# Deterministic Amplification of Space-Bounded Probabilistic Algorithms

Ziv Bar-Yossef
Department of Electrical Engineering
and Computer Science
U.C. Berkeley
Berkeley, CA 94720
zivi@cs.berkeley.edu

Oded Goldreich
Department of Computer Science
and Applied Mathematics
Weizmann Institute of Science
Rehovot 76100, Israel
oded@wisdom.weizmann.ac.il

Avi Wigderson
Institute of Computer Science
The Hebrew University of Jerusalem
Jerusalem 91904, Israel
avi@cs.huji.ac.il

## Abstract

*This paper initiates the study of deterministic amplification of space-bounded probabilistic algorithms. 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 Ajtai-Komlós-Szemerédi method, that enables to amplify an $S$-space algorithm that uses $r$ random bits and errs with probability $\epsilon$ 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 of the AKS method is based on 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.*

## 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 $\epsilon$ its error probability and by $r$ the number of random bits it uses. An $(l, r, k)$*-black-box simulator* is a meta algorithm, to which $A$ is plugged in as a subroutine.

The simulator runs $A$ $k$ times on its input $x$ and outputs the majority vote of the results.[1] The "heart" of the simulator 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 simulator uses a special combinatorial object, called *extractor*, which is a function that receives a random seed of length $l$ and produces $k$ pseudo-random strings of length $r$. The simulator is further called a $(\delta, \epsilon)$*-amplifier*, if it "fools" 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 [8] and Chor and Goldreich [2] 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 [6] succeeded to achieve an exponential reduction in the error probability (i.e., $2^{-\Omega(k)}$), but they require the use of additional $k^2$ random bits. If we insist on using an optimal number of random bits (i.e., $O(r)$), this results in only a subexponential amplification of $2^{-\Omega(\sqrt{r})}$. Nisan [10] 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 [3] and Impagliazzo and Zuckerman [6] independently noticed that a result of Ajtai, Komlós and Szemerédi [1] 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 in $k$.

[1] For some applications it is better to perform a different function on the $k$ results. Refer to [5] for examples.

## 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 paper 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 3.4) proves that, under a suitable encoding, the expander of Margulis [9] has this property. Computing a neighbor at this expander requires making a few summations modulo a power of 2. When choosing 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, 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

[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 $k$.

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 algorithm.

By choosing an arbitrarily large (but still constant) $k$ we obtain the next positive result:

**Theorem (BPL Amplification – Positive Result):** *For any constant $0 < \delta < \epsilon$, there exists an (explicitly given) $(r + O(k), r, k)$-black-box simulator, which is a $(\delta, \epsilon)$-amplifier and can be used for $BPL$ amplification.*

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 lower bound shows 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 (BPL Amplification – Negative Result):** *For any $(\delta, \epsilon)$-amplifier, that runs on an $(O(r), r, k)$-black-box simulator and that can be used for $BPL$ amplification, it holds that $\delta \geq \epsilon^{O(1)}$.*

Again, the technical result that implies this theorem is a combinatorial lower bound on the structure 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* [13] is closely related to deterministic amplification. $(l, r, k)$-*bipartite graphs* are ones with $2^l$ nodes on the left side, each with degree $k$, and $2^r$ nodes on the right side. Such graphs are called $(\delta, \epsilon)$-*extractors*, if 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 $\epsilon$-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 [12] for a survey about extractors.

Our main interest is in the connection of extractors to deterministic amplification. It turns out that black-box amplifiers are equivalent to *weak extractors*,[3], a slightly weaker notion than extractors. An $(l, r, k)$-bipartite graph is called a $(\delta, \epsilon)$-*weak extractor* if for any pair of sets $(U, W)$, where $U$ is a subset of the left side of size $\geq \delta 2^l$, and $W$ is a subset of the right side that contains at least $\frac{k}{2}$ neighbors of each $y \in U$, it holds that $|W| \geq \epsilon 2^r$. Every $(l, r, k)$-bipartite graph can be used to construct an $(l, r, k)$-black-box simulator and vice versa. Furthermore, the graph is a $(\delta, \epsilon)$-weak extractor iff the simulator is a $(\delta, \epsilon)$-amplifier. If an $(l, r, k)$-bipartite graph is a $(\delta, \epsilon)$-extractor it is also a $(\delta, \frac{1}{2} - \epsilon)$-weak extractor. This implies that extractors too are useful for deterministic amplification.

