their input with some constant probability $0 < \epsilon < \frac{1}{2}$.

A natural question that emerges when dealing with Monte-Carlo algorithms is how to reduce their error probability (to *amplify* the success of the algorithm). A naive approach would suggest to run the algorithm k times independently and take the majority vote of the results. This would yield a Monte-Carlo algorithm that errs with probability $\epsilon^{\Omega(k)}$. However, the number of random bits the new algorithm uses is k times the number of random bits used originally.

More involved methods are required for achieving a similar descent in the error probability without "paying" in so many random bits. These methods enable *deterministic amplification* of the probabilistic algorithm, that is reducing the algorithm's error probability while adding only a small number of new random bits.

All amplification methods are based on a *black box simulation*. The general framework of these methods can be outlined as follows. Denote by A the original Monte-Carlo algorithm, by ϵ its error probability and by r the number of random bits it uses. An $(l, r, k, \delta, \epsilon)$ *black box amplifier* is a meta algorithm, to which A is plugged in as a subroutine. The amplifier runs A k times on its input x and outputs the majority vote of the results.[1] The "heart" of the amplifier is the mechanism in which it produces the k r-bit strings, that are supplied as random bits for the k executions of A. The amplifier uses a special combinatorial object, called *extractor*, which is a function that receives a random seed of length l and produces k r-bit pseudo-random strings. These strings "fool" the copies of A, such that the error probability drops to $\delta < \epsilon$.

1.2 BPP Amplification

BPP is the class of languages that are recognized by polynomial time Monte-Carlo algorithms. For this class there exist very good amplification methods, that achieve dramatic (up to exponential) reductions in the error probability with a small addition of random bits.

Karp, Pippenger and Sipser ([KPS85]) and Chor and Goldreich ([CG89]) exhibited two methods that use an optimal number of random bits (r and $2r$ respectively) and reduce the error probability linearly with the number of simulations k. Impagliazzo and Zuckerman ([IZ89]) succeeded to achieve an exponential reduction in the error probability ($2^{-\Omega(k)}$), but they require the use of additional k^2 random bits. If we insist on using an optimal number of random bits ($O(r)$), this results in only a subexponential amplification of

[1] For some applications it is better to perform a different function on the k results. Refer to [Gol97] for examples.

$2^{-\Omega(\sqrt{r})}$. Nisan ([Nis92]) improved on this by presenting an exponential amplification in the cost of a $\log k$ factor in the number of random bits.

Cohen and Wigderson ([CW89]) and Impagliazzo and Zuckerman ([IZ89]) independently noticed that a result of Ajtai, Komlós and Szemerédi ([AKS87]) can be used to form an amplifier that reduces the error probability exponentially to $2^{-\Omega(k)}$ while using only $r + O(k)$ random bits. Hence, this method is optimal up to a constant for BPP amplifications.

The construction of AKS is based on *expander graphs*. These are graphs with a very high connectivity. Random walks on such graphs rapidly yield almost uniform distributions on the nodes. Since there are expanders with a constant degree, they are useful for generating almost uniform distributions using a small number of random bits.

Given a d-regular expander G on 2^r nodes, the AKS amplifier produces a random walk $v_1, \ldots v_k$ of length k, that starts at a random node $v \in G$. $v_1, \ldots, v_k$ are used as random strings for the k simulations of A. The number of random bits used is r (for the starting point) $+ \, k \log d$ (for determining the walk), which is $r + O(k)$, while the error probability drops exponentially.

1.3 Amplification of Space Bounded Computations

The randomness of a probabilistic algorithm may be thought as the ability to flip an unbiased coin at each stage of its execution. Frequently it is more convenient to think of the results of these coin flips as given to the algorithm in advance on a special *random tape*. Note that in the first characterization the algorithm may not recall previous random bits unless it explicitly saves them on its work tape, where in the second one it has a multiple access to the random bits without "paying" for additional space. Therefore, the two characterizations are equivalent only when the algorithm's space is unbounded (e.g. in BPP). In space bounded classes we may use the second characterization only if we restrict the random tape to be *one way*.

This thesis, based on [BYGW98], is the first to study deterministic amplification of space bounded Monte-Carlo algorithms, that have a read once access to their random tape. In particular we consider amplification for the class BPL, which consists of the languages recognized by logspace polynomial time Monte-Carlo algorithms with a one way random tape.

For amplifying such algorithms we need amplifiers that can compute the k pseudo-random strings and use them in the simulations within the same space limits of the original algorithm. All the mentioned above amplifiers, except for the naive one, require multiple access to their random seed during the computation. Therefore, a straightforward implementation of any of these amplifiers should store the seed on the work tape. However, the length of this seed may be exponential in the space limits of the algorithm.

Thus, a different approach is needed in order to utilize the above amplifiers for a space bounded amplification. We look for a non-trivial implementation of one of these amplifiers that computes the pseudo-random strings without storing the random bits of the seed in the work space. Our first main result succeeds to do so for the AKS amplifier.[2] The result requires the use of special expanders, for which neighborhoods can be computed in a small space on a Turing machine with a read-once access to its input tape. Our main lemma (lemma 7.2.1) proves that the expander of Margulis ([Mar73]) has this property. Computing a neighbor at this expander requires making a few summations modulo a power of 2. If we choose a proper encoding for the nodes, the summations can be carried out with a single pass on the input, while using only a constant space. Hence, in this expander one can compute the i^{th} bit of all the nodes in a k length random walk from the $(i-1)^{st}$ bit in these nodes, using a constant space per node. Therefore, we should keep only $O(k)$ bits from stage to stage. The amplifier executes simultaneously k copies of the original algorithm. At each stage k random bits are generated and supplied to the k copies. This yields a Monte-Carlo algorithm that reduces the error probability to $\epsilon^{\Omega(k)}$, adds $O(k)$ random bits and uses k times the space used by the original

[2] We also found a new implementation for the KPS amplifier, that enables to use it for space bounded amplifications. However, this implementation has no advantage on AKS, and reduces the error only linearly with k.

algorithm.

By choosing an arbitrarily large (but still constant) k we obtain the next upper bound:

Theorem (Upper Bound) *For any constant $0 < \delta < \epsilon$, there exists an (explicitly given) $(r+O(k), r, k, \delta, \epsilon)$ amplifier for BPL.*

The technical result that implies this theorem is a construction of a small space *online extractor*, as described in the next section.

The deterministic amplification we achieve for space bounded classes is not as good as the known amplification methods for time bounded classes. Our amplifier succeeds to achieve only a constant reduction of the error probability, since the amount of space it uses grows linearly with the number of simulations k. This connection is a direct result of our strategy: we make the simulations in parallel and not sequentially, as was done in previous methods.

A natural question is whether a non-constant reduction can be achieved also for space bounded computations, while keeping the number of random bits small (e.g. $O(r)$). For being able to obtain such a reduction with a black box amplifier, one has to present a method that makes at most a constant number of parallel simulations at any given moment. Our main lower bound proves this is not possible.

We prove that if a black box amplifier, that uses $O(r)$ random bits, makes at most p parallel simulations, then the error probability drops to at most $\epsilon^{O(p)}$. Since in BPL amplifications we can make only a constant number of parallel simulations we obtain the following lower bound:

Theorem (Lower Bound) *For any $(O(r), r, k, \delta, \epsilon)$ black box amplifier, which can be used for BPL amplification, $\delta \geq \epsilon^{O(1)}$.*

Again, the technical result that implies this theorem is a lower bound for the computational complexity of small space online extractors, as described in the next section.

This result shows that our implementation of the AKS amplifier is optimal with respect to black box amplifiers for BPL, that use $O(r)$ random bits.

1.4 Online Extractors and Online Expanders

The notion of *extractors* ([NZ96]) is closely related to deterministic amplification. $(l, r, k, \delta, \alpha)$ extractors are bipartite graphs with 2^l nodes on the left side, each with degree k, and 2^r nodes on the right side, such that for any distribution on the left that is uniform on some subset of size $\delta 2^l$, the induced distribution on the right (where the neighbor index is chosen uniformly at random) is α-close to uniform (where the distance is the statistical distance between distributions, which is half of the L_1 distance).

Extractors have many applications in theoretical computer science. The most celebrated ones are simulation of randomized algorithms using defective random sources, oblivious sampling, proofs of hardness results and conversion of probabilistic existence proofs to explicit constructions. Refer to [Nis96] for a survey about extractors.

Our main interest is in the connection of extractors to deterministic amplification. It turns out that extractors and black box amplifiers are almost equivalent. Every $(l, r, k, \delta, \epsilon)$ extractor can be utilized to form an $(l, r, k, \delta, \frac{1}{2} - \epsilon)$ amplifier. Furthermore, the complexity of computing neighborhoods in the extractor determines the efficiency of the resulting amplifier and its applicability to concrete classes of randomized algorithms. For example, in order to use an extractor in the amplification of polynomial time randomized algorithms, neighborhoods should be computed in a polynomial time.

We are interested in amplifiers that are applicable for space bounded algorithms. Hence, we seek for extractors, in which neighborhoods can be computed with a small space. Extractor neighborhoods are computed on a Turing machine with a one way input tape and k one way output tapes. When the machine is given a node on the left of the extractor, it outputs its k neighbors, one neighbor per output tape. An output tape is called *alive* at time t, if there is some data that was already written on it, and there is still more to be written until the execution ends. The *parallelism* of the machine is the maximal number (over all inputs) of living output tapes at any given moment. An (S, p) explicit *online extractor* is one that has an S space p-parallel neighborhood computation machine. A black box amplifier can run such a machine, by supplying it input from its one way random tape and using its output bits as random bits for the k simulations. If the original algorithm A uses S_A space, then the amplifier is an $O(S + pS_A)$ space algorithm.

Therefore the next construction yields the upper bound for amplifiers stated in the previous section:

Theorem (Upper Bound) *For any $0 < \epsilon < \frac{1}{16}$ there exists an (explicitly given) $(O(k), k)$ explicit online $(r + O(k), r, k, \epsilon^{\Omega(k)}, \frac{1}{2} - \epsilon)$ extractor.*

It turns out that the other direction is also true: black box amplifiers can be used to form online extractors. Moreover, if the amplifier is p-parallel and uses S space, the resulting extractor is (S, p) explicit. The following lower bound shows that in small space online extractors δ goes down with p and not with k. This implies the lower bound for amplifiers stated in the previous section.

Theorem (Lower Bound) *For all k and any $(o(r), p)$ explicit online $(O(r), r, k, \delta, \epsilon)$ extractor it holds that $\delta \geq 2^{-O(p)}$.*

The main tool in the above construction of the online extractor is showing that the expander of Margulis is a constant space *online expander*. Online expanders are ones, for which neighborhoods can be computed by a Turing machine with a one way input tape. This is the first known example of such expanders:

Theorem *For all $N = w^{2r}$, where w and r are natural numbers, there exists a constant space online expander on N vertices.*

1.5 Thesis Overview

Chapter 2, *Preliminaries*, reminds a few basic notions and facts from finite probability theory and introduces two combinatorial objects that have an essential role in the amplification methods discussed later in the thesis: *expander graphs* and *universal hash functions*. After presenting their definitions and their basic properties, the chapter outlines useful constructions and more advanced properties, such as the *expander mixing lemma* (section 2.2.5), the *leftover hash lemma* (section 2.3.4) and the *hash mixing lemma* (section 2.3.5).

Chapter 3, *Randomized Algorithms*, discusses randomized algorithms. It describes different kinds of randomization and presents the common randomized complexity classes.

Chapter 4, *Amplification of Randomized Algorithms*, introduces the problem of *deterministic amplification*. It sets the framework of our discussion by defining *black box amplifiers*. It presents the notions of *extractors*, *dispersers* and *weak extractors* and outlines their connection to deterministic amplification. It continues by proving a disperser lower bound, that shows exponential amplification is the best possible. It specifies the conditions in which an amplifier may be considered applicable for amplifications of concrete randomized classes, and it characterizes BPP and BPL applicable amplifiers. It proves the *equivalence theorem* (theorem 4.6.1), that exhibits the equivalence between amplifier efficiency and extractor explicitness. It ends by discussing amplification of one sided error randomized algorithms.

Chapter 5, *Amplification Methods*, describes in detail the main amplification methods discovered in the last 13 years.

Chapter 6, *Two Way Random Tape Amplification*, enhances the discussion framework to two way random tape randomized algorithms. It generalizes the black box amplifier notion to support amplification of two way algorithms. It defines a new efficiency measure for amplifiers and a new explicitness measure for extractors. Finally, it presents Nisan's result ([Nis93]) which shows that amplifiers with a two way random tape can reduce the error probability of BPL algorithms to 0.

Chapter 7, *Amplification of Space Bounded Algorithms*, presents the new results of the thesis. It discusses the problematics of applying the known amplifiers to space bounded classes (such as BPL). Then, it presents our new method to use the AKS and the KPS amplifiers for such classes. The upper bound is based on lemma 7.2.1 and lemma 7.2.2, that prove the expander of Margulis and its constant powers are constant space online expanders. The chapter ends by showing these methods are optimal with respect to black box amplifiers, that use a small number of random bits.

Chapter 2

Preliminaries

This chapter outlines briefly the two main tools that are used by deterministic amplification methods: expander graphs and universal families of hash function. It begins by introducing basic notions and facts from finite probability theory, that are used throughout this paper.

2.1 Finite Probability Theory

This paper makes use of finite probability theory. Thus, all the following definitions and propositions refer to finite sets.

Definition 2.1.1 *A* **distribution** *on a set S is a function $D : S \to [0, 1]$, such that $\sum_{x \in S} D(x) = 1$. The pair (S, D) is called a* **probability space**. *Any subset $T \subseteq S$ is called an* **event**. *The* **probability** *of T under D is: $Pr[T] = D(T) = \sum_{x \in T} D(x)$.*

Definition 2.1.2 *The* **uniform distribution** *on a set S (denoted by U_S) is a distribution on S that satisfies $U_S(x) = \frac{1}{|S|}$ for all $x \in S$.*

We will use the notation $x \in_D S$ to denote a random choice of an element x from the set S according to the distribution D. $x \in_R S$ denotes a choice from S according to the uniform distribution.

Definition 2.1.3 *A* **random variable** *X on a set T is a function $X : S \to T$, where (S, D) is a probability space. For any value $\alpha \in T$, the probability of X to get the value α is $Pr[X = \alpha] = D(X^{-1}(\alpha))$.*

Definition 2.1.4 *The* **expectation** *of a random variable X on a set T is $E[X] = \sum_{\alpha \in T} \alpha Pr[X = \alpha]$.*

A basic but useful property of the expectation operator is that it is linear. That is, $E[aX + bY] = aE[X] + bE[Y]$ for any two random variables X, Y and any two constants a, b.

An *indicator random variable* is a random variable on the set $\{0, 1\}$. It has the special property that its expectation equals to its probability to get 1.

Definition 2.1.5 *The* **variance** *of a random variable X on a set T is $Var[X] = E[(X - E[X])^2] = E[X^2] - E[X]^2$.*

Definition 2.1.6 *Let A and B be two events in a probability space (S, D). The* **conditional probability** *of A given B is $Pr[A \mid B] = \frac{Pr[A \cap B]}{Pr[B]}$.*

Definition 2.1.7 *Two events A, B are called* **independent** *if $Pr[A \mid B] = Pr[A]$.*

It is easy to verify that A and B are independent if and only if $Pr[A \cap B] = Pr[A] \cdot Pr[B]$. The following is a basic fact that uses the conditional probability notion:

Lemma 2.1.1 (Complete Probability Formula) *Let A and $B_1, \ldots, B_k$ be events in the same probability space, such that $B_1, \ldots, B_k$ are mutually disjoint and cover the whole space. Then $Pr[A] = \sum_{i=1}^{k} Pr[A \cap B_i] = \sum_{i=1}^{k} Pr[A \mid B_i] Pr[B_i]$.*

Definition 2.1.8 *Two random variables X, Y on sets T_X and T_Y respectively are called* **independent** *if for all $\alpha \in T_X$ and $\beta \in T_Y$ $Pr[X = \alpha \mid Y = \beta] = Pr[X = \alpha]$.*

The variance operator is not linear in the general case. However, for two independent random variables X, Y it is additive: $Var[X + Y] = Var[X] + Var[Y]$. This fact is useful for various applications.

The following are three extremely useful inequalities regarding random variables behavior. Their proofs are fairly simple and can be found in any elementary textbook about probability theory.

Theorem 2.1.1 (Markov Inequality) *For any non-negative random variable X and for any constant $\epsilon > 0$ it holds that:*

$$Pr[X > \epsilon] < \frac{E[X]}{\epsilon}$$

Theorem 2.1.2 (Chebyshev Inequality) *Every random variable X satisfies the following:*

$$Pr[|X - E[X]| > \epsilon] < \frac{Var[X]}{\epsilon^2}$$

Theorem 2.1.3 (Chernoff Bounds) *Let $X_1, \ldots, X_k$ be independent identically distributed indicator random variables and let X be their sum. Then for any $\epsilon > 0$:*

$$Pr[X > (1 + \epsilon)E[X]] < e^{\frac{-\epsilon^2 E[X]}{6}}$$

Definition 2.1.9 *Given two distributions D_1 and D_2 on a set S, the* **statistical distance** *between them is:*

$$d(D_1, D_2) = \frac{1}{2}||D_1 - D_2||_1 = \frac{1}{2} \sum_{x \in S} |D_1(x) - D_2(x)| = \max_{T \subseteq S} |D_1(T) - D_2(T)|$$

Definition 2.1.10 *A distribution D_1 is* **α-close** *to a distribution D_2, if $d(D_1, D_2) \leq \alpha$.*

Definition 2.1.11 *Given two distributions D_1 on the set S_1 and D_2 on the set S_2, their* **independent Cartesian product distribution** *(denoted by $D_1 \times D_2$) is a distribution on the set $S_1 \times S_2$, where for each $(s_1, s_2) \in S_1 \times S_2 : D_1 \times D_2(s_1, s_2) = D_1(s_1) \cdot D_2(s_2)$.*

Proposition 2.1.1 *Let D_1, D_2 be two distributions on a set S and let D_3 be a distribution on a set T. Then, $d(D_1 \times D_3, D_2 \times D_3) \leq d(D_1, D_2)$.*

Proof: Take an arbitrary subset $A \subseteq S \times T$. Denote by A_S its projection on S. It holds that:

$|D_1 \times D_3(A) - D_2 \times D_3(A)| = |\sum_{a \in A} D_1 \times D_3(a) - \sum_{a \in A} D_2 \times D_3(a)| = |\sum_{a=(s,t) \in A} D_1(s) \cdot D_3(t) - \sum_{a=(s,t) \in A} D_2(s) \cdot D_3(t)| = |\sum_{(s,t) \in A} (D_1(s) - D_2(s)) D_3(t)| \leq |\sum_{s \in A_S} D_1(s) - D_2(s)| = |D_1(A_S) - D_2(A_S)|$.

Therefore, $d(D_1 \times D_3, D_2 \times D_3) = max_{A \subseteq S \times T} |D_1 \times D_3(A) - D_2 \times D_3(A)| \leq max_{A \subseteq S \times T} |D_1(A_S) - D_2(A_S)| \leq max_{B \subseteq S} |D_1(B) - D_2(B)| = d(D_1, D_2)$. □

Given a distribution D on a set S, any function $f : S \to T$ defines a distribution on T (f is actually a random variable). We will denote this distribution by D^f.

Proposition 2.1.2 *Let D_1 and D_2 two distributions on a set S and let $f : S \to T$ be some function on this set. Then, $d(D_1^f, D_2^f) \leq d(D_1, D_2)$.*

Proof: Take an arbitrary subset $B \subseteq T$. $|D_1^f(B) - D_2^f(B)| = |\sum_{b \in B} D_1^f(b) - D_2^f(b)| =$
$|\sum_{b \in B} \sum_{a \in f^{-1}(b)} D_1(a) - D_2(a)| = |\sum_{a \in f^{-1}(B)} D_1(a) - D_2(a)| = |D_1(f^{-1}(B)) - D_2(f^{-1}(B))|$.

Similar to the proof of the previous proposition, this means that $d(D_1^f, D_2^f) \leq d(D_1, D_2)$. □

2.2 Expander Graphs

Expander graphs are graphs with a very high connectivity. Each sufficiently small set of nodes has a neighborhood of at least the same size. A random walk on this kind of graphs forms a rapidly mixing Markov chain, i.e. the distribution on the nodes converges very fast to the uniform distribution. Moreover, there exist expander graphs with a bounded degree. This means that one can produce an almost uniform distribution on the nodes of the expander quite easily, without having to use a substantial amount of random bits. This fact is extremely useful in numerous applications.

2.2.1 Definitions

Definition 2.2.1 *An undirected multi-graph $G = (V, E)$ is called a* **$\boldsymbol{\gamma}$-expander** *if for every subset $S \subseteq V$ of size at most $\frac{|V|}{2}$, the size of its neighbor set $\Gamma(S)$ is at least $(1 + \gamma(1 - \frac{2|S|}{|V|}))|S|$.*

Graphs with a substantial amount of edges (e.g. the complete graph) can easily satisfy the expansion property, and thus are expanders. However, such graphs are wasteful in resources, and consequently are not competent for many applications. We will be interested in sparse expanders with a bounded degree d. A probabilistic argument proves the existence of such expanders ([Pin73]). In the sequel we will exhibit an explicit construction of such an expander.

Verifying whether a given graph is a γ-expander is coNP-complete ([BKV$^+$81]): one has to check all the subsets of V of size at most $\frac{|V|}{2}$. Fortunately, there is another (partial) characterization of expander graphs that uses algebraic graph theory and enables to efficiently estimate the expansion factor (γ) of a given graph.

Consider the adjacency matrix M of a d-regular undirected graph G on N vertices. Since G is undirected M is an integer symmetric matrix. Therefore, it has N real eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. The following are basic facts from algebraic graph theory:

Proposition 2.2.1
1. $\lambda_1 = d$.
2. G is connected if and only if $\lambda_2 < d$.

The second fact demonstrates the connection between graph connectivity and the ratio between the first and second eigenvalues of its adjacency matrix. One may relate expansion as some generalization of connectivity. It turns out that indeed the separation between the two first eigenvalues determines the expansion factor of the graph. Alon proved this important connection in [Alo86a]:

Theorem 2.2.1 (Alon) *A d-regular multi-graph G is a γ-expander with: $\gamma \geq 1 - (\frac{|\lambda_2|}{d})^2$.*

Therefore, the smaller the ratio between the first and the second eigenvalues of the graph, the higher its expansion factor.

Computing the eigenvalues of a real matrix is easy. Therefore, this result enables us to present lower bounds for the expansion factor of given graphs.

Following this new characterization we present a new (weaker) definition of expander graphs:

Definition 2.2.2 *An undirected multi-graph G on N vertices is called an* $(\boldsymbol{N}, \boldsymbol{d}, \boldsymbol{\lambda})$ **expander** *if it is d-regular and the absolute value of the second eigenvalue of its adjacency matrix is λ.*

We should remark that this definition is indeed weaker than the original one. Alon proved in [Alo86b] a lower bound of $\Omega(\sqrt{d})$ on $|\lambda_2|$. There is no parallel upper bound on the expansion factor of graphs.

