Derandomization that is rarely wrong from short advice that is typically good

Oded Goldreich*
Weizmann Institute of Science
Rehovot, Israel.
oded@wisdom.weizmann.ac.il

Avi Wigderson†
Institute for Advanced Study
and Hebrew University
avi@ias.edu

July 21, 2002

Abstract

For every $\epsilon > 0$, we present a *deterministic* log-space algorithm that correctly decides undirected graph connectivity on all but at most 2^{n^ϵ} of the n-vertex graphs. The same holds for every problem in Symmetric Log-space (i.e., $\mathcal{SL}$).

Using a plausible complexity assumption (i.e., that $\mathcal{P}$ cannot be approximated by $\mathrm{SIZE}(p)^{\mathrm{SAT}}$, for every polynomial p) we show that, for every $\epsilon > 0$, each problem in $\mathcal{BPP}$ has a *deterministic* polynomial-time algorithm that errs on at most 2^{n^ϵ} of the n-bit long inputs. (The complexity assumption that we use is not known to imply $\mathcal{BPP} = \mathcal{P}$.)

All results are obtained as special cases of a general methodology that explores which probabilistic algorithms can be derandomized by generating their coin tosses *deterministically* from the input itself. We show that this is possible (for all but extremely few inputs) for algorithms which take advice (in the usual Karp-Lipton sense), in which the advice string is short, and most choices of the advice string are good for the algorithm.

To get the applications above and others, we show that algorithms with short and typically-good advice strings do exist, unconditionally for $\mathcal{SL}$, and under reasonable assumptions for $\mathcal{BPP}$ and $\mathcal{AM}$.

Keywords: Derandomization, Average-Case Complexity, Machines that Take Advice, Undirected Connectivity, Log-Space, BPP, Direct Product Problems.

*Supported by the MINERVA Foundation, Germany.
†Partially supported by NSF grants CCR-9987845 and CCR-9987077.

Contents

1 Introduction

1.1 A Motivating Example

More than two decades ago, Aleliunas *et. al.* [3] presented a *randomized log-space algorithm* for deciding undirected connectivity (UCONN). Their randomized algorithm triggered (or maybe only draw attention to) the following open problem:

Can undirected connectivity be decided by a deterministic log-space algorithm?

Despite extensive study, the above question is still open. The lowest space-bound currently known for deterministic algorithms (for undirected connectivity of n-vertex graphs) is $(\log n)^{4/3}$ [4], which builds upon [18] (obtaining space $(\log n)^{3/2}$) and improves upon Savitch's [21] classical bound of $(\log n)^2$. We show that if a deterministic log-space algorithm for UCONN does not exist, then this is due to very few graphs. That is:

Theorem 1 *For every $\epsilon > 0$, there exists a deterministic log-space algorithm that correctly decides undirected connectivity on all but at most 2^{n^ϵ} of the n-vertex graphs. Furthermore, the algorithm never outputs a wrong answer; it either outputs a correct answer or a special* ("don't know") *symbol. Such algorithms exist for every problem in Symmetric Log-space ($\mathcal{SL}$).*[1]

Surprisingly enough, the proof of Theorem 1 is not difficult (see Sections 3 and 4). It is based on a new viewpoint of the high-level "derandomization" process of Nisan, Szemeredi and Wigderson [18]. Under a different setting of parameters, for every $\epsilon > 0$, this process may be viewed as a *deterministic log-space algorithm that takes advice* (for UCONN), denoted A_{NSW}, that satisfies the following two conditions:

1. The advice string is relatively short. Specifically, the length of the advice is $n^{\epsilon/2}$, where n denotes the number of vertices in the input graph.

2. Most choices of the advice string are "good" *for all* n-vertex graphs. More precisely, the algorithm works correctly (i.e., decides correctly whether the input n-vertex graph is connected) whenever the $n^{\epsilon/2}$-bit long advice string is a universal traversal sequence for $n^{\epsilon/10}$-vertex graphs. Moreover, by [3], most advice strings satisfy that property.

Note that we use the term "advice" in the standard sense of Karp and Lipton [15]; an advice string is good if it makes the algorithm using it give the correct answer on *all* inputs of the given length. The remarkable property of algorithm A_{NSW} is that it satisfies *both* conditions above: it has good advice strings that are *much shorter* than the input, and furthermore most strings of this length are good advice strings.

Remark 2 *Note that the Condition 2* (i.e., many good advice strings) *is easy to obtain from every probabilistic algorithm. Indeed, Adleman's simulation [1] of probabilistic algorithms by nonuniform ones shows that any BPP-algorithm A can be modified into a deterministic A' which takes advice, for which most advice strings are good. However, since this transformation requires amplification*[2] *in order to enable a union bound over all inputs of a given length, the advice is* necessarily longer *than the input length, which violates the Condition 1* (i.e., short advice string).

[1] For a recent survey of this class, including a list of complete problems, see [2]

[2] Denoting by $A(x,r)$ the output of A on input x and coins r, we obtain A' by letting $A'(x,(r_1,....,r_{O(|x|)}))$ output the most frequent value among $A(x,r_1),...,A(x,r_{O(|x|)})$. Using sophisticated amplification methods, the sequence $(r_1,....,r_{O(|x|)})$ can be encoded by a string of length $O(|x|+|r|)$, but we cannot hope for length shorter than $|x|$ (because we need to reduce the error to less than $2^{-|x|}$ in order to apply the union bound).