The complexity of computing neighborhoods in a weak extractor determines the efficiency of the resulting amplifier and its applicability to specific classes of randomized algorithms. For example, in order to use a weak 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 weak extractors, in which neighborhoods can be computed with a small space. The 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 a weak 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 machine is called $p$-*parallel*, if $p$ is the maximal number (over all inputs) of living output tapes at any given moment. An $(S, p)$-explicit *online weak 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,

[3] Also known as *majority dispersers*. Refer to [3].

then the amplifier is an $O(S + pS_A)$-space algorithm.

Therefore the next construction yields the positive result for amplifiers stated in the previous section:

**Theorem (Weak Extractors – Positive Result):** *For any constant $0 < \epsilon < \frac{1}{16}$ there exists an (explicitly given) $(O(k), k)$-explicit online $(r + O(k), r, k)$-bipartite graph, which is an $(\epsilon^{\Omega(k)}, \epsilon)$-weak extractor.*

The main tool in this construction is showing that, under a suitable encoding, 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 (Online Expanders):** *Fix any natural number $w \geq 2$. Then, for a varying $r$ there exists a constant-space online family of expanders on $w^{2r}$ vertices.*

On the other hand, a $p$-parallel $S$-space black-box amplifier results in an $(S, p)$-explicit weak extractor. The following lower bound shows that in small-space online $(\delta, \epsilon)$-weak extractors $\delta$ goes down with the parallelism $p$ and not with the degree $k$. This implies the negative result for amplifiers stated in the previous section.

**Theorem (Weak Extractors – Negative Result):** *For any constant $\epsilon$ and for any $(\frac{r}{4}, p)$-explicit online $(O(r), r, k)$-bipartite graph which is a $(\delta, \epsilon)$-weak extractor, it holds that $\delta \geq 2^{-O(p)}$.*

### 1.5 Conclusions

This paper presents an amplification method for space-bounded computations, which is proven to be optimal with respect to black-box methods. This does not mean that other methods, which may utilize properties of space-bounded randomized algorithms, cannot do any better. It is plausible that non black-box methods may achieve better than a constant reduction of the error probability.

Randomization in space-bounded computations seems to be better understood than in time-bounded computations: Specifically, utilizing the structure of space-bounded computations, unconditional results such as $BPL \subseteq SC$ (by Nisan [11]) and $BPL \subseteq L^{\frac{3}{2}}$ (by Saks and Zhou [15]) were obtained. It is thus plausible that, utilizing the structure of space-bounded computations, one may do better than black-box amplification.

On the other hand, if $BPL = L$ then the entire question (of amplification of BPL algorithms) is mute. Yet, trying to come-up with better amplification of BPL algorithms may be a fruitful avenue towards trying to prove that $BPL = L$.

### 1.6 Paper overview

Section 2, *Definitions*, presents the models we are working with: *Monte-Carlo algorithms*, *black-box amplifiers* and *weak extractors*. It outlines the close connection between the two latter notions.

Section 3, *Formal statement of results*, presents the list of results obtained in this paper.

Section 4, *The upper bound*, presents the construction of the $(O(k), k)$-explicit online weak extractor. It begins by a short review of *expander graphs*, proves that, under a suitable encoding, the expander of Margulis is a constant-space online expander and ends by presenting the mentioned above construction.

Section 5, *The lower bound*, proves the lower bound for small-space online weak extractors.

## 2 Definitions

This section sets the framework of our discussion throughout the paper. It begins by a short review of Monte-Carlo algorithms. It introduces the notion of *black-box amplifiers*, which is the model we consider for deterministic amplification. It presents *weak extractors* and their equivalence to black-box amplifiers. Finally, it discusses the applicability of amplifiers to amplification of specific classes and its relation to the complexity of neighborhood-computation in weak extractors.

During the discussion we use the convention that space is counted in binary, and accounts also for the machine state and the positions of the heads. Hence, the maximal number of configurations an $S$-space machine has is exactly $2^S$.

### 2.1 Black-box amplification of Monte-Carlo algorithms

**Definition 2.1** *An* **$(r, \epsilon)$-Monte-Carlo algorithm** *for a language $L$ (where $0 < \epsilon < \frac{1}{2}$ is some constant) is a randomized algorithm $A$ with a one-way random tape, that on inputs of length $n$ uses $r = r(n)$ random bits, and satisfies for all inputs $x$: $Pr_{y \in_R \{0,1\}^r}[A(x, y) \neq \chi_L(x)] < \epsilon$ (where $\chi_L$ is the characteristic function of $L$).*

This paper discusses two randomized complexity classes. $BPP$ is the class of languages computed by Monte-Carlo algorithms that run in a polynomial-time. $BPL$ is the class of logspace polynomial-time Monte-Carlo algorithms. We stress the *one-way* access of the algorithms to their random tape. Hence, they cannot recall previous random bits unless they explicitly store them in the work space.