Many applications of expander graphs require the use of super-polynomial sized expanders. A polynomial time algorithm cannot use such expanders, if it tries to compute the whole expander. If, however, neighborhoods in the expander can be computed efficiently, the algorithm can calculate on the fly only the nodes it needs. This concept motivates the following definition:

Definition 2.2.3 *Let $\mathcal{C}$ be some complexity class. A graph G is called* $\boldsymbol{\mathcal{C}}$**-explicit***, if there are encodings for the vertices and for the indices of neighbors in G and there exists an algorithm $R \in \mathcal{C}$ that on input (j, v), where j is a neighbor index and v is a node in G (given in the above encodings), computes the j^{th} neighbor of v (in the same encoding).*

Usually we will be interested in polytime-explicit expanders.

2.2.2 The Expander of Margulis

In this section we present an explicit construction of a polytime-explicit bounded degree expander graph. This construction is due to Margulis ([Mar73]), but is better known as the expander of Gabber and Galil, who proved its expansion properties ([GG81]).

Margulis introduced a family of graphs $\{G_N\}$ for integers N that have an integer square root m. Let T_1 and T_2 be 2×2 matrices over the ring Z_m and let b_1 and b_2 be 2-dimensional vectors over Z_m. We define two affine transformations: $A_1 z = T_1 z + b_1$ and $A_2 z = T_2 z + b_2$. The vertex set of G_N is $Z_m \times Z_m$. Each node is a pair (x, y) where $x, y \in Z_m$. (x, y) is connected to $T_1\binom{x}{y}, A_1\binom{x}{y}, T_2\binom{x}{y}$ and $A_2\binom{x}{y}$.

G_N is an 8-regular graph. Margulis proved that its second eigenvalue is less than 8, hinting it may be a good expander. However, he did not achieve a bound on the ratio $\frac{\lambda}{d}$. Gabber and Galil ([GG81]) were the first to present such a bound:

Theorem 2.2.2 (Gabber-Galil) *G_N with $T_1 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, b_1 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, T_2 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}$ and $b_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ is an $\frac{2-\sqrt{3}}{4}$-expander graph.*

Jimbo and Maruoka ([JM87]) obtained the best estimation known for the second eigenvalue of this graph:

Theorem 2.2.3 (Jimbo-Maruoka) *G_N with the above affine transformations is an $(N, 8, 5\sqrt{2})$ expander graph.*

Notice that the simple description of the expander of Margulis enables to compute its neighborhoods in a linear time and a logarithmic space. All one needs is to calculate a few summations modulo m.

2.2.3 Increasing the Expansion Factor

Some applications require highly expanding graphs. We presented a construction that has a relatively moderate expansion factor. In the sequel we show how such a construction can be used to form a new expander with a much higher expansion factor.

Definition 2.2.4 *Given a graph G whose adjacency matrix is M, define the* $\boldsymbol{q^{th}}$ **power of G** *(denoted by $G^{(q)}$) to be the multi-graph whose adjacency matrix is M^q.*

Proposition 2.2.2 *Let G be an (N, d, λ) expander. Then $G^{(q)}$ is an (N, d^q, λ^q) expander.*

Proof: $G^{(q)}$ is clearly a multi-graph on N vertices, since the dimensions of M^q are exactly the same as the dimensions of M.

α is an eigenvalue of M if and only if α^q is an eigenvalue of M^q. Therefore, if λ is the second eigenvalue of M then λ^q is the second eigenvalue of M^q.

Since G is d-regular, the sum of each row and each column in M is d. Hence, the sum of each row and each column in M^q is d^q. This means that $G^{(q)}$ is d^q-regular. □

The ratio between the first and the second eigenvalues of $G^{(q)}$ is exponentially smaller than this ratio at G. Recall (theorem 2.2.1) that this results in an expander with a possibly higher expansion factor. Therefore, raising an expander to a constant power enables both to increase its expansion and to keep its degree bounded:

Corollary 2.2.1 *For every constant $\alpha > 0$ and $N = m^2$ (where m is a natural number) there exists an (N, d, λ) expander, for which d is constant and $\frac{\lambda}{d} < \alpha$.*

2.2.4 Random Walks on Expanders

A *random walk* on an undirected graph G is a stochastic process, in which one traverses nodes of the graph, by following the next rules:

1. It chooses one of the nodes as a starting point according to some initial distribution.
2. If it visits node v at step t, it chooses one of its neighbors $u \in \Gamma(v)$ uniformly at random, and visits u at the next step.

For each step $t = 0, 1, 2, \ldots$ the random walk defines a distribution $q^{(t)}$ on the nodes of the graph, where $q_v^{(t)}$ is the probability the process visits node v at step t.

If the graph has certain properties[1], then for any initial distribution $q^{(0)}$ on the nodes, the sequence of distributions converges to some stationary distribution π. If G is d-regular, π is simply the uniform distribution.

One of the most useful properties of expander graphs is that the rate of convergence of random walks on them to the stationary distribution is very fast. If G is an (N, d, λ) expander, then the distance between $q^{(t)}$ and π is proportional to $(\frac{\lambda}{d})^t$.

This fact is used to form an almost uniform distribution on $\{0, 1\}^r$ as follows. Pick the graph of Margulis G_{2^r} and fix a tolerance parameter α, that denotes the allowed deviation from the uniform distribution. Choose the minimal t such that $(\frac{\lambda}{d})^t \leq \alpha$. Now choose a starting point $v \in G_{2^r}$ uniformly at random, and start a random walk of length t from v. From now on any continuation of the random walk forms a distribution which is α-close to the uniform one. Note that the number of random bits needed for getting k

[1] The graph should be connected and non-bipartite.

values from this almost uniform distribution is $r + (t + k)\log d$, which is $r + O(\log \frac{1}{\alpha}) + O(k)$. On the other hand, the number of random bits needed to produce k values of a real uniform distribution is kr.

This example shows how expanders can be utilized to form distributions which are much more efficient in the random bits consumption, but maintain the main properties of the original distributions.

2.2.5 Expander Mixing Lemma

In this section we present a property of expander graphs, that is useful in analyzing their performance at various applications.

Let $G = (V, E)$ be an (N, d, λ) expander graph, and let A and B be two arbitrary subsets of V. Denote by $E(A, B)$ the number of edges between the nodes of A and the nodes of B: $E(A, B) = |\{(u, v) \in E : u \in A, v \in B\}|$. If we choose A and B at random from V the expected value of $E(A, B)$ is $\frac{d|A||B|}{N}$. The next lemma, called the *expander mixing lemma*, shows that in expander graphs the value of $E(A, B)$ is close to the expectation for any two subsets $A, B \subseteq V$.

Lemma 2.2.1 *Let $G = (V, E)$ be an (N, d, λ) expander and let A, B be two subsets of V. Then, $|E(A, B) - \frac{d|A|B|}{N}| \leq \lambda\sqrt{|A||B|} \leq \lambda N$.*

Proof: Let M be the adjacency matrix of G. Let χ_A and χ_B be the characteristic vectors of A and B respectively (these are N-length binary vectors that have 1 on their i^{th} entry iff the i^{th} node in G belongs to A or B respectively). Notice that $E(A, B) = \chi_A \cdot M \cdot \chi_B$.

Let $\lambda_1, \ldots, \lambda_N$ be the N eigenvalues of M in a descending order of absolute values and let $v_1, \ldots, v_N$ be a corresponding basis of orthogonal eigenvectors. Without loss of generality we may choose v_1 to be $(1, 1, \ldots, 1)$, which is an eigenvector of M that corresponds to the first eigenvalue d.

We express χ_B as a linear combination of the basis vectors: $\chi_B = \sum_{i=1}^{N} \alpha_i v_i$. Since the vectors $\{v_i\}$ are orthogonal, then $\chi_B \cdot v_1 = \alpha_1 \|v_1\|_2^2$. However, $\chi_B \cdot v_1 = |B|$ and $\|v_1\|_2^2 = N$. Therefore, $\alpha_1 = \frac{|B|}{N}$.

It follows that $M \cdot \chi_B = M \sum_{i=1}^{N} \alpha_i v_i = \frac{d|B|}{N} v_1 + \sum_{i=2}^{N} \alpha_i \lambda_i v_i$. Hence, $|\chi_A \cdot M \cdot \chi_B - \chi_A \frac{d|B|}{N} v_1| = \|\chi_A \cdot \sum_{i=2}^{N} \alpha_i \lambda_i v_i\|_2$.

Notice that $\chi_A \cdot v_1 = |A|$. Therefore the left hand side of the last equality is exactly $|E(A, B) - \frac{d|A||B|}{N}|$.

By the triangle inequality it follows that $\|\sum_{i=2}^{N} \alpha_i \lambda_i v_i\|_2 \leq \sum_{i=2}^{N} \|\alpha_i \lambda_i v_i\|_2$. Since $|\lambda_i| \leq \lambda$ for every $i = 2, \ldots, n$, then the last sum is $\leq \lambda \sum_{i=2}^{N} \|\alpha_i v_i\|_2$. The fact that $v_2, \ldots, v_N$ are orthogonal implies that this equals to $\lambda \|\sum_{i=2}^{N} \alpha_i v_i\|_2 \leq \lambda \|\chi_B\|_2 = \lambda\sqrt{|B|}$.

We can conclude that $|E(A, B) - \frac{d|A||B|}{N}| \leq \|\chi_A\|_2 \lambda \sqrt{|B|} = \lambda\sqrt{|A||B|}$. □

2.3 Pairwise Independence and Universal Hashing

The production of pairwise independent random variables requires less random bits than the production of totally independent random variables. However, they are completely adequate for many applications, that ordinarily would require a totally uniform distribution. Hence, pairwise independence is another major tool in saving random bits.

The notion of *universal families of hash functions*, which was introduced by Carter and Wegman in [CW79], exhibits an elegant method to produce sequences of pairwise independent random variables.

2.3.1 Pairwise Independence

A random variable is uniform on a set T, if for any $\alpha \in T$ $Pr[X = \alpha] = \frac{1}{|T|}$. Pairwise independence is some generalization of this property:

Definition 2.3.1 *Consider a set of random variables $\{X_i\}_{i \in U}$ that are uniform on a set T. We call these variables* **pairwise independent** *if for any $i \neq j \in U$ and any $\alpha, \beta \in T$ $Pr[X_i = \alpha \wedge X_j = \beta] = \frac{1}{|T|^2}$.*

This definition can be generalized to k-wise independence in the obvious way. The variables are considered to be totally independent if they are k-wise independent for any $k = 1, 2, \ldots, |U|$.

Note that pairwise independence does not imply total independence. Consider, for example, a set of three random variables X, Y, Z on the set $\{0, 1\}$, whose probability space is (S, U_S), where $S = \{0, 1\}^2$. The functions of X, Y, Z are specified by the following table:

s	X	Y	Z
00	0	0	0
01	0	1	1
10	1	0	1
11	1	1	0

It is easy to verify that X, Y, Z are pairwise independent, but since $Z = X \oplus Y$ they are not 3-wise independent.

2.3.2 Universal Hashing

Carter and Wegman presented in [CW79] the notion of universal hashing. Their original goal was to find new ways of hashing. However, many other applications in a realm of areas were found in subsequent years.

Definition 2.3.2 (Carter-Wegman) *A family of functions $H = \{h_s : U \to T \mid s \in S\}$ is called a* **2-universal family of hash functions** *if for any $i \neq j \in U$ and any $\alpha, \beta \in T$ $Pr_{s \in_R S}[h_s(i) = \alpha \wedge h_s(j) = \beta] = \frac{1}{|T|^2}$.*

This definition can be easily generalized to k-universal families of hash functions.

The similarity to the definition of pairwise independent random variables is not accidental, as shown in the next proposition:

Proposition 2.3.1 *Given a 2-universal family of hash functions $H = \{h : U \to T\}$ the set of random variables $\{h(i)\}_{i \in U}$ where $h \in_R H$ is pairwise independent.*

Proof: First we show that each random variable $h(i)$ distributes uniformly on T. Fix any $i \in U$ and any $\alpha \in T$. Take some $j \neq i$ in T. By the complete probability formula we obtain:

$$Pr_{h \in_R H}[h(i) = \alpha] = \sum_{\beta \in T} Pr_{h \in_R H}[h(i) = \alpha \wedge h(j) = \beta] = \sum_{\beta \in T} \frac{1}{|T|^2} = \frac{1}{|T|}$$

The equality before the last one follows from the fact H is a 2-universal hash family.

The independence property follows directly from definition 2.3.2. □

The opposite direction is also true. Assume we are given a set of pairwise independent random variables $\{X_i\}_{i \in U}$ on a set T that distribute on the probability space S. We can describe these random variables in a table, whose rows correspond to the elements of S and its columns correspond to the elements of U. The entry (s, i) in the table contains $X_i(s)$. The rows of this table define a family of hash functions $H = \{h : U \to T\}$. It is easy to check that this is a 2-universal family of hash functions.

2.3.3 A Construction of Universal Hash Functions

We observed that a 2-universal family of hash functions may assist us in producing pairwise independent random variables. But is any construction good for our needs? We will wish to seek for a construction that satisfies the two following requirements:

1. *Succinctness*

   Each function in the family can be represented by a small number of bits.

2. *Computational Efficiency*

   On input (s, i) there is an efficient algorithm (e.g. in logspace) that computes $h_s(i)$.

A family that has these two properties is very useful. For producing a set of $k \leq |U|$ pairwise independent random variables we have to choose at random only one function h from the family and compute its value on k fixed arbitrary elements of U. The succinctness ensures us that we use a small number of random bits, and the efficiency enables us to make the computations efficiently (assuming that k is only a polynomial).

Carter and Wegman exhibited in [CW79] a construction of a 2-universal family of hash functions from $\{0,1\}^r$ to $\{0,1\}^r$, that satisfies the succinctness and the efficiency requirements, as presented in the next theorem:

Theorem 2.3.1 (Carter-Wegman) *Consider the field $GF(2^r)$ of all polynomials of degree $< r$ over the field $GF(2)$. Let H be the set of all linear functions over this field: $H = \{h_{a,b} \mid h_{a,b}(x) = ax + b,\ a, b \in GF(2^r)\}$. Then, H is a 2-universal family of hash functions from $\{0,1\}^r$ to $\{0,1\}^r$, such that each $h \in H$ can be represented by $2r$ bits and computed in logspace.*

Proof: Each string in $\{0,1\}^r$ can be thought as the sequence of coefficients of a polynomial over $GF(2)$ of degree $< r$. Therefore, there is a 1-1 correspondence between the elements of $GF(2^r)$ and the strings in $\{0,1\}^r$. It follows that H is actually a collection of functions from $\{0,1\}^r$ to $\{0,1\}^r$.

Fix arbitrary $i \neq j \in GF(2^r)$ and $\alpha, \beta \in GF(2^r)$. We should prove that $Pr_{a,b \in_R GF(2^r)}[h_{a,b}(i) = \alpha \wedge h_{a,b}(j) = \beta] = \frac{1}{2^{2r}}$.

Consider the following system of two linear equations over the field $GF(2^r)$ on the variables a and b:

$$\begin{aligned} ai + b &= \alpha \\ aj + b &= \beta \end{aligned}$$

The two equations are linearly independent, since $i \neq j$. Therefore, there is a single solution to this system. It follows that only the function $h_{a,b}$ the corresponds to this solution satisfies $h_{a,b}(i) = \alpha \wedge h_{a,b}(j) = \beta$. There are 2^{2r} functions in H, and each one of them is chosen with the same probability. Hence, indeed $Pr_{a,b \in_R GF(2^r)}[h_{a,b}(i) = \alpha \wedge h_{a,b}(j) = \beta] = \frac{1}{2^{2r}}$.

Since each function in H is defined by two elements from the field $GF(2^r)$, $2r$ bits are sufficient for their representation. Computing the value of each such function on an element of the field requires performance of linear operations modulo some irreducible polynomial. This can be done in logspace. $\square$

This construction can be used also for functions from $\{0,1\}^r$ to $\{0,1\}^l$ for $l < r$, by taking the first l bits of the linear functions outputs.

2.3.4 Leftover Hash Lemma

This section presents a lemma due to Impagliazzo, Levin and Luby ([ILL89]), called the *Leftover Hash Lemma*, that shows how a universal family of hash functions can be used to form an almost uniform distribution with a small number of random bits. This lemma has some applications, as we will see in section 5.4.

Lemma 2.3.1 (Leftover Hash Lemma) *Let H be a universal family of hash functions from U to T. Then, for any $A \subseteq U$, the distribution of the random variable $(h, h(x))$, where $x \in_R A$ and $h \in_R H$, is α-close to uniform, where $\alpha = \frac{1}{2}\sqrt{\frac{|T|}{|A|}}$.*

Proof: Consider a distribution D on a set S. Define the *collision probability* (denoted by $c(D)$), to be the probability of two elements independently chosen according to D to be the same: $c(D) = Pr_{i,j \in_D S}[i = j]$.

Claim 2.3.1.1 *Let D be a distribution on a set S, for which $c(D) \leq \frac{1+4\alpha^2}{|S|}$. Then, D is α-close to uniform.*

Proof:

Denote by U_S the uniform distribution on S. Recall that the statistical distance between distributions is half their L_1 distance. Therefore $d(D, U_S) = \frac{1}{2}\|D - U_S\|_1$. By the Cauchy-Schwartz inequality $\|D - U_S\|_1^2 \leq |S|\|D - U_S\|_2^2 = |S|(\|D\|_2^2 - 2(D, U_S) + \|U_S\|_2^2)$.

Notice the next three facts:

1. $c(D) = Pr_{i,j \in_D S}[i = j] = \sum_{s \in S} Pr_{i,j \in_D S}[i = s \wedge j = s] = \sum_{s \in S} Pr_{i \in_D S}[i = s] \cdot Pr_{j \in_D S}[j = s] = \sum_{s \in S} D(s) \cdot D(s) = \|D\|_2^2$.

2. $\|U_S\|_2^2 = \sum_{s \in S} \frac{1}{|S|^2} = \frac{1}{|S|}$.

3. $(D, U_S) = \sum_{s \in S} D(s)\frac{1}{|S|} = \frac{1}{|S|}\sum_{s \in S} D(s) = \frac{1}{|S|}$.

Therefore, $d(D, U_S)^2 \leq \frac{1}{4}|S|(c(D) - \frac{2}{|S|} + \frac{1}{|S|}) \leq \frac{1}{4}|S|(\frac{1+4\alpha^2}{|S|} - \frac{1}{|S|}) = \alpha^2$. Hence, $d(D, U_S) \leq \alpha$. □

Denote by D the distribution of the random variable $(h, h(x))$, where $x \in_R A$ and $h \in_R H$.

Claim 2.3.1.2 $c(D) \leq \frac{1+\frac{|T|}{|A|}}{|H \times T|}$.

Proof:

$$\begin{aligned}
c(D) &= Pr_{x_1,x_2 \in_R A, h_1,h_2 \in_R H}[(h_1, h_1(x_1)) = (h_2, h_2(x_2))] \\
&= Pr_{x_1,x_2 \in_R A, h_1,h_2 \in_R H}[h_1(x_1) = h_2(x_2) \mid h_1 = h_2] \cdot Pr_{h_1,h_2 \in_R H}[h_1 = h_2] \\
&= Pr_{x_1,x_2 \in_R A, h \in_R H}[h(x_1) = h(x_2)] \cdot \frac{1}{|H|} \\
&= (Pr_{x_1,x_2 \in_R A}[x_1 = x_2] + Pr_{x_1,x_2 \in_R A}[x_1 \neq x_2] \cdot Pr_{h \in_R H, x_1,x_2 \in_R A}[h(x_1) = h(x_2) \mid x_1 \neq x_2]) \cdot \frac{1}{|H|} \\
&\leq (\frac{1}{|A|} + \frac{1}{|T|})\frac{1}{|H|} \\
&= \frac{1}{|H||T|}(1 + \frac{|T|}{|A|}) \\
&= \frac{1 + \frac{|T|}{|A|}}{|H \times T|}
\end{aligned}$$

□

Using claim 2.3.1.1, while $S = H \times T$ and $\alpha = \frac{1}{2}\sqrt{\frac{|T|}{|A|}}$, we obtain what we need to prove in the lemma. □

2.3.5 Hash Mixing Lemma

We end the chapter by presenting an interesting and useful property of universal hash families, called *Hash Mixing*, that was discovered by Nisan in [Nis92]. Consider a universal family of hash functions from U to T and two arbitrary subsets $A \subseteq U, B \subseteq T$. Nisan's lemma states that for nearly all the functions h in the family, $h(x)$ distributes almost uniformly on B when x distributes uniformly on A.

In the sequel we use the notation $\mu(S)$ to denote the *proportional volume* of a subset $S \subseteq T$, i.e. $\mu(S) = \frac{|S|}{|T|}$.

Definition 2.3.3 (Nisan) *Let $A \subseteq U, B \subseteq T$, $h : U \to T$ and $\epsilon > 0$. h is called* $(\boldsymbol{\epsilon}, \boldsymbol{A}, \boldsymbol{B})$**-good** *if:*

$$|Pr_{x \in_R U}[x \in A \wedge h(x) \in B] - \mu(A)\mu(B)| \leq \epsilon$$

Lemma 2.3.2 (Nisan's Hash Mixing Lemma) *Let H be a 2-universal family of hash functions from U to T. Then for all $A \subseteq U, B \subseteq T$ and for any $\epsilon > 0$:*

$$Pr_{h \in_R H}[h \textit{ is not } (\epsilon, A, B) - \textit{good}] \leq \frac{\mu(A)\mu(B)(1 - \mu(B))}{|U|\epsilon^2}$$

Proof: Fix arbitrary $A \subseteq U, B \subseteq T$ and $\epsilon > 0$. We are to prove that:

$$Pr_{h \in_R H}[|Pr_{x \in_R U}[x \in A \wedge h(x) \in B] - \mu(A)\mu(B)| > \epsilon] \leq \frac{\mu(A)\mu(B)(1 - \mu(B))}{|U|\epsilon^2} \quad (2.1)$$

Define the random variable $Z = Pr[x \in A \wedge h(x) \in B]$. 2.1 is obtained by a simple application of Chebyshev inequality with the variable Z. For doing that we should note the following:

1. $E_{h \in_R H}[Z] = E_{h \in_R H}[Pr_{x \in_R U}[x \in A \wedge h(x) \in B]] = \mu(A) E_{h \in_R H}[Pr_{x \in_R A}[h(x) \in B]] = \mu(A)\mu(B)$. The equality before the last follows from the conditional probability definition. The last equality follows from the universality of H.