To demonstrate the impact of having *many good advice strings that are shorter than the input*, we sketch how to complete the proof of Theorem 1. The general claim is that an advice-taking algorithm for which at least $2/3$ of the possible advice strings are good can be transformed into a (standard) deterministic algorithm (of comparable complexity) that errs on a number of inputs that is roughly exponential in the length of the (original) advice. Thus, we obtain a meaningful result if and only if the length of the advice is significantly smaller than the length of the input.

The claimed transformation is presented in two stages. First, we derive a randomized (log-space) algorithm A' that uses a *logarithmic amount of randomness* (and errs on at most 2^{n^ϵ} inputs). Specifically, on input G, algorithm A' uses its randomness as a seed to an adequate extractor (cf. [26]), and extracts *out of its input* G an advice string, s, of length $n^{\epsilon/2}$. Then A' invokes $A_{\rm NSW}$ on input G and advice s, and outputs whatever the latter does. It is easy to show that there can be at most $2^{(n^{\epsilon/2})^2}$ inputs on which A' errs with probability greater than $1/2$. Applying a straightforward derandomization (and ruling by majority), we obtain a deterministic (log-space) algorithm A'' that decides correctly on all but at most of the 2^{n^ϵ} inputs.

1.2 The underlying principle

The above transformation can be applied to any advice-taking algorithm, and is meaningful if and only if most of the advice strings are good and the length of the advice is significantly smaller than the length of the input. That is:

Theorem 3 *Let A be an advice-taking polynomial-time* (resp., log-space) *algorithm for a problem Π, and let $\ell : \mathsf{N} \to \mathsf{N}$. Suppose that for every n it holds that at least a $2/3$ fraction of the $\ell(n)$-bit long strings are* good advice stings *for A; that is,*

$$\Pr_{r\in\{0,1\}^{\ell(n)}}[\forall x \in \{0,1\}^n \text{ it holds that } A(x,r) = \Pi(x)] \;\geq\; \frac{2}{3}$$

where $\Pi(x)$ denotes the correct answer for x. Then, for every $c > 1$, there exists a deterministic polynomial-time (resp., log-space) *algorithm for Π that errs on at most $2^{\ell(n)^c}$ of the n-bit inputs. Furthermore, in case $\ell(n) = \Omega(n)$, the resulting algorithm errs on at most $2^{c\cdot\ell(n)}$ of the n-bit inputs.*

The proof of Theorem 3 appears in Section 3. As hinted above, the proof proceeds by viewing the input itself as a source of randomness and extracting a random advice string via an adequate extractor (which uses logarithmically long seeds). The extracted advice string is sufficiently random (and thus is a good advice with probability greater than $1/2$) provided that the input comes from a source with min-entropy sufficiently larger than the length of the extracted advice. It follows that the number of inputs on which most extracted advice strings are not good can be bounded in terms of the min-entropy bound.

1.3 Other applications

The question is how to obtain algorithms to which Theorem 3 can be (meaningfully) applied. We have seen already one such example (i.e., $A_{\rm NSW}$), and in this subsection we will discuss some more.

1.3.1 Conditional derandomization of BPP and AM

As mentioned above, any randomized algorithm can be transformed into a (deterministic) advice-taking algorithm for which most possible advice strings are good. The problem is that the length of the advice will be longer than the length of the input. One natural idea is to use an adequate pseudorandom generator in order to shrink the length of the advice strings, while maintaining the fraction of good advice strings. We observe that the property of being a good advice string for a specific algorithm can be efficiently tested using a SAT-oracle. Thus, it suffices to have pseudorandom generators that withstand probabilistic polynomial-time distinguishers that use a SAT-oracle. (Actually, it suffices to have pseudorandom generators that withstand distinguishers that correspond to the class $\mathcal{MA}$.)

Recall that pseudorandom generators that withstand non-uniform polynomial-size distinguishers that use a SAT-oracle were constructed before (cf. [5, 16]) under various non-uniform assumptions (which also refer to circuits with SAT-oracle gates). In particular, we will use the following result:

Theorem 4 (informal, implicit in [16, 19]): *Suppose that for every polynomial p there exist a predicate in $\mathcal{P}$ that is hard to approximate*[3] *in* $\mathrm{SIZE}(p)^{\mathrm{SAT}}$. *Then, for every polynomial q, there exists a deterministic polynomial-time algorithm that expands k-bit long random seeds to $q(k)$-bit long sequences that are indistinguishable from random ones by any $q(k)^2$-size circuit that uses a* SAT-*oracle.*

Combining Theorem 4 with the observation that being a good advice string for a polynomial-time algorithm is testable in comparable time with the help of an SAT-oracle, we obtain (see proof in Section 5):

Theorem 5 (informal) *For every polynomial p and constant $\epsilon > 0$, under the assumption of Theorem 4, every probabilistic p-time algorithm can be converted into a functionally-equivalent polynomial-time advice-taking algorithm that uses advice strings of length n^ϵ, such that more than $2/3$ fraction of the possible advice strings are good.*

Combining Theorem 3 and 5, we obtain (see proof in Section 5):

Corollary 6 (informal) *Under the assumption of Theorem 4, for every $\epsilon > 0$, every language in $\mathcal{BPP}$ can be decided by a deterministic polynomial-time algorithm that errs on at most 2^{n^ϵ} of the n-bit long inputs.*

Interestingly, under the same assumption we can also derandomize $\mathcal{AM}$ (see proof in Section 5):

Theorem 7 (informal) *Under the assumption of Theorem 4, for every $\epsilon > 0$, every language in $\mathcal{AM}$ can be decided by a non-deterministic polynomial-time algorithm that errs on at most 2^{n^ϵ} of the n-bit long inputs.*

Comparison to previous results: When making the comparison, we refer to three issues:

1. The running-time of the derandomization,
2. For how many instances does the derandomization fail,
3. The intractability assumption used.