The model for all amplification methods is *black-box simulation*. A *black-box simulator* $M$ is a generic algorithm, into which any Monte-Carlo algorithm $B$ can be plugged in, yielding an algorithm $M^B$. $M$ runs $k$ copies of $B$ and outputs the majority vote of the results. $M$ gets $r$, the number of random bits used by $B$, as input. It generates $k$ bit-strings, each of length $r$, and supplies them as random bits for the simulations of $B$. For the generation of these bit-strings $M$ uses $l = l(r)$ random bits.

$B$ already comes with its input. $B$ can be thought as the computation of a Monte-Carlo algorithm $A$ on an input $x$. Therefore, at any given configuration of $B$, only the random bit it reads determines to which of two possible configurations it should move.

$M$ is not an oracle machine with respect to $B$ itself, but rather to its *transition function*. The transition function, denoted by $f_B$, maps pairs of the form $(u, b)$, where $u$ is a configuration of $B$ and $b$ is a bit, to the configuration $v$, to which $B$ moves from $u$ after reading the random bit $b$.

For making sure $M$ uses the queries to $f_B$ to form legal executions of $B$, we impose several restrictions on these queries. First, $M$ is restricted to query $f_B$ with either an initial configuration or with a configuration returned by a previous call to $f_B$. Each query is labeled by a number. When $M$ queries $f_B$ with an initial configuration, it labels the query with a new number. When it queries $f_B$ with a configuration returned from a previous call to $f_B$, it labels the new query with the number of the query that returned this configuration. A sequence of queries labeled by the same number forms a single simulation of $B$. Hence, the number of query labels $M$ uses is the number of simulations it makes (i.e., $k$).

The steps, at which $M$ calls the oracle $f_B$, are called *simulation steps*. The rest are called *computation steps*. We further postulate that during the computation steps $M$ has no access to any of the configurations of $B$. This means that the computation steps of $M$ are not affected by the choice of the algorithm $B$, because all it sees from every algorithm $B$ we plug in is a sequence of $r$ requests for random bits. Hence, fixing $r$ and the random string $y \in \{0,1\}^{l(r)}$ uniquely determines the computation steps of $M$.

**Definition 2.2** *An* $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{k})$**-black-box simulator** *(where $k = k(r)$ and $l = l(r)$) is an algorithm $M$ as specified above. $M$ is further called a* $(\boldsymbol{\delta}, \boldsymbol{\epsilon})$**-amplifier** *(where $\delta = \delta(\epsilon, r)$), if for any $(r, \epsilon)$-Monte-Carlo algorithm $B$ the machine $M^{f_B}$ is an $(l, \delta)$-Monte-Carlo algorithm. In the sequel, we use $M^B$ as shorthand for $M^{f_B}$.*

Without loss of generality, we assume that $M$ stores the last configuration reached in each simulation (which is the configuration returned by the last query labeled by the simulation number). The reason is that if $M$ manages to store less bits than those required for holding the configuration (for every configuration), and then succeeds to restore the configuration before the next call to $f_B$, then there is some shorter description of the configurations of $B$. If the encoding of $B$'s configurations is chosen to be the most efficient one, then this is not possible.

We call the space used by $M$ during its computation steps the *private storage*. The space used for holding the configurations of the currently executed simulations is called the *simulation storage*.

**Definition 2.3** *A simulation made by a black-box simulator $M$ is called* **active** *at time $t$, if $M$ holds an intermediate (not accepting or rejecting) configuration of this simulation at time $t$. $M$ is called* $\boldsymbol{p}$**-parallel**, *if $p$ is the maximal number (over all inputs and all time steps) of active simulations $M$ has.*

If the private storage of a black-box simulator $M$ is $S$, it is $p$-parallel and the space used by $B$ is $S_B$, then $M^B$ uses $O(S + pS_B)$ space. Hence, the parallelism of a simulator is an important factor in estimating its space requirements.