2. $E_{h \in_R H}[Z^2] = E_{h \in_R H}[Pr_{x_1, x_2 \in_R U}[x_1, x_2 \in A \wedge h(x_1), h(x_2) \in B] = \mu(A)^2 E_{h \in_R H}[Pr_{x_1, x_2 \in_R A}[h(x_1), h(x_2) \in B]$. The last equality followed from the conditional probability definition. Using the complete probability formula we obtain: $\mu(A)^2 E_{h \in_R H}[Pr_{x_1, x_2 \in_R A}[h(x_1), h(x_2) \in B \mid x_1 = x_2]\frac{1}{|A|} + Pr_{x_1, x_2 \in_R A}[h(x_1), h(x_2) \in B \mid x_1 \neq x_2](1 - \frac{1}{|A|})] = \frac{\mu(A)}{|U|} E_{h \in_R H}[Pr_{x \in_R A}[h(x) \in B]] + \frac{\mu(A)}{|U|}(|A| - 1) E_{h \in_R H}[Pr_{x_1, x_2 \in_R A}[h(x_1) \in B \wedge h(x_2) \in B \mid x_1 \neq x_2]]$. Using the universality of H we have: $\frac{\mu(A)}{|U|}\mu(B) + \frac{\mu(A)}{|U|}(|A| - 1)\mu(B)^2 = \frac{\mu(A)\mu(B)}{|U|}(1 + \mu(B)|A| - \mu(B))$.

3. $Var[Z] = E[Z^2] - E[Z]^2 = \frac{\mu(A)\mu(B)}{|U|}(1 + \mu(B)|A| - \mu(B)) - \mu(A)^2\mu(B)^2 = \frac{\mu(A)\mu(B)(1 - \mu(B))}{|U|}$.

□

Chapter 3

Randomized Algorithms

3.1 Definitions

A *randomized Turing machine* is one that may generate a random bit at each stage of its execution and use it in the computation. This bit distributes uniformly at random, that is its probability to 0 and 1 is $\frac{1}{2}$. Random bits that are generated at different steps are independent from one another. One may relate these random bits as a sequence of independent flips of an unbiased coin.

Unlike the deterministic counterpart, on a given input x a randomized Turing machine M may have several possible computations. Any fixing of the sequence of random bits M uses yields a new computation. Therefore, we may relate M as a deterministic machine that receives two inputs: the regular input x on the input tape and the random input y on the *random tape*. We denote the computation of M on the input x and the random string y by $M(x, y)$.

We concentrate on randomized Turing machines that compute languages. These machines produce a one-bit answer on any given input x. The interpretation of this bit should be as follows: if it is 1, then x belongs to the language and if it is 0, then it does not.

However, a randomized machine is not guaranteed to output the correct answer on all its possible computations on a given input. We denote by W_x the set of random strings, on which the machine is correct, i.e. the set of all y's for which $M(x, y) = f_L(x)$, where f_L is the characteristic function of the language M computes. We call this set the *set of good witnesses for* x. Similarly, $\overline{W_x}$ is the *set of bad witnesses for* x, that is all the strings y for which $M(x, y) \neq f_L(x)$. We require that for all x $|W_x| > |\overline{W_x}|$.

Since the random strings are taken from a uniform distribution, the last requirement means that on any input the chances of the randomized machine to be correct are higher than its chances to be incorrect.

3.2 Types of Randomized Algorithms

We differ between several types of randomized algorithms, according to the error probability they allow:

1. *Zero Error:* The machine always gives the correct answer. However, such machines may run for a long time with some positive probability. These algorithms are also called *Las-Vegas algorithms*.

2. *One-Sided Error:* The machine may err only on inputs that belong to the language. The error probability is strictly less than ϵ, where $0 < \epsilon < 1$ is some constant.

3. *Bounded Two-Sided Error:* The machine may err on any input with probability, which is strictly less than ϵ, where $0 < \epsilon < \frac{1}{2}$ is some constant.

4. *Unbounded Error:* The machine may err on any input with probability strictly less than $\frac{1}{2}$.

Our discussion concentrates on *Monte Carlo* algorithms, as defined next:

Definition 3.2.1 *An* (r, ϵ) **Monte-Carlo algorithm** *for a language* L *(where* $0 < \epsilon < \frac{1}{2}$ *is some constant) is a randomized algorithm* M*, that on inputs of length* n *uses* $r = r(n)$ *random bits, and satisfies for all inputs* x*:* $\frac{|W_x|}{2^r} < \epsilon$.

3.3 Classes of Randomized Algorithms

Randomized Turing machines consume resources (time, space, etc.) like regular Turing machines. We measure this consumption in terms of the input length. Therefore, given a complexity class of languages accepted by deterministic machines with some limitations on the resources they consume, we can define corresponding classes of languages that are accepted by randomized Turing machines which have the same limitations on the resources. The classification of randomized algorithms according to the error probability imposes 4 randomized classes that correspond to any deterministic class C:

1. ZPC - Languages accepted by zero error randomized machines.
2. RC - Languages accepted by one-sided error randomized machines.
3. BPC - Languages accepted by bounded two-sided error randomized machines.
4. PC - Languages accepted by unbounded error randomized machines.

The symmetry in the attitude of zero error, bounded two sided error and unbounded error algorithms to inputs that belong to the language and to inputs that do not implies that the classes ZPC, BPC and PC are closed under complement. However, this is not likely to be true for RC. The simple observation that $RC \cap coRC = ZPC$ implies that if RC is closed under complement then $RC = ZPC$.

The most famous examples of randomized classes are ZPP, RP, BPP and PP, which are the classes of randomized algorithms that run in a polynomial time and ZPL, RL and BPL, which are the classes of randomized algorithms that run in a logarithmic space and a polynomial time. Note that as opposed to ZPP, RP and BPP, PP is not a realistic model for efficient randomized computation, as evidenced by the fact that $NP \subseteq PP$.

3.4 Access to The Random Tape

The original definition of randomized algorithms states that they may use a random bit at each stage of their execution. When analyzing such algorithms, it is frequently more convenient to think as if these bits are given to the algorithm in advance on the *random tape*. The difference between the two definitions arises with respect to space consumption. In the first definition the machine may not recall previous random bits, unless it explicitly stores them in its work space. Whereas, in the second definition the machine may move the reading head back and forth on the random tape, without being charged for extra space. The two definitions coincide only if we restrict the random tape to be *one way*: it can move only to the right.

When the randomized machine has a bounded space, a two way access to the random tape may increase its power. Hence for every space bounded randomized class R we define $2W - R$ to be the corresponding class with a two way random tape.

3.5 Applications and Examples

During the past two decades randomized algorithms became a major tool in solving computational problems. Randomization turns out to save time, space, communication or any other resource. There are numerous examples in which randomized algorithms do better than any known deterministic algorithm. Sometimes the randomized algorithm is simply simpler than its deterministic counterpart.

Below are some examples for celebrated randomized algorithms. Refer to [MR95] for a textbook about the subject.

One of the first randomized algorithms was Rabin's algorithm for testing primality ([Rab80]). His method uses Fermat's little theorem to verify with a high probability that a given natural number is a prime. The algorithm is always correct whenever the given number is prime and is almost always correct when it is composite. The algorithm runs in a polynomial time, hence it is a $coRP$ algorithm. There is also an RP algorithm for primality ([AH87]), hence the $PRIMES$ language is actually in ZPP. There is no known deterministic algorithm that solves the problem in a polynomial time.

Checking whether a given polynomial in a few variables is identically zero is an elementary problem, that has applications in many algorithms (e.g. in verifying whether a given graph has a perfect matching). A $coRP$ algorithm for the problem due to Schwartz ([Sch80]) is as follows: given a polynomial f on n variables, choose n random integers $a_1, \ldots, a_n$ in the range $0, \ldots, 2n$, and compute $f(a_1, \ldots, a_n)$. Reply that the polynomial is identically 0 iff the result of the computation is 0. There is no known P or even RP algorithm that solves the problem.

The undirected st-connectivity problem asks whether a given undirected graph with two special nodes s and t has a path between these two nodes. [AKL$^+$79] presented a randomized algorithm that solves the problem by simply performing a random walk of a bounded length on the graph. This algorithm runs in logspace and polynomial time. The algorithm is always correct when s and t are not connected and is most of the times correct when they are connected. Thus, this problem belongs to the class RL. The best known deterministic algorithm for solving this problem runs in polynomial time and $O((\log n)^{\frac{4}{3}})$ space ([ATSWZ97]).

Chapter 4

Amplification of Randomized Algorithms

Deterministic amplification is a mechanism that gets as input an (r, ϵ) Monte-Carlo algorithm for some language L (called the *original algorithm*) and outputs an (l, δ) Monte-Carlo algorithm for the same language (called the *amplified algorithm*), where $\delta < \epsilon$ and l is as small as possible.

We are interested in such mechanisms, that do not try to "understand" or to utilize the way the original algorithm works for accomplishing the amplification, but rather use it as a "black box" and depend only on the size of its witness set. If we find such a mechanism, we are assured it can be used for any Monte-Carlo algorithm, no matter how it works. This is the motivation for the *black box amplification* model, described below.

4.1 Black Box Amplification

4.1.1 Definition

A *black box amplifier* $M_?$ is a generic algorithm, into which any Monte-Carlo algorithm A^x can be plugged in, yielding an algorithm M_{A^x}. $M_?$ uses A^x as a subroutine and has no access to its internal configuration. It runs A^x k times and outputs the majority vote of the results. The input of $M_?$ is r, the number of random bits used by A^x. It generates k bit strings of length r, and supplies them as random bits for the simulations of A^x. For the generation $M_?$ uses $l = l(r)$ random bits.

Note that A^x already comes with its input. A^x can be thought as the computation of a Monte-Carlo algorithm A on the input x. To formalize the way A^x is utilized by $M_?$, we introduce the widely used notion of *configuration graph*.

A *configuration* of a Turing machine consists of the contents of its work tapes, the positions of its heads and its local state at a given moment of its execution. The *configuration graph* of A^x is a directed graph, whose nodes correspond to the configurations of A when it runs on the input x. Only configurations, in which A^x waits for a random bit or ends the execution are considered. There is a single node that corresponds to the *initial configuration* and several leaf nodes that correspond to *accepting* and *rejecting* configurations. Each internal node has 2 outgoing edges labeled by 0 and 1. Two configurations u and v are connected by an edge labeled by b, if A^x when being at u and reading the random bit b moves to v.

For simulating the computation of A^x, $M_?$ simply holds the current configuration u of A^x. When it wishes to submit it a random bit b, it calls A^x with u and b as input. A^x returns back v, the configuration connected to u by an edge labeled by b.

Definition 4.1.1 *An* $(\boldsymbol{l, r, k, \delta, \epsilon})$ **black box amplifier** *(where $k = k(r)$, $l = l(r)$ and $\delta = \delta(k)$) is an algorithm $M_?$ as specified above, such that for any (r, ϵ) Monte-Carlo algorithm A^x M_{A^x} is an (l, δ) Monte-Carlo algorithm.*

4.1.2 Possible Implementation

This section outlines an implementation of a black box amplifier. The details are somewhat cumbersome. However, they are essential for convincing the skeptical reader that the black box amplifier notion presented above is realistic and feasible.
$M_?$ is a multi tape randomized Turing machine with the following special constructs:

1. k *simulation tapes*.

2. An *address tape*.

3. A *majority tape*.

4. 7 special states: q_{start}, q_{sim0}, q_{sim1}, q_{accept}, q_{reject}, q_{random} and, q_{end}.

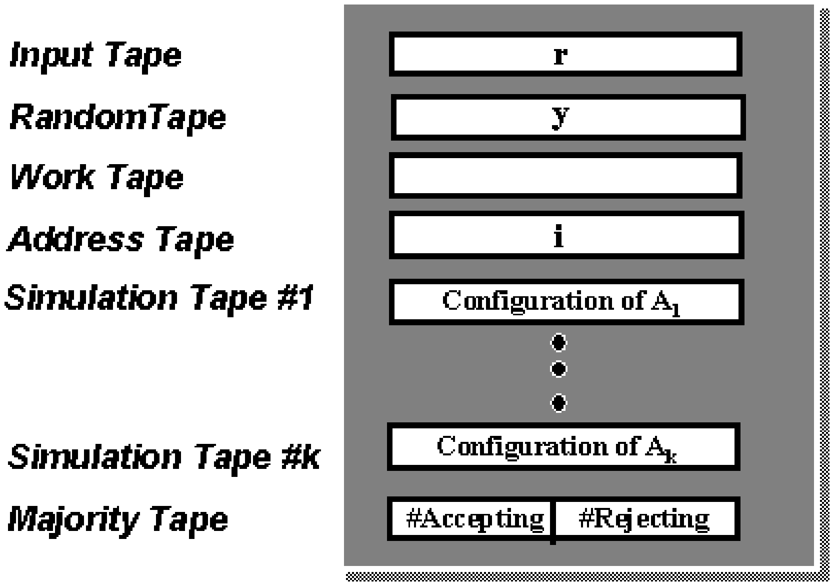

Figure 4.1: A Black Box Amplifier

$M_?$ gets r, the number of random bits used by A^x, on its input tape.

When $M_?$ moves to q_{start} from one of its "ordinary" states, it starts a simulation of A^x on the i^{th} simulation tape, where i is the index written on the address tape. $M_?$ initializes the simulation tape by a bit string that represents the initial configuration of A^x, and moves to q_{random}. This indicates that A_i^x waits for a random bit from $M_?$. From q_{random} $M_?$ can move to any of its "ordinary" states.

When $M_?$ wishes to submit a random bit to A_i^x, it writes i on the address tape, and moves either to q_{sim0} or to q_{sim1}, depending on the value of the submitted bit.

From q_{sim0} and q_{sim1} $M_?$ moves to one of the states q_{accept}, q_{reject} or q_{random}, according to the following rules: $M_?$ replaces the string on the i^{th} simulation tape, which represents some configuration u of A^x, by the configuration v to which A^x moves from u after reading the random bit indicated by the state. If v is an accepting configuration, $M_?$ enters q_{accept}. If it is a rejecting configuration, $M_?$ enters q_{reject}. Otherwise, it enters q_{random}.

$M_?$ processes the outputs of the simulations on its majority tape. It holds on this tape the current number of accepting simulations and the current number of rejecting simulations. When some simulation ends, $M_?$ increments one of these counters, depending on its state: if it is in q_{accept} it increments the acceptance counter, and if it is in q_{reject} it increments the rejection counter. Unless all the simulations ended (the sum of the two counters is k), $M_?$ moves from these states to q_{end}. q_{end} indicates to $M_?$ that the i^{th} simulation has ended. It can move from it to any of its ordinary states.

When all the simulations terminate $M_?$ stops with either acceptance or rejection, depending which of the two counters on the majority tape is greater.

The steps, at which $M_?$ is in one of the special states, except for q_{random} and q_{end}, are called *simulation steps*. The other steps are called *computation steps*. During the computation steps $M_?$ is not allowed to read or write on any of the simulation tapes or the majority tape. We conclude that the execution of $M_?$ cannot be affected by the choice of the algorithm A^x, because what it sees from every algorithm A^x we plug in is a sequence of r requests for random bits. Hence, fixing r and the random string y uniquely determines the computation steps of $M_?$.

According to our definition the amplifier always performs a majority vote on the results. For some applications it may be useful to perform other functions $f : \{0,1\}^k \rightarrow \{0,1\}$ on the k results. For example, if A is wrong only on inputs that belong to L, we could use the OR function instead. The amplifier definition can be easily extended to handle these cases too.

4.2 Extractors and Dispersers

4.2.1 Combinatorial Characterization of Amplifiers

Note that black box amplifiers utilize the original algorithm only through its output interface and not by analysis of the way it works. This observation results in a combinatorial characterization of black box amplifiers.

Definition 4.2.1 *An* $\boldsymbol{(l, r, k)}$ **bipartite graph** *is a bipartite graph, whose left side is* $V_1 = \{0,1\}^l$*, its right side is* $V_2 = \{0,1\}^r$ *and each node in* V_1 *is of degree* k*.*

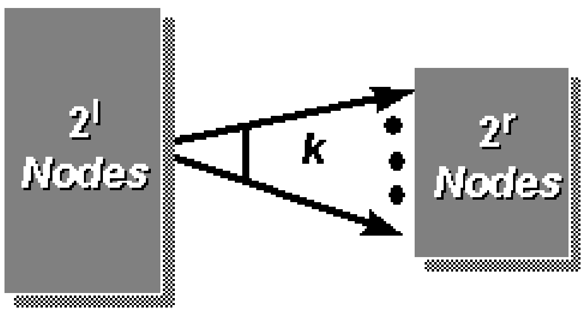

Figure 4.2: An (l, r, k) Bipartite Graph

Consider an $(l, r, k, \delta, \epsilon)$ amplifier $M_?$ and fix a number r and a random string y of length $l = l(r)$. The computation steps of $M_?$ are uniquely determined by r and y. It follows that the bit strings that are supplied to the k simulations as random bits are functions of r and y only. Denote these strings by $y_1, \ldots, y_k$. We define an (l, r, k) bipartite graph $E_{M,r}$ as follows: V_1 corresponds to all the possible random strings of length $l = l(r)$ generated by $M_?$, when its input is r. V_2 corresponds to all the possible random strings of length r generated by an r-bit Monte-Carlo algorithm. Each node $y \in V_1$ is connected to the nodes $y_1, \ldots, y_k \in V_2$.

The fact $M_?$ reduces an ϵ error probability to a δ error probability has a translation to a combinatorial property of $E_{M,r}$.

Definition 4.2.2 *An* $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{k}, \boldsymbol{\delta}, \boldsymbol{\epsilon})$ **weak extractor** *is an* (l, r, k) *bipartite graph, for which if* $B \subseteq V_1$, $|B| \geq \delta 2^l$, *and* $W \subseteq V_2$ *contains the majority of the neighbors of each* $y \in B$, *then* $|W| \geq \epsilon 2^r$.

Lemma 4.2.1 $M_?$ *is an* $(l, r, k, \delta, \epsilon)$ *amplifier if and only if for all r* $E_{M,r}$ *is an* $(l, r, k, \delta, \epsilon)$ *weak extractor.*

Proof: Let $M_?$ be an $(l, r, k, \delta, \epsilon)$ amplifier. Assume, to the contradiction, there is some r for which $E_{M,r}$ is not an $(l, r, k, \delta, \epsilon)$ weak extractor. Therefore, there exists some subset $B \subseteq V_1$ on the left of $E_{M,r}$, $|B| \geq \delta 2^l$, and some $W \subseteq V_2$, $|W| < \epsilon 2^r$, that contains the majority of the neighbors of each $y \in B$. W may be a bad witness set of some (r, ϵ) Monte-Carlo algorithm A^x. M_{A^x} is wrong when the random string it generates belongs to B. Hence, its error probability is at least its chances to hit a random string in B, which at least δ. This contradicts the fact that $M_?$ is supposed to reduce the error probability below δ.

For the other direction, assume that for all r $E_{M,r}$ is an $(l, r, k, \delta, \epsilon)$ weak extractor. Assume, to the contradiction, that $M_?$ is not an $(l, r, k, \delta, \epsilon)$ amplifier. Therefore, there exists some (r, ϵ) Monte-Carlo algorithm A^x, for which M_{A^x} errs with probability at least δ. Denote by $W \subseteq \{0,1\}^r$ the bad witness set of A^x. Since A^x errs with probability less than ϵ, then $|W| < \epsilon 2^r$. Denote by $B \subseteq \{0,1\}^l$ the set or random strings, on which M_{A^x} is wrong. B is exactly the set of nodes in V_1, whose majority of neighbors belong to W. Since the error probability of M_{A^x} is at least δ, then $|B| \geq \delta 2^l$. This is a contradiction to the assumption that $E_{M,r}$ is an $(l, r, k, \delta, \epsilon)$ weak extractor. $\square$

Figure 4.3 illustrates the new characterization of black box amplifiers. One may think of a black box amplifier as a program that chooses a random node on the left side of a weak extractor, computes its k neighbors on the right, and uses these neighbors as random strings for the k executions of the plugged algorithm.

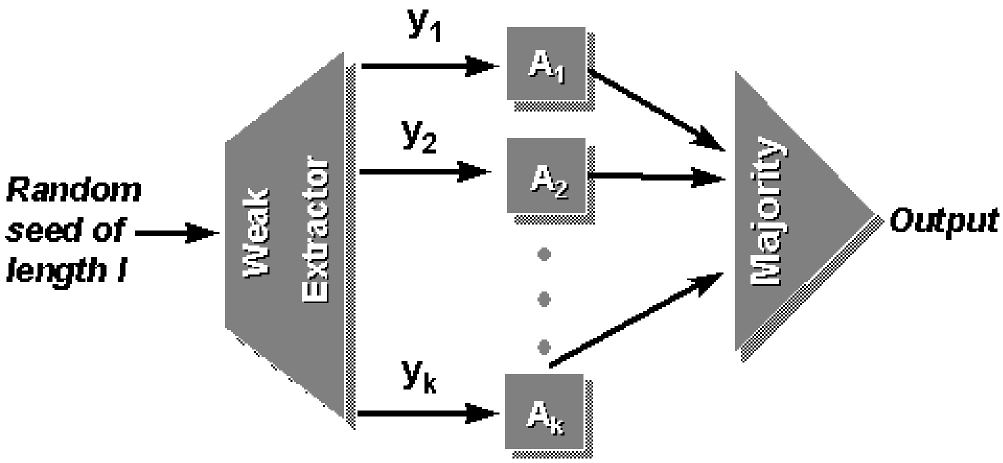

Figure 4.3: Combinatorial Characterization of Amplifiers

Lemma 4.2.1 is extremely useful in the study of deterministic amplification. One direction introduces a generic way for constructing black box amplifiers. The other direction shows that it is possible to study the limitations of amplifiers through an analysis of the combinatorial properties of weak extractors.

4.2.2 Applications and Basic Properties

Weak extractors were introduced by Cohen and Wigderson in [CW89] (they called them *MAJORITY dispersers*), and they lie between the two more familiar notions of *extractors* and *dispersers*.

Definition 4.2.3 *An* $(\boldsymbol{l, r, k, \delta, \alpha})$ **extractor** *is an* (l, r, k) *bipartite graph, such that for every subset* $B \subseteq V_1$*, where* $|B| \geq \delta 2^l$*, if we choose at random a node from* B *and then a random neighbor of this node, the induced distribution on the right is* α*-close to uniform.*

Definition 4.2.4 *An* $(\boldsymbol{l, r, k, \delta, \epsilon})$ **disperser** *is an* (l, r, k) *bipartite graph, such that for every subset* $B \subseteq V_1$*, where* $|B| \geq \delta 2^l$*, it holds that* $|\Gamma(B)| \geq \epsilon 2^r$*, where* $\Gamma(B) \subseteq V_2$ *is the neighbor set of* B*.*

We should note that these notations deviate from the standard definitions in a few technical non significant details.[1] Our choice best illustrates the connection to amplification exhibited in lemma 4.2.1.

Extractors are important combinatorial objects, that were introduced by Nisan and Zuckerman in [NZ96]. They strengthen the former notion of dispersers that were presented by Sipser in [Sip88]. An intensive research of these objects during the last decade exhibited their importance in various applications in theoretical computer science and their close relationship to deterministic amplification. Refer to [Nis96] for a survey about extractors and dispersers.