[3] Here, hard to approximate may mean that any machine in the class fails to guess the correct value of a random instance with success probability greater than $2/3$. Using Yao's XOR Lemma (cf. [27, 8]), hardness to approximate can be amplified such that this class fails to guess the correct value of a random n-bit instance with probability greater than $(1/2) + (1/p(n))$. (Recall that such amplification requires only logarithmically many instances, and thus in our case it affects the resource bounds in a minor way.)

Our focus is on polynomial-time derandomization. Thus, Corollary 6 should be compared with Impagliazzo and Wigderson [12], who proved that $\mathcal{BPP} = \mathcal{P}$ under the assumption that *for some* $c > 0$*, the class* $\mathcal{E}$ *is not contained in* $\mathrm{SIZE}(2^{cn})$. Likewise, Theorem 7 should be compared with Kilvans and van Melkebeek [16], who proved that $\mathcal{AM} = \mathcal{NP}$ under the assumption *for some* $c > 0$*, the class* $\mathcal{E}$ *is not contained in* $\mathrm{SIZE}(2^{cn})^{\mathrm{SAT}}$. Both [12] and [16] present perfect derandomizations, whereas our derandomizations fail on very few inputs (which is of course weaker).

Comparing the assumptions is easier for the $\mathcal{AM}$ derandomization, as both Theorem 7 and [16] refer to lower bounds for circuits with SAT oracle gates. But as Theorem 7 refers to separation of classes with smaller resources, our assumption is weaker.[4] As for the $\mathcal{BPP}$ derandomization, it seems that the assumption made by [12] and Corollary 6 are incomparable. We still need the SAT oracles, but as above, our classes are lower.

There is another set of derandomization results, typically under uniform complexity assumptions (which are seemingly weaker than the analog non-uniform assumptions, cf. [13]). These results yield deterministic simulations that may make many errors; however, no efficient procedure can find inputs on which the simulation errs. This notion of imperfect simulation seems incomparable to ours; we seem to make much fewer errors, but the inputs for which errors occur may be easy to generate.

About our assumption: To gain some intuition about the assumption used in Corollary 6, we consider a few plausible conjectures that imply it.

- *Given a Boolean circuit, determine whether it evaluates to 1 on more than half of its possible inputs.* Clearly, this problem can be decided in time that is exponential in the number of inputs to the circuit, but it seems hard to decide it in significantly less time. Specifically, suppose that the input circuit has size n and $\ell(n)$ input bits (e.g., $\ell(n) = O(\log n)$), then the problem is solvable in time $2^{\ell(n)} \cdot n$ but seems hard to decide in time $2^{\ell(n)/10}$. Furthermore, the problem seems hard even for $2^{\ell(n)/10}$-size circuit that use SAT-gates (i.e., an oracle to SAT). However, conjecturing that this problem is hard even to approximate (i.e., when given a random circuit as input) requires a suitable notion of the distribution of inputs (i.e., circuits). A conceivably good definition is that the input/circuit distribution is uniform over $\ell(n)$-variate polynomials over GF(2) having n monomials.

- For any polynomial q, we consider the following decision problem: *Given a description of a prime P and an natural number $x < P$, decide whether most of the natural numbers in the set $\{x+1, ..., x+q(|P|)\}$ are quadratic residues modulo P.* Clearly, this problem can be decided in time $q(|P|) \cdot |P|^3$, but it seems hard to decide (or even approximate) it by (say) $q(|P|)^{1/3}$-size circuits even ones having SAT-gates.

- *Given a sequence of n-by-n matrices, $A_1, ..., A_\ell$, over a sufficiently large finite field, determine* $\sum_{S \subseteq [\ell]} \det(\sum_{i \in S} A_i)$, where $\det(M)$ denote the determinant of the matrix M. Again, this problem can be solved in time $2^\ell \cdot n^3$, but it seems hard to solve it by (say) $\min(2^{\ell/3}, \ell^{n/3})$-size circuits, even ones having SAT-gates.[5] (Again, we may use $\ell = O(\log n)$.)

[4]For example, if $\mathcal{P}$ is contained in $\mathrm{SIZE}(n^2)$ then $\mathcal{E}$ is contained in $\mathrm{SIZE}(2^{cn})$ for every $c > 0$, but the converse is not known. Thus, "$\mathcal{E}$ is not contained in $\mathrm{SIZE}(2^{cn})$ for every $c > 0$" implies "$\mathcal{P}$ is not contained in $\mathrm{SIZE}(n^2)$" (but again the converse is not known). Also note that, in case of predicates in $\mathcal{E}$, hardness on the worst-case yields hardness to approximate.

[5]The $\ell^{n/3}$ term, which is meaningful only if $\ell = \Omega(n \log n)$, is introduced to account for a possible $(\ell^n \cdot n^3)$-time algorithm that computes the formal (degree n) polynomial $p(x_1, ..., x_\ell) \stackrel{\mathrm{def}}{=} \det(\sum_{i=1}^{\ell} x_i A_i)$, and computes the sum

We comment that this problem is downward-self-reducible and random-self-reducible (see Appendix). This is particularly interesting in light of the paper [13], which shows that for such functions *uniform, worst-case hardness* implies that they can be used in the NW-generator (obtaining derandomization on the average). More concretely, assume this function is not in $\mathrm{BP}\textsc{time}(p)^{\mathrm{SAT}}$, for any fixed polynomial p, for infinitely many input lengths. Then every language in BPP has a deterministic polynomial-time algorithm which, for infinitely many input lengths errs on at most 2^{n^ϵ} inputs, with $\epsilon > 0$ an arbitrarily small constant. To summarize, the special structure of this function, enables reducing our hardness assumption from non-uniform and average-case to uniform and worst-case. Thus, it will be interesting to try and substantiate (in direct or indirect ways), the conjecture that for this function is indeed difficult to compute in time $2^{\ell/O(1)}$.