**Definition 2.4** *A simulator $M$ is called* $(\boldsymbol{S}, \boldsymbol{p})$**-efficient**, *if it is $p$-parallel and uses $S$ private space.*

The next definition determines when a simulator can be used for amplification of algorithms of a specific class:

**Definition 2.5** *Let $\mathcal{C}$ be some randomized complexity class. A simulator $M$ is called* $\boldsymbol{\mathcal{C}}$**-applicable** *if for any algorithm $B \in \mathcal{C}$ it holds that $M^B \in \mathcal{C}$.*

We are interested in $BPL$-applicable amplifiers. It is easy to see that such amplifiers should be $(O(\log r), O(1))$-efficient.[4] The *naive amplifier* described in the introduction reduces the error probability exponentially and is $(O(\log r), 1)$-efficient, thus is $BPL$-applicable. However, it consumes a lot of random bits (i.e., $kr$). The other amplifiers mentioned in the introduction also manage to reduce the error substantially, and even with a small number of random bits. Unfortunately, the straightforward implementations of these amplifiers are only $(O(r), 1)$-efficient, hence are not $BPL$-applicable. We look for an amplifier that keeps both the space and the number of random bits small.

## 2.2 Weak extractors

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 2.6** *An* $(\boldsymbol{l}, \boldsymbol{r}, \boldsymbol{k})$**-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 $k$.*

[4] $BPL$-applicable amplifiers should also run in a polynomial-time. However, time is not a crucial issue in this discussion. It is easy to verify that all the amplifiers we consider run in a polynomial-time.

Consider an $(l, r, k)$-black-box simulator $M$ 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 $G_{M,r}$ as follows: $V_1$ corresponds to all the possible random strings of length $l = l(r)$ fed to $M$, when its input is $r$. $V_2$ corresponds to all the possible random strings of length $r$ fed to 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 being of $M$ a $(\delta, \epsilon)$-amplifier translates to a combinatorial property of $G_{M,r}$.

**Definition 2.7** *An $(l, r, k)$-bipartite graph is called a* **$(\delta, \epsilon)$-weak extractor** *if for every subset $W \subseteq V_2$, where $|W| < \epsilon 2^r$, the set $\{y \in V_1 : |\Gamma(y) \cap W| \geq \frac{k}{2}\}$ is of size $< \delta 2^l$, where $\Gamma(y)$ is the neighbor-set of $y$.*

**Lemma 2.8** *$M$ is an $(l, r, k)$-black-box simulator if and only if for all $r$ the graph $G_{M,r}$ is an $(l, r, k)$-bipartite graph. Furthermore, $M$ is a $(\delta, \epsilon)$-amplifier iff for all $r$ the graph $G_{M,r}$ is a $(\delta, \epsilon)$-weak extractor.*

The proof of this lemma is fairly simple, and can be found in [12].

*Weak extractors* were introduced by Cohen and Wigderson in [3] (they called them *majority dispersers*), and they lie between the two more familiar notions of *extractors* and *dispersers*.

**Definition 2.9** *An $(l, r, k)$-bipartite graph is called a* **$(\delta, \epsilon)$-extractor** *if for every subset $U \subseteq V_1$, where $|U| \geq \delta 2^l$, when choosing at random a node from $U$ and then a random neighbor of this node, the induced distribution on the right is $\epsilon$-close to uniform.*

**Definition 2.10** *An $(l, r, k)$-bipartite graph is called a* **$(\delta, \epsilon)$-disperser** *if for every subset $U \subseteq V_1$, where $|U| \geq \delta 2^l$, it holds that $|\Gamma(U)| \geq \epsilon 2^r$, where $\Gamma(U) = \cup_{u \in U} \Gamma(u)$.*

We should note that these notations deviate from the standard definitions in a few technical non significant details.[5] Our choice best illustrates the connection to amplification exhibited in Lemma 2.8.

If an $(l, r, k)$-bipartite graph is a $(\delta, \epsilon)$-extractor then it is also a $(\delta, \frac{1}{2} - \epsilon)$-weak extractor, and if it is a $(\delta, \epsilon)$-weak extractor then it is a $(\delta, \epsilon)$-disperser. Hence, extractors are useful in constructing black-box amplifiers, and analysis of dispersers may shed light on the limitations of black-box amplifiers. Moreover, dispersers are equivalent to black-box amplifiers that perform an OR function on the results of the simulations. Such amplifiers are useful in amplification of one-sided-error Monte-Carlo algorithms.

[5] $k$ is usually the logarithm of the left degree and $\delta$ is the min-entropy of the distribution on the left.

## 2.3 Bipartite graph explicitness

The equivalence between black-box simulators and bipartite graphs can be used in the context of simulator applicability too. *Explicit* bipartite graphs (ones in which neighborhoods are easy to compute) yield efficient simulators and vice versa.