When dealing with $(l, r, k, \delta, \alpha)$ extractors, we usually fix l and r as parameters, where $l \geq r$, and try to find extractors that minimize δ, α and k. Extractors can be viewed as combinatorial tools that enable us to extract randomness from somewhat random distributions. Given a distribution on l bits, which is concentrated only on a δ- portion of the space, we can form (using the extractor) an almost uniform distribution on r bits while investing only $\log k$ new real random bits.

Extractors have numerous applications. The most celebrated ones are:

1. *Simulation of randomized algorithms using weak random sources.*

   As defined in the previous chapter, randomized algorithms expect to receive their random bits from a uniform distribution. However, such a distribution is not achievable in reality. Extractors enable us to use distributions which are far from uniform, and to form almost-uniform distributions which are sufficient for the needs of randomized algorithms.

2. *Oblivious Sampling.*

   Oblivious samplers are objects that enable to estimate the expectation of every function $f : S \to [0, 1]$, while using a small number of queries. The sampler supplies a list of random elements from the range S, and computes the average of the function of them. If the sampler is good, this average is close to the function's expectation with a high probability. Extractors enable the construction of good samplers that use a minimal number of random bits.

3. *Proofs of Hardness results.*

   Using extractors, Zuckerman showed in [Zuc93] that approximating the function $\log(Clique(G))$ for a given graph G within a constant is NP-hard.

4. *Transformation of probabilistic existence proofs to explicit constructions.*

   The probabilistic method (see [AES92] for a textbook) provides proofs for the existence of various combinatorial objects. However, very frequently these proofs do not yield explicit examples of the combinatorial objects. Wigderson and Zuckerman ([WZ93]) used an extractor to transform such a proof to an explicit construction of depth-2 superconcentrators.

[1] k is usually the logarithm of the left degree and δ is the min-entropy of the distribution on the left.

Dispersers is a slightly weaker notion than weak extractors, and weak extractors are slightly weaker than extractors, as evidenced by the next proposition:

Proposition 4.2.1
1. Every $(l, r, k, \delta, \alpha)$ extractor is an $(l, r, k, \delta, \frac{1}{2} - \alpha)$ weak extractor.
2. Every $(l, r, k, \delta, \epsilon)$ weak extractor is an $(l, r, k, \delta, \epsilon)$ disperser.

Proof:
1. Consider an arbitrary $(l, r, k, \delta, \alpha)$ extractor E. Assume, to the contradiction, it is not an $(l, r, k, \delta, \frac{1}{2} - \alpha)$ weak extractor. Hence, there exist a subset $B \subseteq V_1$, $|B| \geq \delta 2^l$ and a subset $W \subseteq V_2$, $|W| < (\frac{1}{2} - \alpha)2^r$, such that W contains the majority of the neighbors of every $y \in B$. Take the uniform distribution on B. The induced distribution on V_2 gives W a weight of at least $\frac{1}{2}$. However, the uniform distribution on V_2 gives W a weight of less than $\frac{1}{2} - \alpha$. Therefore, the distance between these two distributions is greater than α. This contradicts the assumption that E is an $(l, r, k, \delta, \alpha)$ extractor.

2. Consider some $(l, r, k, \delta, \epsilon)$ weak extractor E and let $B \subseteq V_1$ be some subset of size at least $\delta 2^l$. Define $W = \Gamma(B)$. Naturally W contains the majority (and even all) the neighbors of each $y \in B$. Since E is a weak extractor $|W| \geq \epsilon 2^r$, implying it is also a disperser. □

This proposition shows that extractors are sufficient for constructing black box amplifier (i.e. obtaining upper bounds), and analysis of dispersers suffices for proving amplification limitations (i.e. obtaining lower bounds).

4.3 Disperser Lower Bound

This section shows that exponential amplification is the best we can expect. It presents a lower bound obtained for dispersers by Radhakrishnan and Ta-Shma ([RTS97]) and uses the connection to amplifiers to conclude a parallel lower bound for them too.

Theorem 4.3.1 *Let E be an $(l, r, k, \delta, \epsilon)$ amplifier, where $\delta < 1$ and $k \leq \frac{\epsilon}{2} 2^r$. Then: $\delta \geq \epsilon^{O(k)}$.*

By lemma 4.2.1 and by proposition 4.2.1 it suffices to prove that for any $(l, r, k, \delta, \epsilon)$ disperser $\delta \geq \epsilon^{O(k)}$. For that, we need to show that in such a disperser there is no subset of the left side of size at least $\epsilon^{O(k)} 2^l$ whose neighbor set is of size less than $\epsilon 2^r$. This is equivalent to showing that the bipartite complement of the disperser does not contain a copy of the complete bipartite graph $K_{\epsilon^{O(k)} 2^l, (1-\epsilon)2^r}$.

The problem of determining the maximal number of edges in a bipartite graph that does not contain a subgraph isomorphic to $k_{a,b}$ is known as the *Zarankiewicz Problem*. An extensive study of this problem in extremal graph theory,[2] resulted in several solutions. Radhakrishnan and Ta-Shma use in [RTS97] a probabilistic argument due to Kővari, Sós and Turán to prove the next disperser lower bound:

Theorem 4.3.2 (Radhakrishnan - Ta-Shma) *Let E be an $(l, r, k, \delta, \epsilon)$ disperser with $\epsilon \leq \frac{1}{2}$, $\delta < 1$ and $k \leq \frac{\epsilon}{2} 2^r$. Then, $\delta > 2^{-O(k)}$.*

Proof: Let B be a random subset of V_2 of size $d = \lceil (1-\epsilon)2^r \rceil$. For any node $v \in V_1$ define the random indicator variable X_v as follows:

$$X_v = \begin{cases} 1 & \Gamma(v) \cap B = \phi \\ 0 & Otherwise \end{cases}$$

[2]Refer to [Bol78], pages 309-326 for details.

$\Gamma(v)$ denotes the neighborhood set of v. Since B is chosen randomly from V_2 it holds that $Pr[X_v = 1] = \frac{\binom{2^r-k}{d}}{\binom{2^r}{d}}$.

Denote by S the nodes in V_1 that do not have any neighbor in B. Since B is at least a $1-\epsilon$ fraction of V_2, it follows that $|\Gamma(S)| < \epsilon|V_2| = \epsilon 2^r$. The fact E is a disperser implies that $|S| < \delta 2^l$. Note that $|S|$ is exactly the sum $\sum_{v \in V_1} X_v$. The linearity of the expectation implies that $E[|S|] = \sum_{v \in V_1} E[X_v] = 2^l \frac{\binom{2^r-k}{d}}{\binom{2^r}{d}}$. We obtain:

$$2^l \frac{\binom{2^r-k}{d}}{\binom{2^r}{d}} < \delta 2^l$$

Note that $\frac{\binom{2^r-k}{d}}{\binom{2^r}{d}} = \frac{(2^r-k)!(2^r-d)!}{(2^r-k-d)!(2^r)!} = \frac{(2^r-k-d+1)\cdots(2^r-k)}{(2^r-d+1)\cdots 2^r} \geq (\frac{2^r-k-d+1}{2^r-d+1})^d$. Therefore:

$$\frac{1}{\delta} < (\frac{2^r-d+1}{2^r-k-d+1})^d = (1 + \frac{k}{2^r-k-d+1})^d \leq e^{\frac{kd}{2^r-k-d+1}}$$

The last inequality follows from the fact that $1 + x \leq e^x$ for all $x > 0$.

Now, since $d = \lceil (1-\epsilon)2^r \rceil \leq (1-\epsilon)2^r + 1$ and since $k \leq \frac{\epsilon 2^r}{2}$, it follows that $2^r - k - d + 1 \geq 2^r - \frac{\epsilon 2^r}{2} - (1-\epsilon)2^r = \frac{\epsilon 2^r}{2}$. Therefore, $\frac{d}{2^r-k-d+1} \leq \frac{(1-\epsilon)2^r+1}{\frac{\epsilon 2^r}{2}} < \frac{2(1-\epsilon)}{\epsilon} + 1$. We obtain:

$$\frac{1}{\delta} < e^{k(\frac{2(1-\epsilon)}{\epsilon}+1)} = 2^{O(k)}$$

Hence, $\delta > 2^{-O(k)}$. □

4.4 Amplifier Applicability

So far we measured the performance of a given amplifier only by the number of random bits it uses and the rate of error reduction it achieves. We ignored the computational resources needed to actually use this amplifier. These resources are crucial in deciding whether the amplifier can be used for amplification of specific randomized classes. The next definition determines when an amplifier is considered useful for our purpose:

Definition 4.4.1 *Let $\mathcal{C}$ be some randomized complexity class. An amplifier $M_?$ is called* **$\mathcal{C}$ applicable** *if for any algorithm $A^x \in \mathcal{C}$ it holds that $M_{A^x} \in \mathcal{C}$.*

The applicability of an amplifier is determined by its efficiency in resource consumption. We concentrate on the two most important resources: time and space.

We distinct between *private resources* and *simulation resources*. Private resources are ones used by the amplifier during its *computation steps* (recall this notion from section 4.1.2). These resources do not depend on the algorithm plugged into the amplifier. The simulation resources are the ones used during the simulation steps. They depend solely on the plugged algorithm.

The *private time* is the time needed to carry out the computation steps of the amplifier. The *simulation time* is the time spent for the invocations of the plugged subroutine. Since each such invocation requires a constant number of operations, the simulation time is proportional to the number of random bits demanded by the plugged algorithm. Without loss of generality, we assume the plugged algorithm demands a random bit at each step of its execution. Hence, if an amplifier $M_?$ uses T private time and a T_A time randomized algorithm A^x is plugged into it, then M_{A^x} runs in $O(T + kT_A)$ time.

The simulation storage consists of the space required for holding the configurations of the simulated algorithms. The rest of the space used by the amplifier is considered as private storage.

Note that if an algorithm A^x uses S_A space, then its configuration requires $O(S_A)$ bits. We charge the amplifier for these bits throughout the simulation of A^x. One may argue this has some redundancy, because A^x may not need all the space all the time. However, since the amplifier is supposed to work for any algorithm A^x we plug in, it should work in particular for an algorithm that uses all its space constantly. Therefore, our charge strategy is essential.

For estimating the simulation space used by a black box amplifier we introduce the *parallelism* notion, which is the maximal number of *active* simulations (simulations which already begun and are not done yet) it has.

Definition 4.4.2 *A simulation made by a black box amplifier $M_?$ is called* **active** *at time t, if $M_?$ holds an intermediate (not accepting or rejecting) configuration of this simulation at time t. The* **parallelism** *of $M_?$ is the maximal number (over all inputs and all time steps) of active simulations it has.*

If the private storage of a black box amplifier $M_?$ is S, it is p-parallel and the space used by A^x is S_A, then M_{A^x} uses $O(S + pS_A)$ space. Hence, the parallelism of an amplifier is an important factor in estimating its space requirements.

4.5 BPP and BPL Amplification

The above observations about amplification time and space result in simple characterizations of BPP applicable and BPL applicable amplifiers:

Proposition 4.5.1 *An $(l, r, k, \delta, \epsilon)$ amplifier that uses T private time is BPP applicable iff k and T are polynomial in r.*

Proposition 4.5.2 *A p-parallel $(l, r, k, \delta, \epsilon)$ amplifier that uses T private time and S private space is BPL applicable iff p is constant, T is polynomial in r and S is logarithmic in r.*

The proofs of these propositions are straightforward.

All the amplifiers presented in chapter 5 run in a polynomial private time. Hence, they are all BPP applicable. However, all the straightforward implementations of these amplifiers (except for the naive one) store their random seed on the work tape. Hence they use $\Omega(r)$ private space, and are not BPL applicable. The naive amplifier is BPL applicable (it is 1-parallel and uses $O(\log r)$ private space), but it consumes too much random bits. In chapter 7 we present first examples for random bit efficient BPL applicable amplifiers.

Since the main issue in this paper is space and not time, we will ignore time calculations for the rest of the discussion. It is easy to check that all the amplifiers we present run in a polynomial time. For verifying BPL applicability we will be interested only in the parallelism and the private space. This the motivation for the next definition:

Definition 4.5.1 *An amplifier $M_?$ is called (S, p) efficient, if it is p-parallel and uses S private space.*

4.6 Extractor Explicitness

The equivalence between amplifiers and extractors can be used in the context of amplifier applicability too. *Explicit* extractors (ones in which neighborhoods are easy to compute) yield efficient amplifiers and vice versa.

We begin by introducing two new Turing machine variants:

Definition 4.6.1 *An* **online Turing machine** *is a Turing machine with a one way read-only input tape.*

Definition 4.6.2 *A* $\boldsymbol{k}$**-output Turing machine** *is a Turing machine that has k one-way write-only output tapes. The machine has a special address tape, from which it takes the index of the output tape to be accessed next.*

Such a machine computes a function $f : \{0,1\}^* \to (\{0,1\}^*)^k$, where the i^{th} coordinate of the output is written on the i^{th} output tape. The machine needs the special address tape, since k is allowed to grow with the input length. For space calculations this tape is considered as a work tape, hence the machine is charged for the space it consumes.

We present below notions that correspond to active simulations and amplifier parallelism in the setting of k-output machines. k-output machines will be used later to compute the k neighbors of a node on the left of an (l, r, k) bipartite graph. These neighbors will be used by an amplifier as pseudo-random strings for the simulations of the original algorithm. A simulation is active if it received some of its random bits but not all of them. This happens exactly when the corresponding output tape contains some of the bits, that should be written on it, but not all of them. This is the motivation for the next definition:

Definition 4.6.3 *Fix an input y for a k-output machine, and consider its execution on this input. We denote by $t^y_{i,start}$ and $t^y_{i,end}$ the time steps in which the first and the last bits (respectively) are written on the i^{th} output tape. We say that this tape is* **alive** *at time t on the input y if $t^y_{i,start} \leq t \leq t^y_{i,end}$.*

Definition 4.6.4 *A k-output machine is called* $\boldsymbol{p}$**-parallel***, if p is the maximal (over all inputs y and all time steps t) number of live output tapes.*

Definition 4.6.5 *A k-output machine is called* $\boldsymbol{(S, p)}$ **efficient***, if it is p-parallel and uses S space.*

We are now ready for the explicitness definition:

Definition 4.6.6 *A family of (l, r, k) bipartite graphs $\{E_r\}$ (where $l = l(r)$ and $k = k(r)$) is called* $\boldsymbol{(S, p)}$ **explicit***, if there is an (S, p) efficient k-output machine N_E (called the* **neighborhood machine***) with two input tapes: a two way input tape on which it gets r and a one way input tape on which it gets a node v on the left side of E_r. N_E outputs the k neighbors of v on its output tapes, one neighbor per tape.*

Note that there is no requirement that the i^{th} neighbor will be written on the i^{th} output tape. Any permutation of the neighbors is acceptable.

The next theorem demonstrates the equivalence between weak extractor explicitness and amplifier efficiency. This, together with lemma 4.2.1 and proposition 4.2.1 enables us to concentrate on the study of extractors and dispersers, and obtain conclusions about amplifiers.

Theorem 4.6.1 (Equivalence Theorem) *For $S \geq \log k$ an $(l, r, k, \delta, \epsilon)$ amplifier $M_?$ is $(O(S), p)$ efficient if and only if the family of (l, r, k) bipartite graphs $\{E_{M,r}\}$ is $(O(S), p)$ explicit.*

Proof: Assume $M_?$ is $(O(S), p)$ efficient. We construct an (S, p) efficient neighborhood algorithm N_E for the family $\{E_{M,r}\}$, which is based on $M_?$. When given r on its two way input tape and y on its one way input tape, N_E runs $M_?$, while supplying it r as input and y as a random string. N_E ignores the simulation steps of $M_?$. When $M_?$ submits a random bit to the i^{th} simulation, N_E writes this bit on its i^{th} output tape. It can be easily verified that N_E computes neighborhoods in $E_{M,r}$.

N_E follows the computation steps of $M_?$ accurately. Hence, it uses the same space $(O(S))$. Moreover, N_E's schedule for its output tapes is the same as $M_?$'s schedule for its simulations. Therefore, at any given moment the number of living output tapes N_E has is identical to the number of active simulations $M_?$ has. It follows that since $M_?$ is p-parallel then so is N_E. We conclude that $\{E_{M,r}\}$ is $(O(S), p)$ explicit.

For the opposite direction, assume $\{E_{M,r}\}$ is an (S, p) explicit family of bipartite graphs and let N_E be its neighborhood algorithm. We present an $(O(S), p)$ efficient $(l, r, k, \delta, \epsilon)$ amplifier, which is based on N_E. Without loss of generality, $M_?$ itself is this amplifier.

When given an input r and a random string y $M_?$ runs N_E with r on its two way input tape and y on its one way input tape. $M_?$ uses the bits N_E writes on its i^{th} output tape as random bits submitted to its i^{th} simulation. It is easy to verify that $M_?$ is an $(l, r, k, \delta, \epsilon)$ amplifier.

$M_?$ uses the same space as N_E, except for an additional $O(\log k)$ bits for the acceptance and rejection counters on its majority tape. Since $S \geq \log k$, $M_?$ also uses $O(S)$ space. The parallelism of $M_?$ is the same as of N_E, since the output tapes schedule is identical to the simulation schedule. We conclude that $M_?$ is $(O(S), p)$ efficient. □

4.7 One Sided Error Amplification

So far we discussed amplification of two-sided error randomized algorithms only. However, since a significant fraction of the randomized algorithms we use are one-sided error, it is interesting to study amplification of such algorithms too.

Since two-sided error is a generalization of one-sided error, naturally all the amplification methods for two-sided error algorithms apply also for one-sided error algorithms. However, for one-sided error algorithms it is easier to make the amplification. For example, if the algorithm is wrong only on inputs that belong to the language, then "yes" answers are *always* correct. Therefore, if the amplifier runs k copies of the original algorithm, and bumps into even just a single "yes", it suffices to determine for sure that the input belongs to the language. In other words, the amplifier should perform an OR function on the results of the k simulations instead of a majority vote.

One-sided error black box amplifiers are equivalent to dispersers. Hence, a similar connection to the one presented in lemma 4.2.1 exists also for one-sided error algorithms:

Lemma 4.7.1 *$M_?$ is an $(l, r, k, \delta, \epsilon)$ one-sided error amplifier if and only if for all r $E_{M,r}$ is an $(l, r, k, \delta, \epsilon)$ disperser.*

The proof is very similar to the proof of lemma 4.2.1.

Since the construction of the graph family $\{E_{M,r}\}$ does not depend on the function performed on the results of the simulations, it is the same for one-sided error amplifiers. It follows that the equivalence theorem holds also for such amplifiers.

We will not deal with one-sided error amplification for the rest of the paper.

Chapter 5

Amplification Methods

This chapter surveys the main amplification methods introduced during the last decade by several researcher-s. The main properties of these methods are summarized in the table below.

Method	No. of Random Bits	Error Probability
Naive Method	kr	$2^{-\Omega(k)}$
[CG89]	$2r$	$O(\frac{1}{k})$
[KPS85]	r	$O(\frac{1}{k})$
[IZ89]	$O(r+k^2)$	$2^{-\Omega(k)}$
[Nis92]	$O(r \log k)$	$2^{-\Omega(k)}$
[AKS87, CW89, IZ89]	$r+O(k)$	$2^{-\Omega(k)}$

The naive method achieves the optimal exponential descent, but the price it too high: the number of random bits is multiplied by a factor of k. Karp, Pippenger and Sipser ([KPS85]) and Chor and Goldreich ([CG89]) use an optimal number of random bits ($O(r)$), but their error probability drops only linearly with k. Impagliazzo and Zuckerman ([IZ89]) and Nisan ([Nis92]) do much better in reducing the error probability, by achieving the optimal exponential descent. However, they require more random bits than the previous methods. The method of Ajtai, Komlós and Szemerédi ([AKS87, CW89, IZ89]) is the only one that succeeds both to reduce the error exponentially and to use a minimal number of random bits ($O(r)$).

In the sequel we describe in some detail each of the listed above methods. Based on the equivalence theorem (theorem 4.6.1) it suffices to present the corresponding weak extractors and to prove their structural and explicitness properties.

We use the following notations throughout the chapter:

1. E - An $(l, r, k, \delta, \epsilon)$ weak extractor.
2. V_1 - The left side of E.
3. V_2 - The right side of E. V_2 always equals to $\{0,1\}^r$.
4. $\mu(A)$ - The proportional volume of a subset $A \subseteq \{0,1\}^r$, i.e. $\mu(A) = \frac{|A|}{2^r}$.
5. W - An arbitrary subset of V_2, for which $\mu(W) < \epsilon$.
6. B - The set of nodes in V_1 whose majority of neighbors are in W.
7. $f_W : \{0,1\}^r \rightarrow \{0,1\}$ - The characteristic function of W ($f_W(z) = 1$ iff $z \in W$).

8. $\overline{W}$ - The complement of W: $\overline{W} = \{0,1\}^r \setminus W$.

When introducing a construction of a weak extractor, it suffices to describe the two following elements:

1. V_1 - The set that constitutes the left side of E.

2. $N : V_1 \to (V_2)^k$ - The function that determines which are the neighbors of any given node in V_1.

The main issue in a weak extractor performance proof is establishing a lower bound on δ. The strategy for doing that is rather similar for all the weak extractors we discuss. We outline the general framework of this strategy here, and concentrate on the unique details at each proof.

Assume we wish to prove $\delta \geq q$ for some q. Take an arbitrary subset $W \subseteq V_2$, $\mu(W) < \epsilon$, and consider the set of nodes in V_1 whose majority of neighbors is in W. Call this set B. We prove that $\mu(B) < q$.

Pick $y \in_R V_1$ at random. Note that $\mu(B) = Pr[y \in B]$. Denote the k neighbors of y by $y_1, \ldots, y_k$. These are random variables too. We prove that $Pr[y \in B] = Pr[MAJ(f_W(y_1), \ldots, f_W(y_k)) = 1] < q$.

We do not describe neighborhood algorithms for the weak extractors presented in the chapter. The implementations of such algorithms are straightforward. All of them compute one neighbor at a time, implying they are 1-parallel. All of them, except for the naive one, save their input or some other related data of the same size on the work tape. Hence, they use $\Omega(r)$ space. The naive method is the only one that uses $O(\log r)$ space. All of them run in a polynomial time. We will not consider the efficiency issue for the rest of the chapter.

5.1 The Naive Method

Specification

1. $V_1 = \{0,1\}^{kr}$.

2. Given $y \in V_1$ divide it to k consecutive substrings of length r: $y = y_1 \circ y_2 \circ \cdots \circ y_k$, where $y_1, \ldots, y_k \in \{0,1\}^r$. $N(y) = (y_1, \ldots, y_k)$.

Theorem 5.1.1 *The naive method is an $(kr, r, k, \epsilon^{\Omega(k)}, \epsilon)$ weak extractor.*

Proof: We choose $y \in_R V_1$ at random and bound the probability that $Pr[y \in B]$. Consider the k random variables $y_1, \ldots, y_k$, where y_i is the i^{th} coordinate in $N(y)$. Note that since y distributes uniformly on $\{0,1\}^{kr}$, and since y_i is a substring of y, y_i distributes uniformly on $\{0,1\}^r$. Moreover, $y_1, \ldots, y_k$ are independent random variables.
Define k indicator random variables $X_1, \ldots, X_k$ as follows:

$$X_i = \begin{cases} 1 & y_i \in W \\ 0 & y_i \notin W \end{cases}$$

These random variables are independent, since the random strings y_i are independent. The expectation of X_i is the probability of y_i to be in W, which is $\mu(W)$ since y_i distributes uniformly on $\{0,1\}^r$. $\mu(W) < \epsilon$, implying that $E[X_i] < \epsilon$.