1.3.2 Direct Product Problems

Direct product problems yield another interesting case where randomized algorithms can be transformed into ones having relatively short good advice strings. Specifically, any good advice string for the original problem constitutes a good advice string for the direct product problem. Applying Theorem 3, we obtain a deterministic algorithm (of related complexity) for the direct product problem that errs on a number of inputs that is independent of the arity of the (direct product) problem. For further details, see Section 6.

2 Preliminaries

In this section we recall some standard notions and notations. We will also recall some known results.

2.1 Randomness and Extractors

We consider random variables that are assigned binary values as strings. In particular, U_m will denote a random variable that is uniformly distributed over $\{0,1\}^m$. The min-entropy of a random variable X is the maximal (real number) k such that for every string x it holds that $\Pr[X = x] \le 2^{-k}$.

The statistical difference between two random variables X and Y, denoted $\Delta(X,Y)$, is defined as $\frac{1}{2} \cdot \sum_z |\Pr[X = z] - \Pr[Y = z]||$. Clearly, $\Delta(X,Y) = \max_S\{\Pr[X \in S] - \Pr[Y \in S]\}$. We say that Y is ϵ-close to X if $\Delta(X,Y) \le \epsilon$; otherwise, we say that Y is ϵ-far from X.

A function $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^\ell$ is called a (k,ϵ)-extractor if for every random variable X that has min-entropy at least k it holds that $E(X,U_t)$ is ϵ-close to U_ℓ. The following fact is well-known and easy to establish:

Fact 8 *Suppose that $E : \{0,1\}^n \times \{0,1\}^t \to \{0,1\}^\ell$ is a (k,ϵ)-extractor. Then, for every set $S \subseteq \{0,1\}^\ell$, all but at most 2^k of the x's in $\{0,1\}^n$ satisfy $\Pr[E(x,U_t) \in S] \ge (|S|/2^\ell) - \epsilon$.*

Proof: Let $B \subseteq \{0,1\}^n$ denote the set of x's that satisfy $\Pr[E(x,U_t) \in S] < (|S|/2^\ell) - \epsilon$. Consider a random variable X that is uniformly distributed on the set of the latter x's. Then,

$$\Delta(E(X,U_t),U_\ell) \ge \Pr[E(X,U_t) \in S] - \Pr[U_\ell \in S] > \epsilon$$

Thus, $\log_2|B| < k$ must hold, and $|B| < 2^k$ follows. ■

$\sum_{x_1,...,x_\ell \in \{0,1\}} p(x_1,...,x_\ell)$ by computing separately the contribution of each of the $\binom{n+\ell}{n}$ terms of p.

Uniform families of extractors: We actually consider families of (extractor) functions (of the above type). These families are parameterized by n, whereas t, ℓ, k and ϵ are all functions of n. Thus, when we say that $\{E_n : \{0,1\}^n \times \{0,1\}^{t(n)} \to \{0,1\}^{\ell(n)}\}_{n\in\mathsf{N}}$ is a (k,ϵ)-extractor we mean that for every n the function E_n is a $(k(n), \epsilon(n))$-extractor. We will use the following well-known result of Trevisan [26]:[6]

Theorem 9 *For every $c > 1$, $\epsilon > 0$ and every linear-space computable $\ell : \mathsf{N} \to \mathsf{N}$, there exists a polynomial-time computable family of functions $\{E_n : \{0,1\}^n \times \{0,1\}^{O(\log n)} \to \{0,1\}^{\ell(n)}\}_{n\in\mathsf{N}}$ that constitute a (k,ϵ)-extractor, where $k(n) \stackrel{\text{def}}{=} \ell(n)^c$. Furthermore, these functions are computable in log-space.*

2.2 Some Complexity Classes

We denote by $\mathrm{SIZE}(p)$ the class of (non-uniform) families of p-size circuits, and by $\mathrm{SIZE}(p)^{\mathrm{SAT}}$ the class of such circuits augmented by SAT-gates (oracle gates to SAT).

We refer to two restricted classes of interactive proof systems (cf. [10]), specifically $\mathcal{MA}$ and $\mathcal{AM}$. The class $\mathcal{MA}$ (resp., $\mathcal{AM}$) consists of all languages having a two-round public-coin interactive proof in which the prover (called Merlin) sends the first (resp., second) message (cf. [6]). In both cases, the verifier's (Arthur's) message consists of the entire contents of its random-tape (hence the term *public-coin*). By $\mathcal{MA}(p)$ (resp., $\mathcal{AM}(p)$) we denote a parameterized version of $\mathcal{MA}$ (resp., $\mathcal{AM}$) in which the verifier's complexity is bounded by p.