We begin by introducing two new Turing machine variants:

**Definition 2.11** *An* **online Turing machine** *is a Turing machine with a one-way read-only input tape.*

**Definition 2.12** *A* **$k$-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 $f : \{0,1\}^* \rightarrow (\{0,1\}^*)^k$, 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 simulator 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 a simulator 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 2.13** *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 2.14** *A $k$-output machine is called* **$p$-parallel**, *if $p$ is the maximal (over all inputs $y$ and all time steps $t$) number of living output tapes.*

**Definition 2.15** *An online $k$-output machine is called* **$(S, p)$-efficient**, *if it is $p$-parallel and uses $S$ space.*

We are now ready for the explicitness definition:

**Definition 2.16** *A family of $(l, r, k)$-bipartite graphs $\{G_r\}$ (where $l = l(r)$ and $k = k(r)$) is called* $(\boldsymbol{S}, \boldsymbol{p})$**-explicit**, *if there is an $(S, p)$-efficient $k$-output machine $N_G$ (called the* **neighborhood machine***) with two input tapes: a two-way input tape on which it gets the parameter $r$ and a one-way input tape on which it gets a node $v$ on the left side of $G_r$. $N_G$ 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 bipartite graph explicitness and simulator efficiency. This, together with Lemma 2.8, enables us to concentrate on the study of weak extractors, and obtain conclusions about amplifiers.

**Theorem 2.17 (Equivalence Theorem)** *For $S \geq \log k$ an $(l, r, k)$-black-box simulator $M$ is $(O(S), p)$-efficient if and only if the family of $(l, r, k)$-bipartite graphs $\{G_{M,r}\}$ is $(O(S), p)$-explicit.*

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

$N_G$ follows the computation steps of $M$ accurately. Hence, it uses the same space ($O(S)$). Moreover, $N_G$'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 $N_G$ has is identical to the number of active simulations $M$ has. It follows that since $M$ is $p$-parallel then so is $N_G$. We conclude that $\{G_{M,r}\}$ is $(O(S), p)$-explicit.

The opposite direction is quite similar. $\square$

## 3 Formal statement of results

This paper has two main results. The first one shows that the AKS weak extractor is $(O(k), k)$-explicit. This is the first construction of a small-space online weak extractor.

**Theorem 3.1 (Upper Bound)** *Fix any constant $0 < \epsilon < \frac{1}{16}$. Then, for any $k = k(r)$ there exists an $(O(k), k)$-explicit family of $(r + O(k), r, k)$-bipartite graphs, that are $(\epsilon^{\Omega(k)}, \epsilon)$-weak extractors.*

Using Lemma 2.8 and the Equivalence Theorem (Theorem 2.17) we obtain a first example for amplifiers which use both a small space and a small number of random bits:

**Corollary 3.2 (Space Efficient Amplifiers)** *Fix any constant $0 < \epsilon < \frac{1}{16}$. Then, for any $k = k(r)$ there exists an $(O(k), k)$-efficient $(r + O(k), r, k)$-black-box simulator, which is an $(\epsilon^{\Omega(k)}, \epsilon)$-amplifier.*

The restriction on $\epsilon$ 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 this amplifier uses only $O(k)$ space and runs at most $k$ parallel simulations, then for a constant $k$ it is $BPL$-applicable:

**Corollary 3.3** *For any two constants $0 < \delta < \epsilon < \frac{1}{16}$ there exists a $BPL$-applicable $(r + O(1), r, O(1))$-black-box simulator, which is a $(\delta, \epsilon)$-amplifier.*

The main tool in the efficient weak extractor construction is a constant-space *online expander*. The main lemma proves that, under a suitable encoding, neighborhoods in the expander of Margulis [9] can be computed in a constant space on an online Turing machine. This result is interesting for itself:

**Lemma 3.4** *Fix any natural number $w \geq 2$. Then, for a varying $r$ there exists a family of constant-space online expander graphs on $w^{2r}$ vertices.*

The amplifier 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 this construction is optimal with respect to $BPL$-applicable black-box amplifiers that use $O(r)$ random bits. The result derives from a lower bound on $\delta$ in dispersers that are small-space-explicit. The lower bound shows that in such dispersers $\delta$ goes down with the disperser parallelism and not with the degree $k$:

**Theorem 3.5 (Lower Bound)** *For constants $0 < \epsilon < \frac{1}{2}$ and $f \geq 4$, for any $c = c(r)$, $k = k(r) < 2^{\frac{r}{f}}$, $\delta = \delta(r)$ and for an $(\frac{r}{f}, p)$-explicit family of $(cr, r, k)$-bipartite graphs, that are $(\delta, \epsilon)$-dispersers, it holds that $\delta \geq 2^{-O(q)}$, where $q = pcf$.*