$\sum_{i=1}^{k} X_i$ denotes the number of y's neighbors that belong to W. Since B is exactly the set of y's, for which at least $\frac{k}{2}$ of their neighbors belong to W, then $Pr[y \in B] = Pr[\sum_{i=1}^{k} X_i] \geq \frac{k}{2}$.

By Chernoff bounds (theorem 2.1.3), $Pr[\sum_{i=1}^{k} X_i > \frac{k}{2}] = Pr[\sum_{i=1}^{k} X_i > (1 + (\frac{1}{2\epsilon} - 1))\epsilon k] < e^{\frac{-(\frac{1}{2\epsilon}-1)^2 \epsilon k}{6}} = \epsilon^{\Omega(k)}$. □

5.2 The Chor-Goldreich Method

Specification

1. $V_1 = H = \{h : \{0,1\}^r \to \{0,1\}^r\}$. H is a universal family of hash functions, where each $h \in H$ is represented by $2r$ bits and can be computed in logspace (recall, for example, the construction of such a family in section 2.3.3). We denote an arbitrary node in V_1 by h.

2. $N(h) = (h(1), h(2), \ldots, h(k))$.

Theorem 5.2.1 (Chor-Goldreich) *The CG construction is a $(2r, r, k, O(\frac{1}{k}), \epsilon)$ weak extractor.*

Proof: We choose $h \in_R H$ at random, and bound the probability $Pr[h \in B]$. We define k random variables as follows:

$$X_i = \begin{cases} 1 & h(i) \in W \\ 0 & h(i) \notin W \end{cases}$$

Let us estimate the expectation and the variance of each X_i. $E[X_i] = Pr_{h \in_R H}[X_i = 1] = Pr_{h \in_R H}[h(i) \in W]$. Since h is chosen from a universal family of hash functions, $h(i)$ distributes uniformly on $\{0,1\}^r$. Therefore, the last probability is $\mu(W)$, which is less than ϵ. We conclude that $X_1, \ldots, X_k$ have all the same expectation and it is $< \epsilon$. $Var[X_i] = E[X_i^2] - E[X_i]^2 \leq E[X_i^2]$. Notice that $X_i^2 = X_i$, hence $Var[X_i] \leq E[X_i] < \epsilon$.

Recall proposition 2.3.1 which states that the variables $h(1), \ldots, h(k)$ are pairwise independent. This implies that $X_1, \ldots, X_k$ are also pairwise independent.

Define X to be the sum $\sum_{i=1}^{k} X_i$. The linearity of the expectation operator implies that $E[X] = \sum_{i=1}^{k} E[X_i] < k\epsilon$. Since $X_1, \ldots, X_k$ are pairwise independent it holds also that $Var[X] = \sum_{i=1}^{k} Var[X_i] < k\epsilon$.

Note that $Pr[h \in B] = Pr[X \geq \frac{k}{2}]$. Now, $Pr[X \geq \frac{k}{2}] = Pr[X - k\epsilon \geq k(\frac{1}{2} - \epsilon)] \leq Pr[|X - E[X]| \geq k(\frac{1}{2} - \epsilon)]$. Using Chebyshev inequality we obtain that the last probability $\leq \frac{Var[X]}{k^2(\frac{1}{2}-\epsilon)^2} < \frac{k\epsilon}{k^2(\frac{1}{2}-\epsilon)^2} = O(\frac{1}{k})$.
□

5.3 The Karp-Pippenger-Sipser Method

Specification

1. $V_1 = G^{(q)}$. G is a logspace explicit $(2^r, d, \lambda)$ expander, for which $\frac{\lambda}{d} < \frac{1}{2}(\frac{1}{2} - \epsilon)$ and d is constant (recall corollary 2.2.1). q is $\log_d k$. The nodes of G and of its powers correspond to the elements of $\{0,1\}^r$.

2. Given a node $y \in G^{(q)}$, denote by $y_1, \ldots, y_k$ its $k = d^q$ neighbors. Then, $N(y) = (y_1, \ldots, y_k)$.

Theorem 5.3.1 (Karp-Pippenger-Sipser) *The KPS construction is an $(r, r, k, O(\frac{1}{k}), \epsilon)$ weak extractor.*

Proof: Since $V_1 = V_2 = \{0,1\}^r$ is exactly the vertex set of $G^{(q)}$, we can recognize B and W as sets of nodes in $G^{(q)}$. Denote by $E(B, W)$ the number of edges between nodes of B and nodes of W in $G^{(q)}$. It holds that $E(B, W) \geq |B|\frac{d^q}{2}$, since $G^{(q)}$ is d^q-regular and since the majority of the neighbors of each node in B belongs to W.

By the expander mixing lemma (lemma 2.2.1) $|E(B, W) - \frac{d^q|B||W|}{2^r}| \leq \lambda^q 2^r$. $\epsilon < \frac{1}{2}$ implies that $E(B, W) > |B|d^q\epsilon > \frac{d^q|B||W|}{2^r}$. Therefore, $\lambda^q 2^r \geq |B|\frac{d^q}{2} - d^q|B|\epsilon = |B|d^q(\frac{1}{2} - \epsilon)$. It follows that $\mu(B) = \frac{|B|}{2^r} \leq \frac{1}{\frac{1}{2}-\epsilon}(\frac{\lambda}{d})^q \leq (\frac{1}{\frac{1}{2}-\epsilon})^q \cdot (\frac{\lambda}{d})^q < (\frac{1}{2})^q = O(\frac{1}{k})$. □

5.4 The Impagliazzo-Zuckerman Method

Specification

1. $V_1 = H \times \{0,1\}^r \times (\{0,1\}^m)^{k-1}$. $H = \{h : \{0,1\}^r \to \{0,1\}^{r-m}\}$ is a universal family of hash functions and m is some value greater than 0 and less than r. We denote each element of V_1 by the sequence $(h, y_1, s_2, \ldots, s_k)$ where $h \in H$, $y_1 \in \{0,1\}^r$ and $s_2, \ldots, s_k \in \{0,1\}^m$.

2. $N(h, y_1, s_2, \ldots, s_k) = (y_1, y_2, \ldots, y_k)$, where $y_i = h(y_{i-1}) \circ s_i$ for $i = 2, \ldots, k$.[1]

It is good to think of k and m as $\approx \sqrt{r}$. The amplifier that uses this weak extractor generates a sequence of k r-bit strings, which should have a nearly uniform distribution. The amplifier "recycles" the totally random bits that constitute the first element of sequence (y_1) in the following elements. It does so by repeatedly applying the random hash function on this element and adding a small number of truly random bits at each stage.

Theorem 5.4.1 (Impagliazzo-Zuckerman) *For any $0 < \epsilon < \frac{1}{16}$ and $0 < m < r$ the IZ construction is a $(3r + (k-1)m, r, k, 2^k(\epsilon^{\frac{k}{2}} + k2^{-\frac{m}{2}-1}), \epsilon)$ weak extractor.*

Note that for making δ exponentially small in k we need to take $m \geq 4k$. On the other hand, m should be less than r. It follows, that k should be at most $\frac{r}{4}$.

Proof: We choose $h \in_R H, y_1 \in_R \{0,1\}^r$ and $s_2, \ldots, s_k \in_R \{0,1\}^m$ and bound the probability $Pr[MAJ(f_W(y_1), \ldots, f_W(y_k)) = 1]$. This equals to $Pr[\bigvee_{1 \leq i_1 < \ldots < i_{\frac{k}{2}} \leq k} \bigwedge_{j=1}^{\frac{k}{2}} (f_W(y_{i_j}) = 1)] \leq \sum_{1 \leq i_1 < \ldots < i_{\frac{k}{2}} \leq k} Pr[\bigwedge_{j=1}^{\frac{k}{2}} (f_W(y_{i_j}) = 1)]$.

For bounding the probability that $f_W(y_{i_1}), \ldots, f_W(y_{i_{\frac{k}{2}}}) = 1$, we consider the distribution induced by the random variable $(f_W(y_1), \ldots, f_W(y_k))$ on the set $\{0,1\}^k$. We prove that the distance between this distribution and the distribution of $(f_W(z_1), \ldots, f_W(z_k))$, where $z_1, \ldots, z_k \in_R \{0,1\}^r$ is at most $k2^{-\frac{m}{2}-1}$. Since the probability that $f_W(z_{i_1}), \ldots, f_W(z_{i_{\frac{k}{2}}}) = 1$ under this distribution is $\mu(W)^{\frac{k}{2}}$, then the probability that $f_W(y_{i_1}), \ldots, f_W(y_{i_{\frac{k}{2}}}) = 1$ would be at most $\mu(W)^{\frac{k}{2}} + k2^{-\frac{m}{2}-1} < \epsilon^{\frac{k}{2}} + k2^{-\frac{m}{2}-1}$. Summing up over all possible $1 \leq i_1 < \ldots < i_{\frac{k}{2}} \leq k$ we obtain that the total error probability is $< \binom{k}{\frac{k}{2}}(\epsilon^{\frac{k}{2}} + k2^{-\frac{m}{2}-1}) < 2^k(\epsilon^{\frac{k}{2}} + k2^{-\frac{m}{2}-1})$.

For bounding the distance between these distributions, we use several other intermediate distributions. We define the following for each $i = 1, \ldots, k$:

1. $L_i = H \times \{0,1\}^r \times (\{0,1\}^m)^i$. L_i is the left side of the IZ weak extractor, when $k = i+1$.

2. $R_i = H \times (\{0,1\}^r)^{i+1}$. R_i is H multiplied by the set of possible neighbors of a node in L_i.

3. $E_i : L_i \to R_i$, $E_i(h, y_1, s_2, \ldots, s_i) = (h, y_1, y_2, \ldots, y_i)$ where $y_j = h(y_{j-1}) \circ s_j$ for $j = 2, \ldots, i$. E_i is actually the neighborhood function of the IZ weak extractor, when $k = i+1$.

4. Let U_{L_i} be the uniform distribution on L_i. $(U_{L_i})^{E_i}$ is the distribution of the neighbors of a random node in L_i.

5. $P_i = \{0,1\}^i \times H \times \{0,1\}^r$.

[1] $\circ$ denotes concatenation of two strings.

6. $f_i : R_i \to P_i$, where $f_i(h, y_1, \ldots, y_{i+1}) = (f_W(y_1), \ldots, f_W(y_i), h, y_{i+1})$. f_i indicates which of the first i neighbors of a node in L_i belongs to W.

7. $D_i = ((U_{L_i})^{E_i})^{f_i}$. D_i is the distribution on P_i induced by the indications of which of the neighbors of random node in L_i belongs to W.

8. $U_i = (U_{R_i})^{f_i}$, where U_{R_i} is the uniform distribution on R_i.

9. $Q_i = \{0,1\}^i$

10. $g_i : P_i \to Q_i$, $g_i(b_1, \ldots, b_i, h, y_{i+1}) = (b_1, \ldots, b_i)$. Notice that $D_k^{g_k}$ is the distribution of the random variable $(f_W(y_1), \ldots, f_W(y_k))$ and $U_k^{g_k}$ is the distribution of the random variable $(f_W(z_1), \ldots, f_W(z_k))$.

We will prove by induction on i that $d(D_i, U_i) \le i2^{-\frac{m}{2}-1}$. It would follow that $d(D_k, U_k) \le k2^{-\frac{m}{2}-1}$. Using proposition 2.1.2 we obtain that $d(D_k^{g_k}, U_k^{g_k}) \le d(D_k, U_k) \le k2^{-\frac{m}{2}-1}$, as wanted.

For the induction base, we should prove that $d(D_1, U_1) \le 2^{-\frac{m}{2}-1}$. Note that D_1 is the distribution of the random variable $(f_W(y_1), h, h(y_1) \circ s_2)$ where $h \in_R H, y_1 \in_R \{0,1\}^r$ and $s_2 \in_R \{0,1\}^m$. Since s_2 is chosen independently from h and y_1, we can present D_1 as an independent Cartesian product of two distributions: $D_1^{(a)}$ on the string $(f_W(y_1), h, h(y_1))$ and $D_1^{(b)}$ on s_2. U_1 is the distribution of the random variable $(f_W(z_1), h, z_2)$, where $z_1, z_2 \in_R \{0,1\}^r$ and $h \in_R H$. We can present U_1 too as an independent Cartesian product of $U_1^{(a)}$ and $U_1^{(b)}$, where $U_1^{(b)}$ is the uniform distribution on the last m bits. Note that $D_1^{(b)} = U_1^{(b)} = U_{\{0,1\}^m}$, hence by proposition 2.1.1 $d(D_1, U_1) \le d(D_1^{(a)}, U_1^{(a)})$.

By the statistical distance definition:

$$d(D_1^{(a)}, U_1^{(a)}) = \frac{1}{2}|\sum_{\hat{b}\in\{0,1\},\hat{h}\in H,\hat{z}\in\{0,1\}^{r-m}} (Pr_{h\in_R H, y_1\in_R\{0,1\}^r}[f_W(y_1) = \hat{b} \wedge h = \hat{h} \wedge h(y_1) = \hat{z}] -$$

$$Pr_{h\in_R H, z_1\in_R\{0,1\}^r, \overline{z}_2\in_R\{0,1\}^{r-m}}[f_W(z_1) = \hat{b} \wedge h = \hat{h} \wedge \overline{z}_2 = \hat{z}])|$$

Consider, for example, the first summand over the cases $\hat{b} = 0, 1$. By the complete probability formula it equals to:

$$Pr[h = \hat{h} \wedge h(y_1) = \hat{z} \mid f_W(y_1) = 1] \cdot Pr[f_W(y_1) = 1] + Pr[h = \hat{h} \wedge h(y_1) = \hat{z} \mid f_W(y_1) = 0] \cdot Pr[f_W(y_1) = 0] =$$

$$Pr_{h\in_R H, y_1\in_R W}[h = \hat{h} \wedge h(y_1) = \hat{z}]\mu(W) + Pr_{h\in_R H, y_1\in_R \overline{W}}[h = \hat{h} \wedge h(y_1) = \hat{z}](1 - \mu(W))$$

Define the following:

1. S_W - The distribution of $(h, h(y_1))$ where $h \in_R H$ and $y_1 \in_R W$.
2. $S_{\overline{W}}$ - The distribution of $(h, h(y_1))$ where $h \in_R H$ and $y_1 \in_R \overline{W}$.
3. S_U - The distribution of (h, z) where $h \in_R H$ and $z \in_R \{0,1\}^{r-m}$.

We obtain:

$$d(D_1^{(a)}, U_1^{(a)}) \le \frac{1}{2}(\mu(W)d(S_W, S_U) + (1 - \mu(W))d(S_{\overline{W}}, S_U))$$

We can apply the Leftover Hash Lemma (lemma 2.3.1) to bound each of the two distances in the last expression. We substitute $U = \{0,1\}^r, T = \{0,1\}^{r-m}$ and $A = W$ for the first one and $A = \overline{W}$ for the second. We obtain that the last expression is $\le \frac{1}{4}(\mu(W)\sqrt{\frac{2^{r-m}}{|W|}} + (1-\mu(W))\sqrt{\frac{2^{r-m}}{|\overline{W}|}}) = \frac{1}{4}(\mu(W)\frac{1}{\sqrt{2^m\mu(W)}} +$

$(1-\mu(W))\frac{1}{\sqrt{2^m(1-\mu(W))}}) = (\sqrt{\mu(W)} + \sqrt{1-\mu(W)})2^{-\frac{m}{2}-2} \leq 2^{-\frac{m}{2}-1}$.

Assume now that $d(D_{i-1}, U_{i-1}) \leq (i-1)2^{-\frac{m}{2}-1}$. We will prove that $d(D_i, U_i) \leq i2^{-\frac{m}{2}-1}$. D_i is a distribution on strings of the form $(f_W(y_1), \ldots, f_W(y_i), h, y_{i+1})$, where $h \in_R H, y_1 \in_R \{0,1\}^r, s_2, \ldots, s_{i+1} \in_R \{0,1\}^m$ and $(y_1, y_2, \ldots, y_{i+1}) = N(h, y_1, s_2, \ldots, s_{i+1})$. U_i is a distribution on strings of the form $(f_W(z_1), \ldots, f_W(z_i), h, z_{i+1})$, where $h \in_R H$ and $z_1, \ldots, z_{i+1} \in_R \{0,1\}^r$. As was done in the induction base, we can present D_i and U_i as independent Cartesian products of the distribution on their last m bits and the distribution on the other bits. We denote the distributions on the prefixes by $D_i^{(a)}$ and $U_i^{(a)}$. We obtain that $d(D_i, U_i) \leq d(D_i^{(a)}, U_i^{(a)})$. Now, $D_i^{(a)}$ is a distribution on strings of the form $(f_W(y_1), \ldots, f_W(y_i), h, h(y_i))$ and $U_i^{(a)}$ is a distribution on strings of the form $(f_W(z_1), \ldots, f_W(z_i), h, \overline{z}_{i+1})$, where $\overline{z}_{i+1}$ is a uniform string of length $r-m$.

Define the following function: $q_i : P_{i-1} \rightarrow \{0,1\}^i \times H \times \{0,1\}^{r-m}, q_i(b_1, \ldots, b_{i-1}, h, y_i) = (b_1, \ldots, b_{i-1}, f_W(y_i), h, h(y_i))$. Note that $D_i^{(a)} = (D_{i-1})^{q_i}$. Hence, $d(D_i^{(a)}, U_i^{(a)}) = d((D_{i-1})^{q_i}, U_i^{(a)}) \leq d((D_{i-1})^{q_i}, (U_{i-1})^{q_i}) + d((U_{i-1})^{q_i}, U_i^{(a)})$. By proposition 2.1.2 $d((D_{i-1})^{q_i}, (U_{i-1})^{q_i}) \leq d(D_{i-1}, U_{i-1})$. By the induction hypothesis this is $\leq (i-1)2^{-\frac{m}{2}-1}$. $(U_{i-1})^{q_i}$ is a distribution on strings of the form $(f_W(z_1), \ldots, f_W(z_{i-1}), f_W(z_i), h, h(z_i))$, where $h \in_R H$ and $z_1, \ldots, z_i \in_R \{0,1\}^r$. $U_i^{(a)}$ is a distribution on strings of the form $(f_W(z_1), \ldots, f_W(z_{i-1}), f_W(z_i), h, \overline{z}_{i+1})$, where $h \in_R H, z_1, \ldots, z_i \in_R \{0,1\}^r$ and $\overline{z}_{i+1} \in_R \{0,1\}^{r-m}$. Note that both distributions can be presented as an independent Cartesian product of their first $i-1$ bits and the rest of the bits. Note that the distribution of $(U_{i-1})^{q_i}$ on the rest of the bits is exactly $D_1^{(a)}$ and the distribution of $U_i^{(a)}$ on these bits is exactly $U_1^{(a)}$. The distributions on the first $i-1$ bits are the same, hence proposition 2.1.1 implies that $d((U_{i-1})^{q_i}, U_i^{(a)}) \leq d(D_1^{(a)}, U_1^{(a)})$, which by the proof of the induction base is $\leq 2^{-\frac{m}{2}-1}$. We conclude that $d(D_i, U_i) \leq (i-1)2^{-\frac{m}{2}-1} + 2^{-\frac{m}{2}-1} = i2^{-\frac{m}{2}-1}$, as wanted. □

5.5 Nisan's Method

The Nisan weak extractor, like the IZ and the CG weak extractors, is based on the concept of universal hashing. The weak extractor uses a universal hash family $H = \{h : \{0,1\}^r \rightarrow \{0,1\}^r\}$ (it can use, for example, the construction of Carter and Wegman, that was described in section 2.3.3). Nisan defines the following functions:

$$
\begin{aligned}
&G_i : \{0,1\}^r \times H^i \rightarrow (\{0,1\}^r)^{2^i} \\
&G_0(y) = y \\
&G_i(y, h_1, \ldots, h_i) = G_{i-1}(y, h_1, \ldots, h_{i-1}) \circ G_{i-1}(h_i(y), h_1, \ldots, h_{i-1})
\end{aligned}
$$

This family of functions has a nice characterization that appears in [LW95]. Look at the complete binary tree depicted in figure 5.1. The nodes of the tree are labeled by strings from $\{0,1\}^r$. The root is labeled by y. If a node v at depth j is labeled by the string z, then its left son is labeled by z too and its right son is labeled by $h_{i-j+1}(z)$. $G_i(y, h_1, \ldots, h_i)$ is simply the concatenation of the strings that appear at the i^{th} level of the tree.

Specification

1. $V_1 = \{0,1\}^r \times H^m$. H is a universal hash family $H = \{h : \{0,1\}^r \rightarrow \{0,1\}^r\}$ and $m = \lceil \log k \rceil$. We denote a node in V_1 by $(y, h_1, \ldots, h_m)$.

2. $N(y, h_1, \ldots, h_m) = G_m(y, h_1, \ldots, h_m)$. The output of G_m is a concatenation of k strings of length r, denoted by $y_1, \ldots, y_k$.

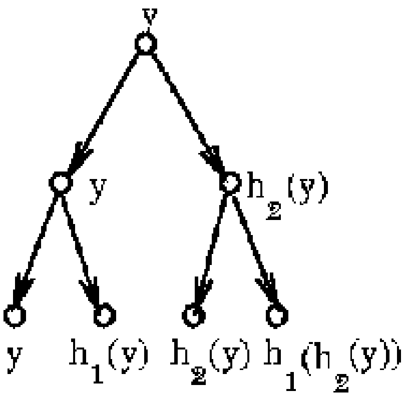

Figure 5.1: Nisan's Construction

Theorem 5.5.1 (Nisan) *For any* $0 < \epsilon < \frac{1}{16}$ *Nisan's construction is an* $(O(r \log k), r, k, 2^k(\epsilon^{\frac{k}{2}} + O(k2^{-\frac{r}{3}})), \epsilon)$ *weak extractor.*

Note that it suffices to take $k \leq \frac{r}{4}$ to gain an exponentially small error probability.

Proof: Define for any set of indices $1 \leq i_1 < i_2 < \ldots < i_j \leq k$ and for every $i_j \leq I \leq k$: $W^I_{i_1,\ldots,i_j} = (\{0,1\}^r)^{i_1-1} \times W \times (\{0,1\}^r)^{i_2-i_1-1} \times W \times \ldots \times (\{0,1\}^r)^{I-i_j}$ ($W^I_{i_1,\ldots,i_j}$ is an I-length Cartesian product of $\{0,1\}^r$ and W, where W appears in positions $i_1, i_2, \ldots, i_j$ and $\{0,1\}^r$ appears in all the rest).