3 The transformation: Proof of Theorem 3

We combine an algorithm A as in the hypothesis (of Theorem 3) with an adequate extractor to obtain the desired deterministic algorithm. Specifically, let use denote by S the set of good advice strings of algorithm A; that is, for every $r \in S$ and $x \in \{0,1\}^n$, it holds that $A(x,r) = \Pi(x)$. Then, by the theorem's hypotheses, $|S| \geq (2/3) \cdot 2^{\ell(n)}$. Let $E_n : \{0,1\}^n \times \{0,1\}^{t(n)} \to \{0,1\}^{\ell(n)}$ be a $(k(n), 0.1)$-extractor. Then, by Fact 8, for all but at most $2^{k(n)}$ of the n-bit long x's, the probability that $E_n(x, U_{t(n)}) \in S$ is at least $(2/3) - 0.1 > 1/2$. Thus, for all but at most $2^{k(n)}$ of the x's, for a strict majority of the (extractor seeds) $r' \in \{0,1\}^{t(n)}$ it is the case that $E_n(x,r') \in S$. Scanning all possible $r' \in \{0,1\}^{t(n)}$, an ruling by the majority of the $A(x, E_n(x,r'))$ values, we obtain the correct answer for all but at most $2^{k(n)}$ of the n-bit long inputs. (The resulting algorithm is depicted in Figure 1.)

Clearly, the time complexity of the resulting algorithm is $2^{t(n)} \cdot (T_A(n) + T_E(n))$, where T_A (resp., T_E) denotes the time complexity of algorithm A (resp., of the extractor $E = \{E_n\}_{n\in\mathsf{N}}$). Similarly, the space complexity of the resulting algorithm is $t(n) + S_A(n) + S_E(n)$, where S_A (resp., S_E) denotes the space complexity of A (resp., of E).

Using Theorem 9, we may set $t(n) = O(\log n)$ and $k(n) = \ell(n)^c$, while using a log-space computable extractor. Thus, the main part of Theorem 3 follows. To establish the furthermore part of Theorem 3 (which refers to $\ell(n) = \Omega(n)$), we use Zuckerman's extractor [28] instead of Theorem 9.[7]

[6] We mention that both the error-correcting code and the (weak) designs used by Trevisan's extractor can be constructed in log-space.

[7] To handle log-space algorithms, we need a log-space computable extractor for this case (i.e., of $\ell(n) = \Omega(n)$ and $k(n) = c \cdot \ell(n)$). Such extractors do exist [22].

On input $x \in \{0,1\}^n$, scan all possible $r' \in \{0,1\}^{t(n)}$,
performing the following steps for each r':

1. Obtain $r \leftarrow E_n(x, r')$.
2. Invoke A on input x and advice r, and record the answer in $v(r')$.

Output the majority value in the sequence of $v(r')$'s.

Figure 1: The resulting deterministic algorithm.

Remark 10 *If algorithm A never errs* (but may rather output a special "dont know" symbol on some (x, r) pairs) *then the same property is inherited by the resulting deterministic* (single-input) *algorithm. A similar statement holds for one-sided error of algorithms for decision problems.* (In both cases, we may actually use dispersers instead of extractors.)

4 Undirected Connectivity: Proof of Theorem 1

As explained in the introduction, Theorem 1 is proved by applying Theorem 3 to an advice-taking algorithm that is implicit in (or rather derived from) the work of Nisan, Szemeredi and Wigderson [18]. The latter algorithm refers to the notion of a universal traversal sequence for the set of all (3-regular) graphs of a certain size. Such sequences were defined by Cook (cf. [3]) and extensively studied since. There are many variants of this definition, and we choose one that is most convenient for our purpose.

Definition 11 (Universal traversal sequences for d-regular graphs):

- *Let G be a d-regular graph, v be a vertex in G, and $\sigma = (\sigma_1, ..., \sigma_t)$ be a sequence over $[d] \stackrel{\text{def}}{=} \{1, ..., d\}$. The* σ-directed G-walk starting at *v is the vertex sequence $(v_0, v_1, ..., v_t)$, where $v_0 = v$ and v_{i+1} is the σ_i^{th} neighbor of v_i.*
- *We say that $\sigma \in [d]^*$ is* (d, m)-universal *if for every m-vertex d-regular graph G and every vertex v in G, the σ-directed G-walk starting at v visits all the vertices of the connected component of v in G.*

  We say that $\sigma \in \{0,1\}^*$ is (d, m)-universal *if when viewed as a sequence over $[d]$ it is (d, m)-universal.*

The following result is implicit in the work of Nisan, Szemeredi and Wigderson [18].

Theorem 12 (following [18]): *Let $\ell, m : \mathsf{N} \to \mathsf{N}$ be space-constructible functions such that $m(n) \leq \ell(n) \leq n$ for all n's.*[8] *Then, there exists a deterministic $O((\log^2 n)/(\log m(n)))$-space advice-taking algorithm A_{NSW} for UCONN that on input an n-vertex graph uses an advice string of length $\ell(n)$,*

[8]The growth restriction on these functions is natural in the context of [18]. Of course $m(n) \leq \ell(n)$ must hold, as otherwise the claim holds vacuously (because m-universal sequences must have length greater than m). For $\ell(n) > n$, algorithm A_{NSW} uses space $O((\log \ell(n))(\log n)/(\log m(n)))$ rather than $O((\log^2 n)/(\log m(n)))$.

where the set of good advice strings contains every $(3, m(n))$*-universal sequence.*[9] *Furthermore, algorithm* A_{NSW} *never errs, but may rather output a special* ("dont know") *symbol* (in case the advice string is not good).