Using Lemma 2.8 and the Equivalence Theorem we obtain the following conclusion for space-bounded amplification:

**Corollary 3.6** *For a constant $\epsilon$, for any $k = k(r)$, $\delta = \delta(r)$ and for a $BPL$-applicable $(O(r), r, k)$-black-box simulator, which is a $(\delta, \epsilon)$-amplifier, it holds that $\delta \geq \epsilon^{O(1)}$.*

## 4 The upper bound

This section presents the construction of $(O(k), k)$-explicit online weak extractors. The main tool in the construction is constant-space online expanders.

The section begins by a review of expander graphs, a presentation of the Margulis construction and a definition of *online expanders*. It presents an encoding for the Margulis construction, under which it is a constant-space online expander. Then, it proves that raising a constant-space online expander to a constant power yields a better expander, which is still constant-space and online. Finally, it shows the AKS weak extractor is $(O(k), k)$-explicit, by presenting a neighborhood algorithm that uses a constant-space online expander.

### 4.1 Expander graphs

We call a graph $G$ a $\gamma$-*expander* if every subset $S$ of at most half of its nodes has a neighborhood of size at least $(1+\gamma)|S|$.[6] There is a strong connection between the expansion factor ($\gamma$) of an expander graph and the second eigenvalue of its adjacency matrix. Let $M$ be the adjacency matrix of an expander graph $G$ and let $\lambda_1, \ldots, \lambda_N$ be its $N$ eigenvalues in a decreasing order of absolute values. It turns out that the smaller the ratio $\frac{|\lambda_2|}{d}$ the greater the expansion factor. Therefore, the second eigenvalue is sometimes used to measure the expansion quality of a graph. Furthermore, $\lambda_2$ is the actual parameter which matters in the AKS construction (employed below).

**Definition 4.1** *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 $\lambda$.*

Margulis [9] introduced a family of graphs $\{G_N\}$ for integers $N$ that have an integer square root $m$. Let $T_1$ and $T_2$ be $2 \times 2$ 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 [4] were the first to present such a bound. Jimbo and Maruoka [7] obtained the best estimation known for the second eigenvalue of this graph:

**Theorem 4.2 (Margulis, Gabber–Galil, Jimbo–Maruoka)** *The graph $G_N$ 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 $(N, 8, 5\sqrt{2})$-expander graph.*

We will 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 4.3** *A family of $d$-regular multi-graphs $\{F_N\}$ on $N$ vertices is called* $\boldsymbol{S}$**-space online of order** $\boldsymbol{q}$ *if the following hold:*[7]

1. *There exists an encoding $e_V$ for the vertex set of $F_N$ by binary strings of length $O(\log N)$.*
2. *There exists an encoding $e_d$ for $\{1, \ldots, d\}$ by binary strings of length $O(\log d)$.*
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 $F_N$, 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 $F_N$ that starts at $v$ and is specified by $j_1, \ldots, j_q$.*[8]

Some applications require highly expanding graphs. We presented a construction that has a relatively moderate expansion factor. It is standard to increase expansion by looking at powers of the graph.

**Definition 4.4** *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 4.5** *Let $G$ be an $(N, d, \lambda)$-expander. Then $G^{(q)}$ is an $(N, d^q, \lambda^q)$-expander.*

We leave the proof of this simple proposition to the reader.

### 4.2 Main lemmas

The following is the main lemma of the upper bound. It exhibits an encoding for the expander of Margulis, under which it is an online constant-space graph:

[6] Actually, the standard definition requires the neighborhood set to be of size at least $(1 + \gamma(1 - \frac{2|S|}{|G|}))|S|$.

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

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

**Lemma 4.6 (Lemma 3.4 restated)** *Fix any natural number $w \geq 2$. Then, for a varying $r$ the Margulis expander family (of Theorem 4.2), $\{G_{w^{2r}}\}$, is constant-space online.*

**Proof:** The encoding we present for the Margulis family uses an alphabet of size $w$. For simplicity we assume $w = 2$, in which case we can use a binary alphabet.

Every node in $G_{2^{2r}}$ corresponds to a pair $(x, y)$, where $x, y \in Z_{2^r}$. We encode a node by the sequence $x_1, y_1, x_2, y_2, \ldots, x_r, y_r$, where $x_1$ and $y_1$ are the least significant bits of $x$ and $y$ respectively, and $x_r$ and $y_r$ are their most significant bits. The neighbor indices $\{1, \ldots, 8\}$ are encoded standardly.