For any set of indices $1 \leq i_1 < i_2 < \ldots < i_j \leq k$ and for every i such that $i_j \leq 2^i \leq k$ we define: $B^{h_1,\ldots,h_i}_{i_1,\ldots,i_j} = \{y \in \{0,1\}^r \mid G_i(y, h_1, \ldots, h_i) \in W^{2^i}_{i_1,\ldots,i_j}\}$. These are exactly the y's for which $y_{i_1}, \ldots, y_{i_j}$ belong to W, given that $h_1, \ldots, h_i$ are the chosen hash functions.

We choose $y \in_R \{0,1\}^r$ and $h_1, \ldots, h_m \in_R H$ at random and wish to bound the probability $Pr[MAJ(f_W(y_1), \ldots, f_W(y_k)) = 1]$. This equals to $Pr[\bigvee_{1 \leq i_1 < i_2 < \ldots < i_{\frac{k}{2}} \leq k} y_{i_1}, \ldots, y_{i_{\frac{k}{2}}} \in W] \leq$
$\sum_{1 \leq i_1 < i_2 < \ldots < i_{\frac{k}{2}} \leq k} Pr[y_{i_1}, \ldots, y_{i_{\frac{k}{2}}} \in W] =$
$\sum_{1 \leq i_1 < i_2 < \ldots < i_{\frac{k}{2}} \leq k} Pr[G_m(y, h_1, \ldots, h_m) \in W^k_{i_1,\ldots,i_{\frac{k}{2}}}] =$
$\sum_{1 \leq i_1 < i_2 < \ldots < i_{\frac{k}{2}} \leq k} Pr[y \in B^{h_1,\ldots,h_m}_{i_1,\ldots,i_{\frac{k}{2}}}]$.

We will prove that $Pr_{y \in_R \{0,1\}^r, h_1,\ldots,h_m \in_R H}[y \in B^{h_1,\ldots,h_i}_{i_1,\ldots,i_j}] \leq \mu(W)^j + (k+1)\alpha$, where $\alpha = 2^{-\frac{r}{3}}$. It would follow that $Pr[y \in B^{h_1,\ldots,h_m}_{i_1,\ldots,i_{\frac{k}{2}}}] \leq \mu(W)^{\frac{k}{2}} + (k+1)\alpha < \epsilon^{\frac{k}{2}} + O(k\alpha)$. Hence, $Pr[MAJ(f_W(y_1), \ldots, f_W(y_k)) = 1] < \binom{k}{\frac{k}{2}}(\epsilon^{\frac{k}{2}} + O(k\alpha)) < 2^k(\epsilon^{\frac{k}{2}} + O(k\alpha))$, as wanted.

We call a sequence of m hash functions $h_1, \ldots, h_m \in H$ *good* if each h_i is (α, U, V)-good[2] for any pair of sets $U, V \subseteq \{0,1\}^r$ we need (we will specify to exactly which pairs we mean in the sequel). Fix a good sequence $h_1, \ldots, h_m$. We prove by induction on i that $Pr_{y \in_R \{0,1\}^r}[y \in B^{h_1,\ldots,h_i}_{i_1,\ldots,i_j}] \leq \mu(W)^j + 2\alpha$.

For $i = 0$, the probability we wish to bound is exactly $Pr_{y \in_R \{0,1\}^r}[y \in W]$, which is $\mu(W)$. Hence, the claim is satisfied for this case. Assume the claim is true for $i - 1$. We will show its correctness for i.

Let t be the maximal for which $i_t < 2^{i-1}$. It holds that $Pr_{y \in_R \{0,1\}^r}[y \in B^{h_1,\ldots,h_i}_{i_1,\ldots,i_j}] =$
$Pr_{y \in_R \{0,1\}^r}[G_i(y, h_1, \ldots, h_i) \in W^{2^i}_{i_1,\ldots,i_j}] =$

[2] Recall definition 2.3.3.

$Pr_{y\in_R\{0,1\}^r}[G_{i-1}(y,h_1,\ldots,h_{i-1}) \in W^{2^{i-1}}_{i_1,\ldots,i_t} \wedge G_{i-1}(h_i(y),h_1,\ldots,h_{i-1}) \in W^{2^{i-1}}_{i_{t+1}-2^{i-1},\ldots,i_j-2^{i-1}}] =$
$Pr_{y\in_R\{0,1\}^r}[y \in B^{h_1,\ldots,h_{i-1}}_{i_1,\ldots,i_t} \wedge h_i(y) \in B^{h_1,\ldots,h_{i-1}}_{i_{t+1}-2^{i-1},\ldots,i_j-2^{i-1}}]$.
Let us denote the set $B^{h_1,\ldots,h_{i-1}}_{i_1,\ldots,i_t}$ by B_1 and the set $B^{h_1,\ldots,h_{i-1}}_{i_{t+1}-2^{i-1},\ldots,i_j-2^{i-1}}$ by B_2. We separate to three cases:

Case 1: $t=0$
Then, $W^{2^{i-1}}_{i_1,\ldots,i_t} = (\{0,1\}^r)^{2^{i-1}}$. Hence, $B_1 = \{0,1\}^r$, which means that $Pr_{y\in_R\{0,1\}^r}[y \in B_1 \wedge h_i(y) \in B_2] = Pr_{y\in_R\{0,1\}^r}[h_i(y) \in B_2]$. $h_i(y)$ distributes uniformly on $\{0,1\}^r$ when y is chosen at random from $\{0,1\}^r$ (recall the universal hash functions definition). Hence, the last probability equals to $Pr_{y\in_R\{0,1\}^r}[y \in B_2]$. The induction hypothesis implies that this is $\le \mu(W)^j + 2\alpha$.

Case 2: $t=j$
It means that $W^{2^{i-1}}_{i_{t+1}-2^{i-1},\ldots,i_j-2^{i-1}} = (\{0,1\}^r)^{2^{i-1}}$ implying that $B_2 = \{0,1\}^r$. It follows that $Pr_{y\in_R\{0,1\}^r}[y \in B_1 \wedge h_i(y) \in B_2] = Pr_{y\in_R\{0,1\}^r}[y \in B_1]$, and using the induction hypothesis we obtain that this is $\le \mu(W)^j + 2\alpha$.

Case 3: $0<t<j$
By our assumption that $h_1,\ldots,h_m$ is a good sequence of hash functions, it follows that h_i is (α,B_1,B_2)-good. Hence, $Pr_{y\in_R\{0,1\}^r}[y \in B_1 \wedge h_i(y) \in B_2] \le Pr_{y,z\in_R\{0,1\}^r}[y \in B_1 \wedge z \in B_2] + \alpha = Pr_{y\in_R\{0,1\}^r}[y \in B_1] \cdot Pr_{z\in_R\{0,1\}^r}[z \in B_2] + \alpha$.

We can use the induction hypothesis to bound each of these probabilities, and obtain: $Pr_{y\in_R\{0,1\}^r}[y \in B^{h_1,\ldots,h_{i-1}}_{i_1,\ldots,i_t}] \cdot Pr_{z\in_R\{0,1\}^r}[z \in B^{h_1,\ldots,h_{i-1}}_{i_{t+1}-2^{i-1},\ldots,i_j-2^{i-1}}] + \alpha \le (\mu(W)^t + 2\alpha)(\mu(W)^{j-t} + 2\alpha) + \alpha = (\mu(W)^j + 4\alpha^2 + 2\alpha(\mu(W)^t + \mu(W)^{j-t}) + \alpha$. Since $\mu(W) < \epsilon < \frac{1}{16}$, since $\alpha = 2^{-\frac{r}{3}}$ and since $0<t<j$, the last expression is $\le \mu(W)^j + 2\alpha$, as wanted.

We conclude that for a good sequence of hash functions, $Pr_{y\in_R\{0,1\}^r}[y \in B^{h_1,\ldots,h_m}_{i_1,\ldots,i_j}] \le \mu(W)^j + 2\alpha$. Hence, the overall probability is $Pr_{y\in_R\{0,1\}^r, h_1,\ldots,h_m \in_R H}[y \in B^{h_1,\ldots,h_m}_{i_1,\ldots,i_j}] \le Pr[h_1,\ldots,h_m\ is\ not\ a\ good\ sequence] + \mu(W)^j + 2\alpha$. A careful examination of the above induction proof reveals that we required h_i to be (α,U,V)-good for 2^{m-i} pairs of subsets (U,V). By the Hash Mixing Lemma (lemma 2.3.2), the probability of a random h_i not to be (α,U,V)-good for a specific pair (U,V) is at most $\frac{\mu(U)\mu(V)(1-\mu(V))}{2^r\alpha^2} \le \alpha$. Hence, the probability it is not (α,U,V)-good for a set of 2^{m-i} pairs is at most $2^{m-i}\alpha$. Summing up over all i's we obtain that the probability of $h_1,\ldots,h_m$ not to be a good sequence is at most $(2^m-1)\alpha = (k-1)\alpha$. Hence, the total probability is at most $\mu(W)^j + (k+1)\alpha$. □

5.6 The Ajtai-Komlós-Szemerédi Method

Specification

1. $V_1 = G \times \{1,\ldots,d\}^k$. G is a logspace explicit $(2^r,d,\lambda)$ expander, for which $\frac{\lambda}{d} < \frac{\beta^2}{4} - \sqrt{\epsilon}$ and d is constant (recall corollary 2.2.1). $2\sqrt[4]{\epsilon} < \beta < 1$ is an arbitrary constant. Each node in V_1 is denoted by a sequence $(y,j_1,\ldots,j_k)$ where y is a node in G and $j_1,\ldots,j_k$ are indices of neighbors in G. This sequence defines a *walk* of length k on G that starts at y. We denote the nodes that occur at this walk by $y_0,y_1,\ldots,y_k$, where $y_0 = y$ and y_i is the j_i^{th} neighbor of y_{i-1} for $i=1,\ldots,k$.

2. $N(y,j_1,\ldots,j_k) = (y_1,\ldots,y_k)$.

Theorem 5.6.1 (Ajtai-Komlós-Szemerédi) *For any $\epsilon < \frac{1}{16}$ and for any $2\sqrt[4]{\epsilon} < \beta < 1$ the AKS construction is an $(r+O(k),r,k,\beta^k,\epsilon)$ weak extractor.*

Proof: We choose $y \in_R G$ and $j_1, \ldots, j_k \in_R \{1, \ldots, d\}$ at random and bound the probability $Pr[MAJ(f_W(y_1), \ldots, f_W(y_k)) = 1]$. This probability equals to $Pr[For\ at\ least\ \frac{k}{2}\ of\ y_1, \ldots, y_k : y_i \in W] \leq \sum_{1 \leq i_1 < \ldots < i_{\frac{k}{2}} \leq k} Pr[\bigwedge_{j=1}^{\frac{k}{2}} y_{i_j} \in W]$.

Fix some $1 \leq i_1 < \ldots < i_{\frac{k}{2}} \leq k$. We will prove that $Pr[\bigwedge_{j=1}^{\frac{k}{2}} y_{i_j} \in W] \leq (\frac{\lambda}{d} + \sqrt{\epsilon})^{\frac{k}{2}}$. Summing up over all possibilities for $i_1, \ldots, i_{\frac{k}{2}}$ we obtain $Pr[MAJ(f_W(y_1), \ldots, f_W(y_k)) = 1] < \binom{k}{\frac{k}{2}}(\frac{\lambda}{d} + \sqrt{\epsilon})^{\frac{k}{2}} < 2^k(\frac{\lambda}{d} + \sqrt{\epsilon})^{\frac{k}{2}} \leq \beta^k$.

Denote by M the adjacency matrix of G and by P the double-stochastic matrix $\frac{1}{d}M$. Let C be a $2^r \times 2^r$ binary matrix which has the characteristic function of W on its diagonal and zero everywhere else. Let C_0 be a stochastic vector of length 2^r whose entries are all $\frac{1}{2^r}$. C_0 specifies the distribution of y (since it is chosen uniformly at random from $\{0,1\}^r$). We define:

$$C_i = \begin{cases} C_{i-1} \cdot P \cdot C & i = i_j\ for\ some\ 1 \leq j \leq \frac{k}{2} \\ C_{i-1} \cdot P & Otherwise \end{cases}$$

The m^{th} entry in C_i contains $Pr[y_i = m \wedge y_{i_1}, \ldots y_{i_t} \in W]$, where t is the maximal for which $i_t < i$. If $i = i_j$ for some $1 \leq j \leq \frac{k}{2}$ then we project this vector on the coordinates that correspond to W. $\|C_i\|_1$ is the probability that $y_{i_1}, \ldots, y_{i_s}$ belong all to W, where s is the maximal for which $i_s \leq i$. Therefore, $\|C_k\|_1$ is exactly the probability we wish to bound. That is we need to show that $\|C_k\|_1 \leq (\frac{\lambda}{d} + \sqrt{\epsilon})^{\frac{k}{2}}$.

Let $\lambda_1, \ldots, \lambda_{2^r}$ be the eigenvalues of P such that $|\lambda_i| \geq |\lambda_{i+1}|$ for $i = 1, \ldots, k-1$. Notice that $\lambda_1 = 1$, since the largest eigenvalue of the adjacency matrix of a d-regular graph is always d. This together with the fact P is stochastic implies that $(1, 1, \ldots, 1)$ is an eigenvector of P that corresponds to λ_1.

Let $v_1, \ldots, v_{2^r}$ be a basis of orthogonal eigenvectors. Without loss of generality, we can choose v_1 to be $(1, 1, \ldots, 1)$.

By the Cauchy-Schwartz inequality we have $\|C_k\|_1 \leq \sqrt{2^r}\|C_k\|_2$. Now, $\|C_k\|_2 = \|C_0 \cdot (\prod_{j=1}^{\frac{k}{2}} P^{i_j - i_{j-1}} \cdot C) \cdot P^{k - i_{\frac{k}{2}}}\|_2$ where $i_0 = 0$. This is $\leq \|C_0\|_2\|P\|_2^k\|C\|_2^{\frac{k}{2}} = \sqrt{\frac{1}{2^r}}\|P \cdot C\|_2^{\frac{k}{2}}\|P\|_2^{\frac{k}{2}}$. We conclude that $\|C_k\|_1 \leq \|P \cdot C\|_2^{\frac{k}{2}}\|P\|_2^{\frac{k}{2}}$. Since the largest eigenvalue of P is 1, it does not increase vector lengths. This means that $\|P\|_2 \leq 1$, implying that $\|C_k\|_1 \leq \|P \cdot C\|_2^{\frac{k}{2}}$.

Now, $\|P \cdot C\|_2 = \|C \cdot P\|_2 = \max_{\|v\|_2 = 1} \|C \cdot P \cdot v\|_2$. Take an arbitrary $v \in R^{2^r}$ such that $\|v\|_2 = 1$. We represent v as a linear combination of the basis vectors $v = \sum_{i=1}^{2^r} \alpha_i v_i$.

$\|C \cdot P \cdot v\|_2 = \|\sum_{i=1}^{2^r} \alpha_i C \cdot P \cdot v_i\|_2 = \|\sum_{i=1}^{2^r} \alpha_i C \lambda_i v_i\|_2$. By the triangle inequality this is $\leq \|\alpha_1 \lambda_1 C v_1\|_2 + \|C \cdot (\sum_{i=2}^{2^r} \lambda_i \alpha_i v_i)\|_2$.

Recall that $v_1 = (1, 1, \ldots, 1)$. Therefore, $C \cdot v_1$ is a characteristic vector of W. We get that $\|C \cdot v_1\|_2 = \sqrt{|W|}$ whereas $\|v_1\|_2 = \sqrt{2^r}$. Hence, $\|C \cdot v_1\|_2 = \sqrt{\frac{|W|}{2^r}}\|v_1\|_2 < \sqrt{\epsilon}\|v_1\|_2$. Notice also that C may only decrease vector norms. Therefore $\|C \cdot (\sum_{i=2}^{2^r} \lambda_i \alpha_i v_i)\|_2 \leq \|\sum_{i=2}^{2^r} \lambda_i \alpha_i v_i\|_2$. We conclude that $\|C \cdot P \cdot v\|_2 < \sqrt{\epsilon}\|\alpha_1 v_1\|_2 + \|\sum_{i=2}^{2^r} \lambda_i \alpha_i v_i\|_2$.

Since $v_2, \ldots, v_{2^r}$ are orthogonal then $\|\sum_{i=2}^{2^r} \lambda_i \alpha_i v_i\|_2^2 = \sum_{i=2}^{2^r} \|\lambda_i \alpha_i v_i\|_2{}^2$. $|\lambda_2| \geq |\lambda_i|$ for each $i = 2, \ldots, 2^r$. Therefore, the last sum $\leq |\lambda_2|^2 \sum_{i=2}^{2^r} \|\alpha_i v_i\|_2{}^2 = |\lambda_2|^2 \|\sum_{i=2}^{2^r} \alpha_i v_i\|_2^2$. Hence, $\|C \cdot P \cdot v\|_2 < \sqrt{\epsilon}\|\alpha_1 v_1\|_2 + |\lambda_2| \|\sum_{i=2}^{2^r} \alpha_i v_i\|_2$.

Notice that $\|v\|_2^2 = \|\alpha_1 v_1 + \sum_{i=2}^{2^r} \alpha_i v_i\|_2^2 = \|\alpha_1 v_1\|_2^2 + \|\sum_{i=2}^{2^r} \alpha_i v_i\|_2^2$. Therefore, $\|\alpha_1 v_1\|_2, \|\sum_{i=2}^{2^r} \alpha_i v_i\|_2 \leq \|v\|_2 = 1$. Hence, $\|C \cdot P \cdot v\|_2 < \sqrt{\epsilon} + |\lambda_2| = \sqrt{\epsilon} + \frac{\lambda}{d}$.

We conclude that $\|C_k\|_1 \leq \|C \cdot P\|_2^{\frac{k}{2}} < (\sqrt{\epsilon} + \frac{\lambda}{d})^{\frac{k}{2}}$. □

Chapter 6

Two Way Random Tape Amplification

The amplification model we presented in chapter 4 deals with Monte-Carlo algorithms that have a *one way random tape*. In this chapter we enhance our discussion to amplification of randomized algorithms that have a *two way random tape* (recall section 3.4).

6.1 Two Way Black Box Amplification

As in the one way case, we are interested in *black box* amplification methods. We can generalize the black box amplifier notion introduced in section 4.1 to support amplification of two way randomized algorithms as well. In the one way case the plugged algorithm requests the random bits in a sequential order: the first random bit, the second, an so on until the r^{th} random bit. We now let the plugged algorithm request the random bits in an arbitrary order, and even to ask for the same bit several times. The sequential read once order of the one way algorithms is only a special case of this one.

The implementation of the additional feature is quite simple: we add another *address tape* on which the plugged algorithm writes the index of the random bit it needs. The amplifier supplies this bit when it is available.

Note that now the execution of the amplifier is not the same for all plugged algorithms, since each plugged algorithm may have a different request sequence. We will be interested mainly in *oblivious black box amplifiers*. These are amplifiers that generate the same sequences of pseudo-random bits, no matter at which order these sequences are queried. Hence, in such amplifiers the pseudo-random sequences depend only on r and on the random string of the amplifier. Note that in the one way case all black box amplifiers are oblivious.

When the plugged algorithm has a two way random tape, it is only fair (and reasonable too, due to applicability considerations) that the amplifier itself will have a two way random tape. We call such amplifiers *2W black box amplifiers*.

6.2 Two Way Explicit Weak Extractors

Oblivious black box amplifier maintain the useful connection to weak extractors (recall section 4.2.1). $(l, r, k, \delta, \epsilon)$ oblivious amplifiers are equivalent to $(l, r, k, \delta, \epsilon)$ weak extractors. This equivalence does not hold for non-oblivious amplifiers.

In this respect we are to present a new definition for *bipartite graph explicitness*. We use the same definition as in the one way case (definition 4.6.6) with two modifications:

1. The neighborhood algorithm has an ordinary two way input tape.

2. For each neighbor there is an additional one way input tape, that contains a sequence of indices in the range $1, \ldots, r$. This sequence indicates to the algorithm in what order to output the bits of the corresponding neighbor. We denote by Q the maximal length of such a sequence. Q might grow as a function of r.

We call such an algorithm a *2W neighborhood algorithm*, as opposed to the *1W neighborhood algorithms* we used so far. We call bipartite graphs that have 2W neighborhood algorithms *2W explicit*. It turns out that the efficiency of oblivious 2W amplifiers is equivalent to the 2W explicitness of the corresponding weak extractors (compare with the equivalence theorem, theorem 4.6.1).

The next proposition outlines the connection between 1W explicitness and 2W explicitness:

Proposition 6.2.1 *Let E be any (l, r, k) bipartite graph, that has a p-parallel 1W neighborhood algorithm N_1 which runs in T time and S space. Let $q_1, \ldots, q_k$ be any collection of query sequences of length at most Q. Then, E has a 1-parallel 2W neighborhood algorithm N_2 that outputs the bits of the i^{th} neighbor according to q_i and runs in $O(kTQ)$ time and $O(S)$ space.*

Proof: N_2 computes the neighbors one at a time. It invokes N_1 for each output bit it needs to compute. When running N_1, N_2 ignores all the output bits generated by N_1 except for the one it needs in the current invocation. □

6.3 $2W - BPL$ Amplification

$2W - BPL$ is the class of languages accepted by logspace polynomial time algorithms, that have a two way random tape.

The amplifier applicability notion (definition 4.4.1) is relevant in the context of two way amplification too. Naturally, amplifiers which are applicable for two way classes may have a two way random tape too. The next proposition presents the characterization of $2W - BPL$ applicable amplifiers (compare with proposition 4.5.2 for BPL):

Proposition 6.3.1 *A p-parallel $(l, r, k, \delta, \epsilon)$ 2W oblivious amplifier that uses T private time and S private space is $2W - BPL$ applicable iff p is constant, T is polynomial in r and S is logarithmic in r.*

Since $2W - BPL$ algorithms may have at most a polynomial number of requests for random bits, proposition 6.2.1 yields a way to produce $2W - BPL$ applicable amplifiers from BPL applicable amplifiers.

None of the weak extractors presented in chapter 5 (except for the naive one) can be used trivially for BPL amplification. It turns out that for $2W - BPL$ amplification they are completely adequate!

The multiple access to the random seed required at the straightforward implementations of the 1W neighborhood algorithms of these weak extractors compels them to store the seed on the work tape. This seed is the main ingredient (($\Omega(r)$ bits) in the space consumption of the algorithms. When a two way random tape is available, the algorithms do not longer need to store the seed on the work tape. They can multiply access it on the random tape. The rest of the space used by the algorithms (usually for computing universal hash functions or for computing neighborhoods in expanders) is logarithmic. Hence, 2W neighborhood algorithms for these weak extractors run in a logarithmic space and yield $2W - BPL$ applicable amplifiers.

6.4 Amplification of One Way Algorithms Using 2W Amplifiers

We go back to amplification of ordinary Monte-Carlo algorithms, that have a one way random tape, but consider two augmentations to the amplifier power:

1. We allow the amplifier to have a two way random tape.
2. We allow the amplifier to access the simulation tapes even at computation steps. This means that pseudo-random bits it supplies for the simulations are affected also by the configurations of these simulations.