Nisan *et. al.* [18] used the fact that (by [17]) $(3, m)$-universal sequences of length $\exp(\log^2 m)$ can be constructed in $O(\log^2 m)$-space. Setting $\ell(n) = n$ and $m(n) = \exp(\log^{1/2} m)$, their $O(\log^{3/2} m)$-space algorithm follows by combining Theorem 12 with the $O(\log n)$-space algorithm (of Nisan [17]) for constructing a $m(n)$-universal sequence of length n. Here, instead, we use the well-known fact that most sequences of length $\tilde{O}(m^3)$ are $(3, m)$-universal [3]. That is:

Theorem 13 (Aleliunas *et. al.* [3]): *More than a* $2/3$ *fraction of the* $O(m^3 \cdot \log m)$*-bit long strings are* $(3, m)$*-universal sequences.*

Setting $\ell(n) = O(m(n)^3 \cdot \log m(n)) < m(n)^4$, it follows that a $2/3$ fraction of the $\ell(n)$-bit long strings are $(3, \ell(n)^{1/4})$-universal, and thus are good advice strings for A_{NSW} (when applied to n-vertex graphs). Specifically, under this setting, A_{NSW} uses space $O((\log^2 n)/\log m(n)) = O((\log^2 n)/\log \ell(n))$ (and $\ell(n)$-bit long advice strings). For $\ell(n) = n^{1/O(1)}$, we obtain a deterministic log-space advice-taking algorithm for UCONN that uses $\ell(n)$-bit long advice strings such that at least a $2/3$ fraction of the possible advice strings are good.

We are now ready to invoke Theorem 3: given any desired constant $\epsilon > 0$, we set $\ell(n) = n^{\epsilon/c}$ (for any constant $c > 1$) and invoke Theorem 3. This yields a deterministic log-space algorithm for UCONN that errs on at most $2^{\ell(n)^c} = 2^{n^\epsilon}$ of the n-vertex graphs. The main part of Theorem 1 follows. By Remark 10, the fact that A_{NSW} never errs is inherited by the log-space algorithm that we derive, and thus the furthermore-part of Theorem 1 follows.

USTCONN and $\mathcal{SL}$. The arguments presented above apply also to USTCONN, where one is given an undirected graph G and a pair of vertices (u, v) and needs to determine whether or not these vertices are connected in G. Actually, the main algorithm presented in [18] is for that version, and so all the above holds. Since USTCONN is log-space complete for the class $\mathcal{SL}$ (symmetric log-space), it is easy to see that for every $\epsilon > 0$ every problem in $\mathcal{SL}$, has a deterministic log-space algorithm which is always correct, and answers "dont-know" on at most 2^{n^ϵ} inputs of length n. Indeed, one only has to observe that log-space reductions can only blow-up the input length polynomially, and since ϵ can be made arbitrarily small we get the same bound on the number of "dont-know"s. A compendium of interesting problems in $\mathcal{SL}$ can be found in [2].

5 Derandomizing BPP and AM

Comments on the proof of Theorem 4: With a minor modification (to be discussed), Theorem 3.2 in (the full version of) [16] yields Theorem 4. Theorem 3.2 in [16] assumes the existence of a predicate that is hard to approximate[10] by $\text{SIZE}(p)^{\text{SAT}}$, and conclude that a certain pseudorandom generator expanding k-bit strings to $q(k)$-bit strings exists, where $q(k) \leq p(\sqrt{k} \log q(k))^{1/2}$. The resulting generator withstand $q(k)$-sized circuits with SAT-gates, and operates in time related to

[9] Nisan *et. al.* [18] work with an auxiliary 3-regular $O(n^2)$-vertex graph that is derived from the initial n-vertex graph in a straightforward manner. The algorithm works in iterations, where a good advice allows to shrink the size of the graph by a factor of $m(n)/4$ in each iterations. (In case the advice is not good, the algorithm may fail to shrink the graph, but will detect this failure, which justifies the furthermore clause.) Thus, there are $(\log n)/(\log(m(n)/4))$ iterations, and each iteration is implementable in $O(\log n)$ space.

[10] See Footnote 3.

the complexity of evaluating the predicate on $(\sqrt{k}\log q(k))$-bit long inputs and to $2^{O(k)}$. However, the latter additive term (of $2^{O(k)}$) is merely due to the fact that [16] use the brute-force design construction of [19] rather than their efficient (i.e., poly(k)-time) construction, which can be used whenever p is a polynomial.[11] (Another minor detail is that we want to withstand $q(k)^2$-sized circuits rather than withstand $q(k)$-sized circuits.) Theorem 4 follows by setting, for any given polynomial q, the polynomial p such that $q(k)^2 < p(\sqrt{k}\log q(k))^{1/2}$ holds (e.g., $p(n) \stackrel{\text{def}}{=} (q(n^2))^4$). ■

Proof of Theorem 5: For each set in $\mathcal{BPP}$, by using straightforward amplification, we obtain a randomized polynomial-time algorithm that errs with probability at most $2^{-(n+2)}$ (on each n-bit input). This algorithm yields an advice-taking algorithm for which at least 3/4 of the possible advice strings are good. We call such an algorithm canonical.

For any polynomial q and $\epsilon > 0$, we construct a generator $G_{\epsilon,q}$ as in Theorem 4 such that n^ϵ-bit long strings are stretched into sequences of length $q(n)$ that pass all $q(n)^2$-size distinguishers (i.e., circuits with SAT-gates).