Fix an input $(e_d(j), e_V(v))$ for the algorithm, and denote by $v'$ the $j^{th}$ neighbor of $v$ ($e_V(v')$ should be the output of the algorithm). For $v = (x, y)$ and $v' = (x', y')$, both $x'$ and $y'$ are obtained by linear combinations of $x$,$y$ and $1$ with coefficients in $\{-1, 0, 1\}$ (e.g., $x - y - 1$). The combination itself is determined by $j$.

For computing $e_V((x', y'))$ from $e_V((x, y))$ (and $j$), one just performs a straightforward addition, while taking advantage of two facts: (1) For every $i$, the bits $x_i$ and $y_i$ in $e_V((x, y))$ are adjacent. (2) The $r$ bits of the sum modulo $2^r$ are exactly the $r$ least significant bits of the sum without a modulo. □

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

**Proof:** Let $R$ be the order-1 neighborhood machine of $\{F_N\}$. The order-$q$ neighborhood machine, $M$, runs $q$ copies $R_1, \ldots, R_q$ of $R$ simultaneously. $R_i$ computes the $i^{th}$ node in the walk.

$M$ starts by running $R_1, \ldots, R_q$ (in this order) until each $R_i$ reads all the bits of $e_d(j_i)$ from the input tape. The way $e_d(j_1), \ldots, e_d(j_q)$ are put on the input tape 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 on 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.

Per each $R_i$, the machine $M$ keeps $O(S)$ bits on its work tape to store the configuration of $R_i$. Therefore, it runs in $O(qS)$ space. Clearly, $M$ is an online machine. □

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

**Proof:** Note that every neighbor index $j$ in $F_N^{(q)}$ corresponds to a walk of length $q$ $j_1, \ldots, j_q$ on $F_N$. Hence, the neighborhood machine of $F_N^{(q)}$ has to compute the last node at $q$-length walks on $F_N$. It can use the order-$q$ neighborhood machine of $F_N$, by considering only its last output tape. By Lemma 4.7 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 4.9** *Fix any constant $\alpha > 0$ and a natural number $w \geq 2$. Then, for a varying $r$ there exists a constant-space online family of $(w^{2r}, d, \lambda)$-expanders, for which $d$ is constant and $\frac{\lambda}{d} < \alpha$.*

### 4.3 The weak extractor construction

In the following construction we use the AKS random-walk method with graphs as in Lemma 4.7 as expanders.

**Theorem 4.10 (Theorem 3.1 restated)** *Fix any constant $0 < \epsilon < \frac{1}{16}$. Then, for any $k = k(r)$ there exists an $(O(k), k)$-explicit family of $(r + O(k), r, k)$-bipartite graphs, that are $(\epsilon^{\Omega(k)}, \epsilon)$-weak extractors.*

**Proof:** We follow the [3, 6] construction of extractors using random walks on expander graphs [1]. Specifically, we construct a weak-extractor $H$, using a constant power of the Margulis expander $G_{2^r}$, for which $\frac{\lambda}{d} < \epsilon^{\Omega(1)}$. Call this expander $G$. The nodes in $V_1$ correspond to walks of length $k$ on $G$ and the nodes in $V_2$ correspond to the nodes of $G$. Each walk is connected to all the nodes that appear in it. To encode a walk we need $r$ bits for the starting point and $k \log d$ bits for the neighbor indices. Hence, $H$ is an $(r + k \log d, r, k)$-bipartite graph.

For the proof that $H$ is an $(\epsilon^{\Omega(k)}, \epsilon)$-weak extractor refer to [1, 3, 6].

Note that $N_H$, the neighborhood algorithm of $H$, has to compute walks of length $k$ on the online expander $G$. Lemma 4.7 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, $N_H$ needs only $O(k)$ space. $N_H$ uses all the $k$ output tapes throughout the whole computation, implying it is $k$-parallel. □

## 5 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 $\frac{r}{4}$-space neighborhood algorithms. We show that in such dispersers $\delta$ 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 $\delta$ 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 5.1** *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 say that $i$ is* **$m$-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 the input.

**Lemma 5.2** *Let $D$ be a $(cr, r, k)$-bipartite graph whose neighborhood algorithm $N_D$ is $p$-parallel. Then, for any input $y$ of $N_D$ there are at most $pcf$ output tapes, which are $\frac{r}{f}$-rich.*

**Proof:** Since the machine is online, each of the $cr$ input bits 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 $l^y_i$ such pairs, then there are at most $wp$ tapes for which $l^y_i > \frac{l}{w}$. Substituting $l = cr$ and $w = cf$ yields what is stated in the lemma. $\square$

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 $\frac{r}{4}$-space neighborhood algorithm, $\delta$ depends on the neighborhood algorithm parallelism.