It turns out that these two augmentations add an interesting power to the amplifier. Nisan proved in [Nis93] that such amplifiers can reduce the error of BPL algorithm to 0!! The lower bound of Radhakrishnan and Ta-Shma (recall theorem 4.3.1) implies this cannot be done with the standard amplifiers. This beautiful result can be phrased as follows:

Theorem 6.4.1 (Nisan) $BPL \subseteq 2W - ZPL$.

For the proof of this theorem we need several preliminary definitions and lemmas.

Let A be an (r, ϵ) Monte-Carlo algorithm that runs in logspace and a polynomial time and uses a one way random tape. A computes a language $L \in BPL$. Fix an input x for A of length n, and denote by A^x the computation of A on x. Consider the configuration graph of A^x (recall this notion from section 4.1). Any random string $y \in \{0,1\}^r$ induces a single path on this graph from the starting configuration to either an accepting or a rejecting configuration. This is done by simply leaving in the graph only the edges that correspond to the bits of y.

Since A uses a logarithmic space, the configuration graph is of at most a polynomial size n^c. The longest path in the graph is bounded by T, the running time of A. Obviously, T is also polynomial in n, hence $T = n^d$. We assume, WLOG, that A reads a bit from its random tape at each step of the execution. If it does not need it, it simply ignores it. Hence, $r = T$.

Denote by M the augmented black box amplifier, by W the set of witnesses on which A accepts x, and by $\mu_R(S) = \frac{|S|}{|R|}$ the proportional volume of a subset $S \subseteq R$. M uses a polynomial size discrepancy set in $\{0,1\}^r$ to approximate $\mu_{\{0,1\}^r}(W)$ within a constant factor $\alpha < \frac{1}{2} - \epsilon$. M accepts the input if and only if the result of this approximation is greater than $\frac{1}{2}2^r$.

Definition 6.4.1 *Let $S, W \subseteq R$. The* **discrepancy** *of S with respect to W is* $D(S, W) = |\mu_S(W \cap S) - \mu_R(W)|$.

Definition 6.4.2 *Let $S, W \subseteq R$. S is called an* **α-representative** *for W if $D(S, W) \leq \alpha$.*

Think of R as $\{0,1\}^r$ and of W as the set of accepting witnesses for x. If M succeeds to find a polynomial size S that is α-representative for W, then it can approximate $\mu_R(W)$ efficiently. It turns out that a random S will do. The problem is how to *verify* that such an S is α-representative.

Definition 6.4.3 *Let v be a configuration of A^x and $i \in \{0, 1, \ldots, r\}$. $W_{v,i}$ is the set of random strings in $\{0,1\}^r$ that induce a path, that has v on its i^{th} position. We divide this set to two halves: $W^0_{v,i}$ are those that have 0 on their $(i+1)^{st}$ position and $W^1_{v,i}$ have there 1.*

Definition 6.4.4 *A set $S \subseteq \{0,1\}^r$ is called* **α-good**, *if for every $i = 0, 1, \ldots, r-1$ and for every configuration v:*

$$|\mu_S(W^1_{v,i}) - \mu_S(W^0_{v,i})| \leq \alpha$$

The next lemma shows that a random choice of a polynomial sized S is likely to be α-good.

Lemma 6.4.1 *Let $\alpha = n^{-a}$ and let $b = c + d + 2a$. Choose n^b independent random elements from $\{0,1\}^r$ and call the resulting set S (which has size of at most n^b). Then, $Pr[S\ is\ not\ \alpha - good] \leq \frac{1}{2}$.*

Proof: We will be interested only in sets S that are of size exactly n^b. Therefore, we first bound the probability S has a smaller size.

Denote by $x_1, \ldots, x_{n^b}$ the random choices from $\{0,1\}^r$. Assume S has size $< n^b$. Then there is at least one pair $i \neq j$ such that $x_i = x_j$. Therefore,

$$Pr[|S| < n^b] \leq \sum_{1 \leq i < j \leq n^b} Pr[x_i = x_j] = \binom{n^b}{2} \frac{1}{2^r} \leq \frac{n^{2b}}{2^r} \leq \frac{1}{4}$$

.

Assume now that $|S| = n^b$ and fix a configuration v and $i \in \{0, 1, \ldots, r-1\}$. It holds that:

$$Pr[|\mu_S(W^1_{v,i}) - \mu_S(W^0_{v,i})| > \alpha] = Pr[|\frac{|W^1_{v,i} \cap S|}{|S|} - \frac{|W^0_{v,i} \cap S|}{|S|}| > \alpha] = Pr[||W^1_{v,i} \cap S| - |W^0_{v,i} \cap S|| > \alpha |S|]$$

Consider any subset $W \subseteq \{0,1\}^r$. Define for each random choice x_i the following random variable:

$$X_i = \begin{cases} 1 & x_i \in W \\ 0 & x_i \notin W \end{cases}$$

Since X_i is an indicator random variable, $E[X_i] = Pr[X_i = 1] = \frac{|W|}{2^r}$. Let X be the number of choices that fall in W. On one hand $X = |W \cap S|$, and on the other hand $X = \sum_{i=1}^{n^b} X_i$. The linearity of the expectation implies that $E[|W \cap S|] = \frac{|S||W|}{2^r}$. Applying this to $W^0_{v,i}$ and to $W^1_{v,i}$, while noting that $|W^1_{v,i}| = |W^0_{v,i}| = \frac{|W_{v,i}|}{2}$, yields $E[|W^0_{v,i} \cap S|] = E[|W^1_{v,i} \cap S|] = \frac{|W_{v,i}||S|}{2^{r+1}}$. Therefore,

$$Pr[||W^1_{v,i} \cap S| - |W^0_{v,i} \cap S|| > \alpha |S|] =$$

$$Pr[||W^1_{v,i} \cap S| - \frac{|W_{v,i}||S|}{2^{r+1}} + \frac{|W_{v,i}||S|}{2^{r+1}} - |W^0_{v,i} \cap S|| > \alpha |S|] \leq$$

$$Pr[||W^1_{v,i} \cap S| - \frac{|W_{v,i}||S|}{2^{r+1}}| + |\frac{|W_{v,i}||S|}{2^{r+1}} - |W^0_{v,i} \cap S|| > \alpha |S|] \leq$$

$$Pr[||W^1_{v,i} \cap S| - \frac{|W_{v,i}||S|}{2^{r+1}}| > \frac{\alpha |S|}{2}] + Pr[||W^0_{v,i} \cap S| - \frac{|W_{v,i}||S|}{2^{r+1}}| > \frac{\alpha |S|}{2}] =$$

$$2 \cdot Pr[||W^1_{v,i} \cap S| - E[|W^1_{v,i} \cap S|]| > \frac{\alpha |S|}{2}] =$$

$$2 \cdot Pr[||W^1_{v,i} \cap S| - E[|W^1_{v,i} \cap S|]| > \beta E[W^1_{v,i}]]$$

where $\beta = \frac{\alpha 2^{r-1}}{|W_{v,i}|}$. Using Chernoff bounds yields:

$$Pr[||W^1_{v,i} \cap S| - E[|W^1_{v,i} \cap S|]| > \beta E[W^1_{v,i}]] \leq e^{-\frac{\beta^2 E[|W^1_{v,i} \cap S|]}{6}} = e^{-\frac{\alpha^2 |S| 2^r}{48 |W_{v,i}|}}$$

Since $|W_{v,i}| \leq 2^r$, then:

$$e^{-\frac{\alpha^2 |S| 2^r}{48 |W_{v,i}|}} \leq e^{-\frac{\alpha^2 |S|}{48}} = e^{-\frac{n^{c+d}}{48}} \leq \frac{1}{8n^{c+d}}$$

We conclude that:

$$Pr[|\mu_S(W^1_{v,i}) - \mu_S(W^0_{v,i})| > \alpha] \leq \frac{1}{4n^{c+d}}$$

Summing up over all possibilities of v and i we obtain:

$$Pr[S \text{ is not } \alpha - good \mid |S| = n^b] \leq \frac{1}{4}$$

Hence,

$$Pr[S\ is\ not\ \alpha - good] \leq Pr[S\ is\ not\ \alpha - good \mid |S| = n^b] + Pr[|S| < n^b] \leq \frac{1}{4} + \frac{1}{4} = \frac{1}{2}$$

□

We prove now that it is possible to check whether a given set S is α-good in logspace:

Lemma 6.4.2 *There is a logspace algorithm N that on input (S, α), where $S \subseteq \{0,1\}^r$ and α is a rational number, outputs "Yes" iff S is α-good.*

Proof: N works as follows:

```
for each configuration v
    for each i = 0,1,...,r
        Initialize W^0_{v,i} and W^1_{v,i} to 0
        for each s ∈ S
            Run A^x on the random string s
            If A^x reaches v at the i^th step add 1 to
            either W^0_{v,i} or to W^1_{v,i}, depending on s_{i+1}.

        If |W^1_{v,i} - W^0_{v,i}| > α|S|, return ``no''
return ``yes''
```

N's input length is $O(r|S|)$. It has to keep in its space the following:

1. The counters v and i ($O(\log n)$ bits, since the configuration graph is of polynomial size.
2. The counter that points to s ($O(\log(r|S|))$ bits).
3. The current configuration of A^x ($O(\log n)$ bits).
4. The accumulators $W^0_{v,i}$ and $W^1_{v,i}$ ($O(\log |S|)$ bits).

Since r and $|S|$ are polynomial in n, the total space used by N is logarithmic in its input length. □

Lemma 6.4.3 *If S is α-good, then it is $(\alpha r n^c)$-representative for the set of random strings on which A^x accepts.*

This lemma follows from a stronger lemma we can prove:

Lemma 6.4.4 *Let S be α-good. Then, for all $i = 0, 1, \ldots, r$ we have:*

$$\sum_v |\mu_S(W_{v,i}) - \mu_{\{0,1\}^r}(W_{v,i})| \leq n^c i \alpha$$

First we prove how lemma 6.4.3 follows from lemma 6.4.4.

Proof of lemma 6.4.3: Denote by J the set of all random strings that cause A^x to accept. Notice that J is the union of $W_{v,r}$ over all accepting configurations v. Therefore,

$$|\mu_S(J) - \mu_{\{0,1\}^r}(J)| \leq \sum_{accepting\ v} |\mu_S(W_{v,r}) - \mu_{\{0,1\}^r}(W_{v,r})|$$

By lemma 6.4.4 this value is at most $n^c r\alpha$. □

Proof of lemma 6.4.4: The proof goes by induction on i.

For $i = 0$, $W_{v,0} = \{0,1\}^r$ if and only if v is the starting configuration. For this v $\mu_S(W_{v,0}) = \mu_{\{0,1\}^r}(W_{v,0}) = 1$ and for all the others $\mu_S(W_{v,0}) = \mu_{\{0,1\}^r}(W_{v,0}) = 0$. Hence the sum is 0, which perfectly matches the lemma.

Assume the lemma is true for $i-1$. We show it holds also for i. Fix a configuration v. We define two sets: U_v^0 is the set of configurations that have an outgoing edge that reaches v and is labeled by 0. U_v^1 is defined similarly. We can present $W_{v,i}$ using these new definitions:

$$W_{v,i} = \bigcup_{u \in U_v^0} W_{u,i-1}^0 \cup \bigcup_{u \in U_v^1} W_{u,i-1}^1$$

Note that $W_{u,i-1}^0 \cap W_{u',i-1}^1 = \phi$ for any u, u' (even $u = u'$), since the strings of the two sets differ on their i^{th} coordinate. Note also that $W_{u,i-1}^0 \cap W_{u',i-1}^0 = \phi$ and that $W_{u,i-1}^1 \cap W_{u',i-1}^1 = \phi$ for any $u \neq u'$, since the same random string cannot lead to different configurations at a given step. It follows that the above union is a union of disjoint sets. This means that with respect to any set $W \subseteq \{0,1\}^r$ it holds:

$$\mu_W(W_{v,i}) = \sum_{u \in U_v^0} \mu_W(W_{u,i-1}^0) + \sum_{u \in U_v^1} \mu_W(W_{u,i-1}^1)$$

Since S is α-good, then for any u:

$$|\mu_S(W_{u,i-1}^0) - \frac{1}{2}\mu_S(W_{u,i-1})| =$$

$$|\mu_S(W_{u,i-1}^0) - \frac{1}{2}(\mu_S(W_{u,i-1}^0) + \mu_S(W_{u,i-1}^1))| =$$

$$\frac{1}{2}|\mu_S(W_{u,i-1}^0) - \mu_S(W_{u,i-1}^1)| \leq \frac{\alpha}{2}$$

Similarly,

$$|\mu_S(W_{u,i-1}^1) - \frac{1}{2}\mu_S(W_{u,i-1})| \leq \frac{\alpha}{2}$$

Since $W_{u,i-1}^0$ and $W_{u,i-1}^1$ partition $W_{u,i-1}$ to two halves, it holds that:

$$\mu_{\{0,1\}^r}(W_{u,i-1}^0) = \mu_{\{0,1\}^r}(W_{u,i-1}^1) = \frac{1}{2}\mu_{\{0,1\}^r}(W_{u,i-1})$$

Combining the above statements we obtain:

$$|\mu_S(W_{u,i-1}^0) - \mu_{\{0,1\}^r}(W_{u,i-1}^0)| \leq$$

$$|\mu_S(W_{u,i-1}^0) - \frac{1}{2}\mu_S(W_{u,i-1})| + |\frac{1}{2}\mu_S(W_{u,i-1}) - \frac{1}{2}\mu_{\{0,1\}^r}(W_{u,i-1})| \leq$$

$$\frac{\alpha}{2} + \frac{1}{2}|\mu_S(W_{u,i-1}) - \mu_{\{0,1\}^r}(W_{u,i-1})|$$

A similar statement can be obtained for $W_{u,i-1}^1$. We obtain for v the following:

$$|\mu_S(W_{v,i}) - \mu_{\{0,1\}^r}(W_{v,i})| \leq$$

$$\sum_{u\in U_v^0} |\mu_S(W^0_{u,i-1}) - \mu_{\{0,1\}^r}(W^0_{u,i-1})| + \sum_{u\in U_v^1} |\mu_S(W^1_{u,i-1}) - \mu_{\{0,1\}^r}(W^1_{u,i-1})| \leq$$

$$\frac{1}{2}\sum_{u\in U_v^0} (|\mu_S(W_{u,i-1}) - \mu_{\{0,1\}^r}(W_{u,i-1})| + \alpha) + \frac{1}{2}\sum_{u\in U_v^1} (|\mu_S(W_{u,i-1}) - \mu_{\{0,1\}^r}(W_{u,i-1})| + \alpha)$$

Note that the sets U_v^0 for all configurations v form a partition of the configuration set, since each configuration has exactly one outgoing edge labeled by 0. The same for U_v^1. Therefore, if we sum up the above expression over all configurations v we obtain:

$$\sum_v |\mu_S(W_{v,i}) - \mu_{\{0,1\}^r}(W_{v,i})| \leq$$

$$\sum_v (\frac{1}{2}\sum_{u\in U_v^0} (|\mu_S(W_{u,i-1}) - \mu_{\{0,1\}^r}(W_{u,i-1})| + \alpha) + \frac{1}{2}\sum_{u\in U_v^1} (|\mu_S(W_{u,i-1}) - \mu_{\{0,1\}^r}(W_{u,i-1})| + \alpha)) =$$

$$\sum_v |\mu_S(W_{v,i-1}) - \mu_{\{0,1\}^r}(W_{v,i-1})| + \alpha n^c$$

Applying the induction hypothesis we obtain what is wanted. □

Proof of theorem 6.4.1: The $2W - ZPL$ machine M simulates A^x as follows:

M selects a rational α such that $\alpha r n^c \leq \frac{1}{2} - \epsilon$. Then it chooses a random set $S \subseteq \{0,1\}^r$ of size n^b, by generating $r \cdot n^b$ bits on its two way random tape. First, M verifies that S has no duplicate elements. Then, it verifies S is α-good using the algorithm specified in lemma 6.4.2. If one of the verifications fails, M chooses a new random set S. It goes on until an α-good set of size n^b is found.

M iterates over the elements of S, while at each iteration it simulates A^x with the current element as a random bit string. M computes the proportional volume of the elements that lead A^x to acceptance, and accepts iff this value is at least $\frac{1}{2}$.

Denote by W the set of random strings, on which A^x accepts the input. By lemma 6.4.3 S is an $(\alpha r n^c)$-representative for this set. Hence, the computed proportional volume of W is at most at distance $\frac{1}{2} - \epsilon$ from its real proportional volume. If the input belongs to the language then proportional volume of W is at least $1 - \epsilon$ and therefore the value M computes is at least $\frac{1}{2}$. If the input does not belong to the language, then the volume is less than ϵ, and thus the computed value is less than $\frac{1}{2}$. This means, that M is always correct.

M uses its work tapes for the following duties:

1. Checking that S has no duplicate elements. This requires $O(\log(r|S|))$ bits, which is $O(\log n)$ since r and $|S|$ are polynomial in n.

2. Verifying that S is α-good. According to lemma 6.4.2 this requires $O(\log(r|S|))$ bits, which is again logarithmic in n.

3. Simulating A^x on the elements of S. This requires logarithmic space, since A^x belongs to BPL.

We conclude, that M runs in a logarithmic space. Note that M is not charged for the space needed to store the set S, since S is written on the two way random tape.

M runs in an expected polynomial time, since the expected number of tries until a good S of size n^b is found is less than 3. □

Chapter 7

Amplification of Space Bounded Algorithms

This thesis, based on the results presented in [BYGW98], initiates a first study of deterministic amplification of space bounded Monte-Carlo algorithms with a one way random tape.

Recall proposition 4.5.2 that specifies the necessary and sufficient conditions for an amplifier to be BPL applicable. The amplifier should be $(O(\log r), O(1))$ efficient.

All the amplification methods we know (refer to chapter 5), except for the naive one, use $\Omega(r)$ private space. The reason is that they need to multiply access their random seed for computing the k pseudo-random strings for the k simulations. The trivial solution is to store this seed on the work tape. Hence, all these methods are not BPL applicable.

Only the naive amplifier is $(O(\log r), 1)$ efficient, and thus is BPL applicable. However it uses a substantial number of random bits (kr). We seek for an amplifier which succeeds to keep both the number of random bits and the space small. Such an amplifier would be a first example for a random bit efficient BPL applicable amplifier.

In this chapter we present new implementations for the AKS and the KPS amplifiers, that are $(O(k), k)$ efficient. For a constant k these amplifiers are BPL applicable. Moreover, they can reduce the error probability to less than any given constant (by choosing an arbitrarily large k), and consume up to $r + O(k)$ random bits. Therefore, these amplifiers are the first non-trivial examples for BPL applicable amplifiers.

The implementations are based on designing new neighborhood algorithms for the AKS and the KPS weak extractors. These algorithms are $(O(k), k)$ efficient. They are based on a constant space online algorithm for computing neighborhoods in the expander of Margulis.

Note that what we achieve for BPL amplification is weaker than what is known for BPP amplification. The error can be reduced only to a constant and not beyond, where in BPP we got an exponential reduction of the error probability. We prove that BPL applicable black box amplifiers that use $O(r)$ random bits cannot reduce the error to less than a constant. This implies that our constructions are optimal with respect to black box amplifiers that use $O(r)$ random bits.

The lower bound is achieved by proving a lower bound on δ in dispersers. We show that in dispersers that have a small space neighborhood algorithm δ goes down with the parallelism of the algorithm and not with k (no matter how large is k). This means that if the parallelism is bounded to be a constant, δ cannot go down beyond a constant.

7.1 Results

This thesis has two main results. The first one shows that the AKS and the KPS weak extractors are $(O(k), k)$ explicit. These are the first constructions of small space online weak extractors.

Theorem 7.3.1 (Upper Bound) *1. Fix any $0 < \epsilon < \frac{1}{16}$. Then for all r and k the AKS construction[1] is an $(O(k), k)$ explicit $(r + O(k), r, k, \epsilon^{\Omega(k)}, \epsilon)$ weak extractor.*

2. For all r and k the KPS construction[2] is an $(O(k \log k), k)$ explicit $(r, r, k, O(\frac{1}{k}), \epsilon)$ weak extractor.

Using lemma 4.2.1 and the equivalence theorem (theorem 4.6.1) we obtain a first example for amplifiers which use both a small space and a small number of random bits:

Corollary 7.1.1 (Space Efficient Amplifiers)
1. Let $\epsilon < \frac{1}{16}$. Then for all r and k there exists an $(r + O(k), r, k, \epsilon^{\Omega(k)}, \epsilon)$ amplifier, which is $(O(k), k)$ efficient.
2. Let $\epsilon < \frac{1}{2}$. Then for all r and k there exists an $(r, r, k, O(\frac{1}{k}), \epsilon)$ amplifier, which is $(O(k \log k), k)$ efficient.

The first construction achieves a better error probability (exponential in k vs. linear in k), and the second uses less random bits (uses the optimal number of r bits vs. $r + O(k)$ random bits). However, there is no qualitative difference between the two constructions when k is bounded to be a constant.

The restriction on ϵ is not a significant obstacle, since if the original algorithm has a higher error probability, we can repeat it a constant number of times and take the majority vote of the answers. Using the amplifier with this repeated algorithm would change our estimations by a constant factor only.

Since these amplifiers uses only $O(k)$ space and run at most k parallel simulations, then for a constant k they are BPL applicable:

Corollary 7.1.2 *For any $\epsilon < \frac{1}{2}$ and any constant $0 < \delta < \epsilon$ there exists a BPL applicable $(r, r, O(1), \delta, \epsilon)$ amplifier.*

The main tool in the efficient weak extractors constructions is a constant space *online expander*. The main lemma proves that neighborhoods in the expander of Margulis ([Mar73]) can be computed in a constant space on an online Turing machine. This result is interesting for itself:

Lemma 7.2.1 (Constant Space Online Expanders) *For all $N = w^{2r}$, where w and r are natural numbers, there exists a constant space online expander graph on N vertices.*

The amplifiers we presented cannot achieve more than a constant reduction of the error probability for BPL algorithms, since this probability goes down with k, and k is bounded to be a constant due to the space limitations. Our second main result shows that these constructions are optimal with respect to BPL applicable black box amplifiers that use $O(r)$ random bits. This result derives from a lower bound on δ in dispersers that are small space explicit. This lower bound shows that in such dispersers δ goes down with the disperser parallelism and not with the degree k:

Theorem 7.4.1 (Lower Bound) *For all $k < 2^{\frac{r}{f}}$ and any $(\frac{r}{f}, p)$ explicit family of $(cr, r, k, \delta, \epsilon)$ dispersers it holds that $\delta \geq 2^{-O(q)}$, where $q = pcf$.*

Using lemma 4.2.1, proposition 4.2.1 and the equivalence theorem we obtain the following conclusion for space bounded amplification:

Corollary 7.1.3 *For all r, k and $0 < \epsilon < \frac{1}{2}$ and any BPL applicable $(O(r), r, k, \delta, \epsilon)$ black box amplifier it holds that $\delta \geq \epsilon^{O(1)}$.*

[1]Refer for the specification of this construction at section 5.6.
[2]Refer for the specification of this construction at section 5.3.