We claim that for every q-time canonical algorithm, A, at least a 2/3 fraction of the sequences generated by $G_{\epsilon,q}$ are good advice strings. Otherwise, we consider an $\mathcal{MA}$-proof system for bad (i.e., non-good) advice strings. On input a string $r \in \{0,1\}^m$, where $m = q(n)$, the prover (i.e., Merlin) sends $x \in \{0,1\}^n$, and the verifier accepts if and only if $A(x,r)$ differs from the majority vote of $A(x,s)$ taken over a sample (of say 100) uniformly selected $s \in \{0,1\}^m$. Note that if r is a bad advice string then the verifier accepts with probability at at least 0.99 (provided Merlin acts optimally), whereas if r is a good (i.e., not bad) advice string then the verifier accepts with probability at most 0.01 (no matter what Merlin does). Thus, the probability that the verifier accepts a string produced by $G_{\epsilon,q}$ is at least $(1/3) \cdot 0.99 = 0.33$, whereas the probability that the verifier accepts a uniformly distributed $q(n)$-bit long string is at most $(1/4) \cdot 1 + (3/4) \cdot 0.01 < 0.3$. Observe that the verifier's running-time is bounded by $O(m) = O(q(n))$. Applying known transformations from $\mathcal{MA}$ to $\mathcal{AM}$, and from the latter to $\mathrm{BP}\textsc{time}^{\mathrm{SAT}}$, yields a distinguisher in $\mathrm{BP}\textsc{time}(m^2)^{\mathrm{SAT}}$ for m-bit inputs (provided that $q(n) > n\log n$).[12] We derive a contradiction to the security of $G_{\epsilon,q}$, and the theorem follows. ■

Proof of Corollary 6: By applying Theorem 5 (with ϵ replaced by $\epsilon/2$), we can derive for any set in $\mathcal{BPP}$ a polynomial-time advice-taking algorithm with advice strings of length $n^{\epsilon/2}$ such that at least 2/3 of the advice strings are good. Then applying Theorem 3 (with $c = 2$), the current claim follows. ■

Proof of Theorem 7: We just follow the proof of Corollary 6, adapting the notion of a good advice to the AM setting, and observing that the proof of Theorem 5 still applies. Specifically, a good advice for an AM-game is a verifier message that is good for all inputs of a certain length (i.e., for inputs in the language there exist an acceptable prover response, whereas no such response exists in case the input is not in the language). The $\mathcal{MA}$-proof system described in the proof of

[11] Indeed, Theorem 3.2 in [16] is stated for arbitrary p's, and the focus is actually on super-polynomial p's.

[12] The transformation of $\mathcal{MA}$ to $\mathcal{AM}$ increases the running-time by a factor related to the length of the original Merlin's message, which equals n in our case. In the second transformation (i.e., from $\mathcal{AM}$ to $\mathrm{BP}\textsc{time}^{\mathrm{SAT}}$), we need to make a SAT-query that is answered with an optimal Merlin message. This amounts to encoding the accepting predicate of Arthur as a SAT-instance, where the length of that instance is almost linear in the verifier's running-time (on a multi-tape Turing machine). Thus, $\mathcal{MA}(m) \subseteq \mathcal{AM}(mn) \subseteq \mathrm{BP}\textsc{time}(mn\log(nm))^{\mathrm{SAT}}$.

Theorem 5 extends in the straightforward manner (i.e., Merlin now sends an adequate x along with an adequate prover message for the AM-game). Indeed, moving to $\mathcal{AM}$ will blow-up the complexity by a factor related to the prover message, and so we should start (w.l.o.g.) with a AM-game in which the verifier's messages are longer than the prover messages.[13] But otherwise, the argument remains intact. ■

6 Direct Product Problems

The following observation is due to Noam Nisan: A natural domain where one may obtain advice-taking algorithms with good advice strings that are much shorter than the input is the domain of direct product problems. That is, suppose that Π is a problem having a (polynomial-time) randomized algorithm. As we have commented in the introduction, by straightforward amplification [1] we may obtain an advice-taking algorithm for Π such that for some polynomial ℓ, the algorithm uses $\ell(n)$-bit long advice for n-bit long inputs such that at least a $2/3$ fraction of all possible advice are good. The key point is that such an advice-taking algorithm for Π yields an algorithm with similar performance for the direct product of Π. That is, for any n and t, given input $(x_1, ..., x_t) \in \{0,1\}^{t \cdot n}$ and an $\ell(n)$-bit long advice, the latter algorithm invokes the single-instance algorithm on each x_i using the same advice in all invocations. Clearly, if the advice is good for the single-instance algorithm then it is also good for the multiple-instance algorithm. Applying Theorem 3, we obtain

Theorem 14 *For every problem Π in $\mathcal{BPP}$, there exist a polynomial p and a deterministic polynomial-time algorithm A such that for every n and t, for all but at most $2^{p(n)}$ of the sequences $\overline{x} = (x_1, ..., x_t) \in \{0,1\}^{t \cdot n}$ it holds that $A(\overline{x}) = (\Pi(x_1), ..., \Pi(x_t))$.*

We stress that the number of inputs on which A may err depends only on the length of the individual Π-instances (i.e., n), and not on their number (i.e., t). We comment that this is superior to what could be obtained by straightforward considerations.[14]

Acknowledgments

We are grateful to Noam Nisan for suggesting the application to direct product presented in Section 6. We also thank Eric Allender and the anonymous reviewers of *Random'02* for their helpful comments.

---

[13] Specifically, assuming that the prover's messages in the AM-game are shorter than m, we get an $\mathcal{MA}(m)$ proof system for bad advice strings, which is transformed to an $\mathcal{AM}(m^2)$ proof system, which in turn resides in $\mathrm{BPTIME}(m^2 \log(m^2))^{\mathrm{SAT}}$. So we should use a generator as in Theorem 4 such that n^ϵ-bit long strings are stretched into sequences of length $q(n)$ that pass all $q(n)^3$-size distinguishers (rather than $q(n)^2$-size ones).