**Theorem 5.3 (Theorem 3.5 restated)** *For constants $0 < \epsilon < \frac{1}{2}$ and $f \geq 4$, for any $c = c(r)$, $k = k(r) < 2^{\frac{r}{f}}$, $\delta = \delta(r)$ and for an $(\frac{r}{f}, p)$-explicit family of $(cr, r, k)$-bipartite graphs, that are $(\delta, \epsilon)$-dispersers, it holds that $\delta \geq 2^{-O(q)}$, where $q = pcf$.*

**Proof:** Let $D$ be some disperser as in the theorem, and let $N_D$ be its neighborhood algorithm.

Radhakrishnan and Ta-Shma show in [14] that for a constant $0 < \epsilon \leq \frac{1}{2}$ and for any $r$ and $l$, if $G$ is an $(l, r, k)$-bipartite graph, which is a $(\delta, \epsilon)$-disperser, then necessarily $\delta \geq 2^{-O(k)}$. We will construct from $D$ a $(cr, r, q)$-bipartite graph, which is a $(\delta, \frac{\epsilon}{2})$-disperser, implying that $\delta \geq 2^{-O(q)}$.

For each node $y \in V_1$, consider the execution of $N_D$ on $y$. Denote by $I^y = \{i^y_1, \ldots, i^y_q\}$ the $q$ richest output tapes in this execution, and by $J^y = \{j^y_1, \ldots, j^y_{k-q}\}$ the rest of the output tapes. Lemma 5.2 implies that the tapes in $J^y$ are all at most $\frac{r}{f}$-rich on $y$.

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 portion read during the tape's lifetime. Hence, the total number of possible outputs on the $i^{th}$ tape going over all the inputs in $S_i$ is at most the product of: (1) The number of configurations when the machine starts to write on this tape. Since $N_D$ uses at most $\frac{r}{f}$ space, this number is at most $2^{\frac{r}{f}}$ (recall our space accounting convention). (2) The number of possible input portions read during the tape's lifetime. For any input $y \in S_i$ it holds that $i \in J^y$, implying that the $i^{th}$ tape sees on $y$ an input portion of length at most $\frac{r}{f}$ (i.e. $l^y_i \leq \frac{r}{f}$). Therefore, the number of possible input portions is at most $2^{\frac{r}{f}+1}$. Hence, the total is at most $2^{\frac{2r}{f}+1}$.

Denote by $U_i \subseteq V_2$ the set of nodes that are written on the $i^{th}$ output tape when the input is in $S_i$. Denote by $U$ the union over all $i$: $U = \cup_{i=1}^{k} U_i$. The above observation means that $|U| \leq \sum_{i=1}^{k} |U_i| \leq k2^{\frac{2r}{f}+1} \leq 2^{\frac{3r}{f}+1} < \frac{\epsilon}{2}2^r$, for a sufficiently large $r$.

Color the edges of $D$ 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 $D$ into two dispersers: $D_1$ contains all the red edges and $D_2$ all the blue edges. Note that every $y$ has degree $q$ in $D_1$, and that all the edges in $D_2$ lead to nodes in $U$.

By the disperser property, every subset of $V_1$ of size at least $\delta 2^{cr}$ has at least $\epsilon 2^r$ neighbors. After dropping the edges of $D_2$, which only lead to nodes in $U$, each such subset has at least $\frac{\epsilon}{2}2^r$ neighbors. Therefore, $D_1$ is the $(cr, r, q)$-bipartite graph, which is a $(\delta, \frac{\epsilon}{2})$-disperser, we looked for. $\square$

## References


[1] 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.

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

[3] 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.

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

[5] O. Goldreich. A sample of samplers – a computational perspective on sampling (survey). *Electronic Colloquium on Computational Complexity (ECCC)*, TR97-020, 1997.

[6] 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.

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

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

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

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

[11] N. Nisan. $RL \subseteq SC$. *Computational Complexity*, 4(1):1–11, 1994.

[12] 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.

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

[14] 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.

[15] M. Saks and S. Zhou. $RSPACE(S) \subseteq DSPACE(S^{3/2})$. In *Proceedings of the* $36^{th}$ *Annual IEEE Symposium on Foundations of Computer Science*, pages 344–353, 1995.