7.2 Online Graphs

In the new implementations of the AKS and the KPS amplifiers we need expander graphs which are easy to compute by space bounded algorithms that have a read-once access to their input tape. For this we introduce the next notion of *online graphs*. These are graphs in which neighborhoods can be computed on an online machine with a bounded space. An online graph of order q is one, on which walks of length q are computable by space bounded online machines.

Definition 7.2.1 *A family of d-regular multi-graphs $\{E_N\}$ on N vertices is called* ***S* space online of order *q***[3] *if the following hold:*

1. *There exists an encoding e_V for the vertex set of E_N.*

2. *There exists an encoding e_d for $\{1, \ldots, d\}$.*

3. *There is an S space q-output online machine R (called the* **order *q* neighborhood machine***), that on input $(e_d(j_1), \ldots, e_d(j_q), e_V(v))$, where $j_1, \ldots, j_q \in \{1, \ldots, d\}$ and v is a node in E_N, outputs $e_V(v_1), \ldots, e_V(v_q)$ on its q output tape (one node per tape), where $v_1, \ldots, v_q$ are the nodes in the walk on E_N that starts at v and is specified by $j_1, \ldots, j_q$.*[4]

7.2.1 Constant Space Online Expanders

The following is the main lemma of the upper bound. It proves that the expander of Margulis (recall section 2.2.2) is an online constant space graph:

Lemma 7.2.1 (Constant Space Online Expanders) *For $N = w^{2r}$, where w and r are natural numbers, $\{G_N\}$ is a family of constant space online graphs.*

Proof: For simplicity, we will sometimes use the following notation in the proof: $\overline{a} = a \bmod m$.

We are to construct an online Turing machine M, that gets as input (j, v), where $j \in \{1, \ldots, 8\}$ and v is a node in G_N and computes in a constant space the j^{th} neighbor of v. We should also present a method to encode the input and the output of the machine.

For simplicity, we assume that $w = 2$. In this case our Turing machine is able to use a binary alphabet. If w is greater than 2, we would have to use an alphabet of size w. This will change our estimations by a constant only.

Since the vertex set of G_N is $Z_m \times Z_m$ ($m = \sqrt{N}$), we can associate with v a pair (x, y) where $x, y \in Z_m$. We encode x and y in a standard binary encoding using r bits for each. To encode v we merge these two encoding sequences in an interleaving manner, i.e. $x_1, y_1, x_2, y_2, \ldots, x_r, y_r$, where x_1, y_1 are the least significant bits and x_r, y_r are the most significant bits. j is encoded in the standard binary encoding.

Denote the j^{th} neighbor of v by v' and the associated pair by (x', y'). By the definition of G_N there exist constants $a^{(x)}, b^{(x)}, c^{(x)}, a^{(y)}, b^{(y)}, c^{(y)} \in \{-1, 0, 1\}$ such that $x' = (a^{(x)}x + b^{(x)}y + c^{(x)}) \bmod m$ and $y' = (a^{(y)}x + b^{(y)}y + c^{(y)}) \bmod m$. Notice that these constants depend on the index j only. M holds in its finite control a table of size 8×6 that specifies for each index j its associated constants. Each table entry requires only two bits (one for the constant's absolute value and one for its sign). When M starts its execution it reads j from the input tape and determines the associated constants accordingly.

M computes its output ($v' = (x', y')$) by calculating the two expressions $(a^{(x)}x + b^{(x)}y + c^{(x)}) \bmod m$ and $y' = (a^{(y)}x + b^{(y)}y + c^{(y)}) \bmod m$. All the calculations are done modulo m, hence each value in the computation requires exactly r bits. M computes the two expressions in a bit by bit summation. At the i^{th} stage it computes x'_i and y'_i and writes them to the output tape.

[3] When a graph is online of order 1, we call it simply an *online graph.*

[4] $v_0 = v$ and v_i is the j_i^{th} neighbor of v_{i-1}.

We will show, for example, how M computes x'_i. The details of the description are somewhat cumbersome, however the intuition is very simple: we add three values $(a^{(x)}x, b^{(x)}y, c^{(x)})$ modulo m. For performing the summation we use the standard algorithm. Performing the modulo is straightforward: since m is a power of 2, we need to take the $r = \log m$ least significant bits of the result. You should pay attention to one slightly delicate detail: even though the constants $a^{(x)}, b^{(x)}, \ldots, c^{(y)}$ are kept in the finite control in a compact form (two bits each), M has to compute their r-bit representation. For example, $\overline{-1}$ is encoded as $11\ldots1$ (r times). For the positive constants it is simple: we pad them with zeroes on their left. For the negative constants we have to compute the difference between 0 and their absolute value modulo m. The computation takes care of these details together with the calculation of the above sum.

At the beginning of the i^{th} stage the reading head of M is located over x_i. M reads it and y_i that appears immediately after. It keeps the following data from previous computations:[5]

1. p - A two-bit carry from the computation of x'_{i-1}.
2. q_1, q_2 and q_3 - One-bit carries from the computations of $(\overline{a^{(x)}x})_{i-1}, (\overline{b^{(x)}y})_{i-1}$ and $\overline{c^{(x)}}_{i-1}$ respectively.

The computation of x'_i is done by following the next steps:

1. . Computing $(|a^{(x)}|x)_i$. It equals 0, if $a^{(x)} = 0$ and x_i otherwise.

2. Computing $(\overline{a^{(x)}x})_i$. If $a^{(x)} \geq 0$ then it is the same as the bit computed in step 1. Otherwise it is the i^{th} bit in the difference $(0 - |a^{(x)}|x) \bmod m$. Since m is a power of 2 the calculation steps are: $(\overline{a^{(x)}x})_i = (|a^{(x)}|x)_i \oplus q_1$ and $q_1 = q_1 \vee (|a^{(x)}|x)_i$.

3. Computing $(\overline{b^{(x)}y})_i$ and $\overline{c^{(x)}}_i$ in a similar way, using the carry bits q_2 and q_3 respectively.

4. Calculating x'_i as follows: denote by S the sum $(\overline{a^{(x)}x})_i + (\overline{b^{(x)}y})_i + \overline{c^{(x)}}_i + p$. Then, $x'_i = S \bmod 2$ and $p = \frac{1}{2}(S - x'_i)$. p is a 2-bit carry since we add here 3 values.

M keeps only 10 bits from stage to stage. Therefore, the total number of bits it keeps on its work tape during the whole execution is constant. □

7.2.2 Online Walk Lemma

The next lemma shows that computing walks of length q on online graphs can be done by only multiplying the space by a factor of q.

Lemma 7.2.2 *Let $\{E_N\}$ be a family of S space online graphs. Then, it is also $O(qS)$ space online of order q.*

Proof: The input of M, the order q neighborhood machine of $\{E_N\}$, is a string of the form $(e_d(j_1), \ldots, e_d(j_q), e_V(v))$, where $j_1, \ldots, j_q \in \{1, \ldots, d\}$, v is a node in E_N and e_d and e_V are the encodings used by R, the order 1 neighborhood machine of $\{E_N\}$.

M runs q copies of R denoted by $R_1, \ldots, R_q$. R_i is supposed to output v_i, the i^{th} node in the walk that starts at v and is specified by $j_1, \ldots, j_q$.

M begins by allocating $O(S)$ cells on its work tape for each R_i. The configuration of R at any given moment of its execution is determined by the contents of its work tape and by its local state (there is no need for the position of the reading head, since it is an online Turing machine).

Figure 7.1 illustrates the way $R_1, \ldots, R_q$ are executed my M. M lets $R_1, \ldots, R_q$ (in this order) run until they read all the bits of $j_1, \ldots, j_q$ from the input tape. The way $e_d(j_1), \ldots, e_d(j_q)$ are put on the input tape

[5] Before the first stage the data is initialized to 0.

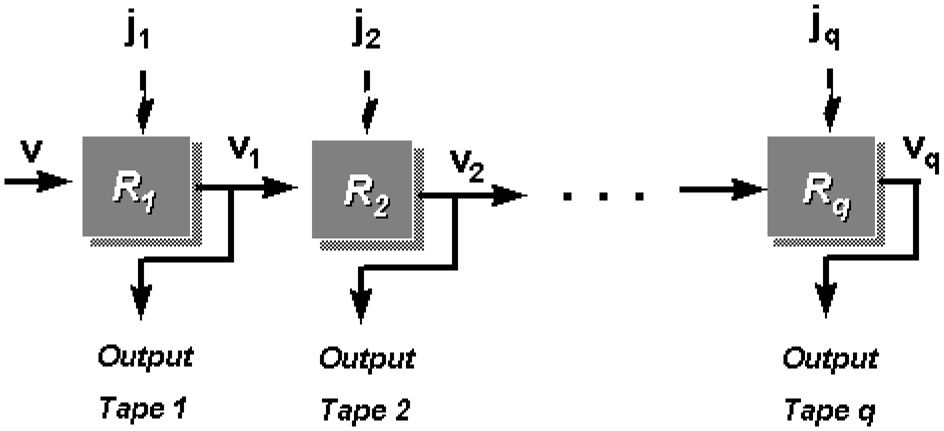

Figure 7.1: Online Walk Lemma

enables to do it serially. Notice that R cannot output any bit before it starts to read the bits of v. Therefore, $R_1, \ldots, R_q$ do not try to output anything at this stage.

M starts the second stage of its execution by running R_q until it demands an input bit. This bit should come from the output of R_{q-1}. Therefore, M suspends R_q and runs R_{q-1} until it requires an input bit. This goes on, until R_1 needs an input bit. R_1 can simply read its input bits from the input tape of M.

When R_i outputs a bit, its execution is suspended and this bit is used twice. First, it is written to the i^{th} output tape. Then, it is transferred to R_{i+1} as an input. The execution of R_{i+1} is resumed until it demands again an input bit or until it generates an output bit.

M keeps $O(S)$ bits on its work tape to store the configuration of each copy R_i, and another $O(\log(qS))$ bits for a pointer to its current position in the work tape. Therefore, the total storage requirements are $O(qS)$.
□

As a corollary, we obtain that neighborhoods in a q power of an online graph can be computed in the cost of a q factor only in the space.

Corollary 7.2.1 *Let $\{E_N\}$ be a family of S space online d-regular graphs. Then, $\{E_N^{(q)}\}$ is a family of $O(qS)$ space online d^q-regular graphs.*

Proof: Note that every neighbor index j in $E_N^{(q)}$ corresponds to a walk of length q $j_1, \ldots, j_q$ on E_N. Hence, the neighborhood machine of $E_N^{(q)}$ has to compute the last node at q length walks on E_N. It can use the order q neighborhood machine of E_N, by considering only its last output tape. By lemma 7.2.2 this machine uses only $O(qS)$ space. □

By taking an arbitrarily large constant power q of the Margulis expanders, we obtain the following useful result:

Corollary 7.2.2 *For every constant $\alpha > 0$ and $N = w^{2r}$ (where w, r are natural numbers) there exists a constant space online (N, d, λ) expander, for which d is constant and $\frac{\lambda}{d} < \alpha$.*

7.3 The Upper Bound

Theorem 7.3.1 (Upper Bound)
1. Fix any $0 < \epsilon < \frac{1}{16}$. Then for all r and k the AKS construction is an $(O(k), k)$ explicit $(r + O(k), r, k, \epsilon^{\Omega(k)}, \epsilon)$ weak extractor.

2. For all r and k the KPS construction is an $(O(k \log k), k)$ explicit $(r, r, k, O(\frac{1}{k}), \epsilon)$ weak extractor.

Proof: For specifications of the AKS and the KPS constructions refer to sections 5.6 and 5.3. For proofs they are weak extractors refer to theorems 5.6.1 and 5.3.1.

The original AKS and KPS constructions are based on an arbitrary polytime-explicit expander graph G, for which the degree is constant and $\frac{\lambda}{d}$ is sufficiently small. We will further require that G is a constant space online graph. Corollary 7.2.2 states that there exists such an expander.

We will present here neighborhood algorithms for these two weak extractors that run in $O(k)$ space and are k-parallel.

The AKS Neighborhood Algorithm

The neighborhood algorithm of the AKS weak extractor, N_{AKS}, has to compute walks of length k on G. Lemma 7.2.2 proves that this can be done on a k-output online machine, while increasing the space by a factor of k only. Hence, since G is a constant space online graph, N_{AKS} needs only $O(k)$ space. N_{AKS} uses all the k output tapes throughout the whole computation, implying it is k-parallel.

The KPS Neighborhood Algorithm

The neighborhood algorithm of the KPS weak extractor, N_{KPS}, computes in parallel the d^q neighbors of a node in $G^{(q)}$, where $q = \log_d(k)$. Note that the computations are independent, hence there is no problem in doing them simultaneously.

Computing neighborhoods in G can be done in a constant space. It follows, using corollary 7.2.1, the neighborhoods in $G^{(q)}$ can be computed in $O(q) = O(\log k)$ space. Since we do it in parallel for all the d^q neighbors, the space used by N_{KPS} is $O(k \log k)$. □

7.4 The Lower Bound

This section presents the lower bound we obtained for dispersers, that have $2^{O(r)}$ nodes on their left, and that have $o(r)$ space neighborhood algorithms. We show that in such dispersers δ depends only on the parallelism of the neighborhood algorithm, and not on k. It follows, that no matter how large is k, if the parallelism is small, then δ is large.

For the proof of the lower bound we introduce a new notion regarding computations on k-output Turing machines. This notion tries to capture the influence of the input on the output written on the i^{th} tape.

Definition 7.4.1 *Let M be an online k-output Turing machine, and let y be some input for it. Denote by t_j the time in which y_j, the j^{th} bit of y, is read from the input tape. We say that y_j is* **seen** *by the i^{th} output tape, if $t^y_{i,start} \leq t_j \leq t^y_{i,end}$. In other words, y_j is read during i's lifetime. We denote by l^y_i the number of input bits seen by the i^{th} output tape. We call i* **m-rich** *on input y, if $l^y_i \geq m$.*

The next lemma proves that if a disperser has a p-parallel neighborhood algorithm, then for each fixed input only $O(p)$ of the outputs are substantially affected by it.

Lemma 7.4.1 *Let E be a $(cr, r, k, \delta, \epsilon)$ disperser, whose neighborhood algorithm N_E is p-parallel. Then, for any input y of N_E there are at most pcf output tapes, which are $\frac{r}{f}$-rich.*

Proof: Since the machine is online, each input bit is read only once. Therefore, it is seen by at most p output tapes (i.e. at most p output tapes are alive when this bit is read). It follows that there are at most lp pairs of the form (i, y_j), where i is an output tape index and y_j is a bit seen by i during its lifetime. Since the i^{th} output tape contributes l_i^y such pairs, then there are at most pw tapes for which $l_i^y > \frac{l}{w}$. Substituting $l = cr$ and $w = cf$ yields what is stated in the lemma. □

The *richness* notion becomes crucial in determining the influence of the input on each of the output tapes when the machine can use only a small space. The lower bound proof uses this fact to prove that in "good" dispersers (ones that have $2^{O(r)}$ nodes on their left), which have only an $o(r)$ space neighborhood algorithm, δ depends on the parallelism of this algorithm.

Theorem 7.4.1 (Lower Bound) *For all $k < 2^{\frac{r}{f}}$ and any $(\frac{r}{f}, p)$ explicit family of $(cr, r, k, \delta, \epsilon)$ dispersers it holds that $\delta \geq 2^{-O(q)}$, where $q = pcf$.*

Proof: Let E be some disperser as in the theorem, and let N_E be its neighborhood algorithm.

Radhakrishnan and Ta-Shma show in [RTS97] (theorem 4.3.2) that for a constant $0 < \epsilon \leq \frac{1}{2}$ and for any r and l, if G is an $(l, r, k, \delta, \epsilon)$ disperser then necessarily $\delta \geq 2^{-O(k)}$. We will construct from E an $(cr, r, q, \delta, \frac{\epsilon}{2})$ disperser, implying that $\delta \geq 2^{-O(q)}$.

For each node $y \in V_1$, consider the execution of N_E on y. Denote by $I^y = \{i_1^y, \ldots, i_q^y\}$ the q richest output tapes in this execution, and by $J^y = \{j_1^y, \ldots, j_{k-q}^y\}$ the rest of the output tapes. Lemma 7.4.1 implies that the tapes in J^y are all at most $\frac{r}{f}$-rich.

For each $i \in \{1, \ldots, k\}$ denote by S_i the set of all y's for which $i \in J^y$. The string written on the i^{th} output tape is a deterministic function of the configuration of the machine, when it starts to write on this tape, and of the input string read during the tape's lifetime. Since N_E uses at most $\frac{r}{f}$ space, it has at most $2^{\frac{2r}{f}}$ configurations. On inputs in S_i the i^{th} tape sees input strings of length at most $\frac{r}{f}$ (i.e. $l_i^y \leq \frac{r}{f}$ for each $y \in S_i$). Hence, there are at most $2^{\frac{r}{f}+1}$ such strings. It follows, that on inputs in S_i the i^{th} tape can output less than $2^{\frac{4r}{f}}$ possible strings.

Denote by $B_i \subseteq V_2$ the set of nodes that are written on the i^{th} output tape when $y \in S_i$. Denote by B the union over all i: $B = \cup_{i=1}^{k} B_i$. The above observation means that $|B| \leq \sum_{i=1}^{k} |B_i| < \sum_{i=1}^{k} 2^{\frac{4r}{f}} < 2^{\frac{5r}{f}} < \frac{\epsilon}{2} 2^r$.

Color the edges of E by red and blue as follows: the edges between y and the neighbors written on the tapes in I^y are colored by red and the edges between y and the neighbors written on the tapes in J^y are colored by blue. We decompose E into two dispersers: E_1 contains all the red edges and E_2 all the blue edges. Note that all the edges in E_2 lead to nodes in B.

Every subset of V_1 of size at least $\delta 2^{cr}$ has at least $\epsilon 2^r$ neighbors in E. After dropping the edges of E_2, each such subset has at least $\frac{\epsilon}{2} 2^r$ neighbors. Therefore, E_1 is the $(cr, r, q, \delta, \frac{\epsilon}{2})$ disperser we looked for. □

Bibliography

[AES92] N. Alon, P. Erdős, and J.H. Spencer. *The Probabilistic Method.* John Wiley and Sons, 1992.

[AH87] L. Adleman and M. Huang. Recognizing primes in random polynomial time. In *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing*, pages 462–470, 1987.

[AKL$^+$79] R. Aleliunas, R.M. Karp, R.J. Lipton, L. Lovász, and C. Rackoff. Random walks, universal traversal sequences and the complexity of maze problems. In *Proceedings of the* 20^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[AKS87] M. Ajtai, J. Komlós, and E. Szemerédi. Deterministic simulation in logspace. In *Proceedings of the* 19^{th} *Annual ACM Symposium on Theory of Computing*, pages 132–140, 1987.

[Alo86a] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.

[Alo86b] N. Alon. Eigenvalues, geometric expanders, sorting in rounds and ramsey theory. *Combinatorica*, 6:207–219, 1986.

[ATSWZ97] R. Armoni, A. Ta-Shma, A. Wigderson, and S. Zhou. $SL \subseteq L^{\frac{4}{3}}$. In *Proceedings of the* 29^{th} *Annual ACM Symposium on Theory of Computing*, pages 230–239, 1997.

[BKV$^+$81] M. Blum, R.M. Karp, O. Vornberger, C.H. Papadimitriou, and M. Yannakakis. The complexity of testing whether a graph is a superconcentrator. *Information Processing Letters*, 13:164–167, 1981.

[Bol78] B. Bollobás. *Extremal Graph Theory*. Academic Press, London, 1978.

[BYGW98] Z. Bar-Yossef, O. Goldreich, and A. Wigderson. Deterministic amplification of space bounded probabilistic algorithms. Manuscript, 1998.

[CG89] B. Chor and O. Goldreich. On the power of two-point sampling. *J. of Complexity*, 5:96–106, 1989.

[CW79] J.L. Carter and M.N. Wegman. Universal classes of hash functions. *J. of Computer and System Sciences*, 18(2):143–154, 1979.

[CW89] A. Cohen and A. Wigderson. Dispersers, deterministic amplification and weak random sources. In *Proceedings of the* 30^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 14–19, 1989.

[GG81] O. Gabber and Z. Galil. Explicit constructions of linear-sized superconcentrators. *J. of Computer and System Sciences*, 22:407–420, 1981.

[Gol97] O. Goldreich. A sample of samplers – a computational perspective on sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, 020, 1997.

[ILL89] R. Impagliazzo, L. Levin, and M. Luby. Pseudo-random generation from one-way functions. In *Proceedings of the* 21^{st} *Annual ACM Symposium on Theory of Computing*, pages 12–24, 1989.

[IZ89] R. Impagliazzo and D. Zuckerman. How to recycle random bits. In *Proceedings of the* 30^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 248–253, 1989.

[JM87] S. Jimbo and A. Maruoka. Expanders obtained from affine transformations. *Combinatorica*, 7(4):343–355, 1987.

[KPS85] R.M. Karp, N. Pippenger, and M. Sipser. A time randomness tradeoff. In *AMS Conference on Probabilistic Computational Complexity*, 1985.

[LW95] M. Luby and A. Wigderson. Pairwise independence and derandomization. Lecture Notes, 1995.

[Mar73] G. A. Margulis. Explicit construction of concentrators. *Problemy Peredači Informacii*, 9(4):71–80, 1973. English translation in Problems Inform. Transmission (1975).

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12(4):449–461, 1992.

[Nis93] N. Nisan. On read-once vs. multiple access to randomness in logspace. *Theoretical Computer Science*, 107(1):135–144, 1993.

[Nis96] N. Nisan. Extracting randomness: How and why, a survey. In *Proceedings of the* 11^{th} *Annual IEEE Conference on Computational Complexity*, pages 44–58, 1996.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *J. of Computer and System Sciences*, 52(1):43–52, 1996.

[Pin73] M.S. Pinsker. On the complexity of a concentrator. In *Proceedings of the* 7^{th} *International Teletraffic Conference*, pages 318/1–318/4, 1973.

[Rab80] M. O. Rabin. Probabilistic algorithm for testing primality. *J. of Number Theory*, 12:128–138, 1980.

[RTS97] J. Radhakrishnan and A. Ta-Shma. Tight bounds for depth-two superconcentrators. In *Proceedings of the* 38^{th} *Annual IEEE Symposium on Foundations of Computer Science*, pages 585–594, 1997.

[Sch80] J.T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. of the ACM*, 27:710–717, 1980.

[Sip88] M. Sipser. Expanders, randomness or time versus space. *J. of Computer and System Sciences*, 36(3):379–383, 1988.

[WZ93] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound. In *Proceedings of the* 25^{th} *Annual ACM Symposium on Theory of Computing*, pages 245–251, 1993.

[Zuc93] D. Zuckerman. NP-complete problems have a version that's hard to approximate. In *Proceedings of the* 8^{th} *Annual IEEE Structure in Complexity Theory Conference*, pages 305–312, 1993.