[14] For example, suppose that Π has a BPP-algorithm of randomness complexity ρ (such that $\rho(n) \geq n$ or else we can derandomize Π itself). Then, by straightforward amplification, we obtain a 2/3-majority of good advice strings for advice length $\ell(n) = O(\rho(n) \cdot n)$ (or even $\ell(n) = O(\rho(n) + n)$ by using expander-based amplification). Thus, in Theorem 14, we obtain $p(n) = \ell(n)^c$, for any desired $c > 1$. In contrast, if we use $x_2, ..., x_t$ to generate $m \approx t/\rho(n)$ disjoint random pads for invocations of the BPP-algorithm on input x_1 then we may err on $\exp(-m) \cdot 2^{tn} = 2^{(1-o(1)) \cdot tn}$ of the sequences. Using $x_2, ..., x_t$ to generate $tn/(\log d)$ related pads (by using a d-regular optimal expander) may yield error on $(1/\sqrt{d})^{tn/(\log d)} \cdot 2^{tn} = 2^{tn/2}$ sequences. For large t, this is inferior to the upper bound of $2^{O(\rho(n) \cdot n)^{3/2}} \leq 2^{\rho(n)^3}$ sequences (not to mention $2^{O(\rho(n)+n)^c} = 2^{\rho(n)^c}$, for any $c > 1$) obtained by using Theorem 14.

References

[1] L. Adleman. Two theorems on random polynomial time. In *10th FOCS*, pages 75–83, 1978.

[2] C. Alvarez and R. Greenlaw. A compendium of problems complete for symmetric logarithmic space. ECCC report TR96-039, 1996.

[3] 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 *20th FOCS*, pages 218–223, 1979.

[4] R. Armoni, M. Saks, A. Wigderson and S. Zhou. Discrepancy sets and pseudorandom generators for combinatorial rectangles. In *37th FOCS*, pages 412-421, 1996.

[5] V. Arvind and J. Köbler. On pseudorandomness and resource-bounded measure. In *17th FSTTCS*, Springer-Verlag, LNCS 1346, pages 235–249, 1997.

[6] L. Babai. Trading Group Theory for Randomness. In *17th STOC*, pages 421–429, 1985.

[7] M. Blum and S. Micali. How to Generate Cryptographically Strong Sequences of Pseudo-Random Bits. *SICOMP*, Vol. 13, pages 850–864, 1984. Preliminary version in *23rd FOCS*, 1982.

[8] O. Goldreich, N. Nisan and A. Wigderson. On Yao's XOR-Lemma. *ECCC*, TR95-050, 1995.

[9] O. Goldreich, D. Ron and M. Sudan. Chinese Remaindering with Errors. TR98-062, available from *ECCC*, at `http://www.eccc.uni-trier.de/eccc/`, 1998.

[10] S. Goldwasser, S. Micali and C. Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SICOMP*, Vol. 18, pages 186–208, 1989. Preliminary version in *17th STOC*, 1985.

[11] R. Impagliazzo, V. Kabanets and A. Wigderson. In search of an easy witness: Exponential versus probabilistic time. In proceedings of *16th CCC*, pages 2–12, 2001.

[12] R. Impagliazzo and A. Wigderson. P=BPP if E requires exponential circuits: Derandomizing the XOR Lemma. In *29th STOC*, pages 220–229, 1997.

[13] R. Impagliazzo and A. Wigderson. Randomness vs. Time: De-randomization under a uniform assumption. In *39th FOCS*, pages 734–743, 1998.

[14] V. Kabanets. Easiness assumptions and hardness tests: Trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001.

[15] R.M. Karp and R.J. Lipton. Some connections between nonuniform and uniform complexity classes. In *12th STOC*, pages 302-309, 1980.

[16] A. Klivans and D. van Melkebeek. Graph Nonisomorphism has Subexponential Size Proofs Unless the Polynomial-Time Hierarchy Collapses. In *31st STOC*, pages 659–667, 1998. To appear in *SICOMP*.

[17] N. Nisan. Pseudorandom Generators for Space Bounded Computation. *Combinatorica*, Vol. 12 (4), pages 449–461, 1992.

[18] N. Nisan, E. Szemeredi, and A. Wigderson. Undirected connectivity in $O(log^{1.5}n)$ space. In *33rd FOCS*, pages 24-29, 1992.

[19] N. Nisan and A. Wigderson. Hardness vs Randomness. *JCSS*, Vol. 49, No. 2, pages 149–167, 1994.

[20] M.O. Rabin. Probabilistic Algorithm for Testing Primality. *Journal of Number Theory*, Vol. 12, pages 128–138, 1980.

[21] W.J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *JCSS*, Vol. 4 (2), pages 177-192, 1970.

[22] R. Shaltiel. A log-space extractor for high values of min-entropy. Personal communication, June 2002.

[23] M. Sudan. Decoding of Reed-Solomon codes beyond the error-correction bound. *Journal of Complexity*, Vol. 13 (1), pages 180–193, 1997.

[24] M. Sudan, L. Trevisan and S. Vadhan. Pseudorandom Generators without the XOR Lemma. *JCSS*, Vol. 62, No. 2, pages 236–266, 2001.

[25] A. Ta-Shma. Almost Optimal Dispersers. In *30th STOC*, pages 196–202, 1998.

[26] L. Trevisan. Constructions of Near-Optimal Extractors Using Pseudo-Random Generators. In *31st STOC*, pages 141–148, 1998.

[27] A.C. Yao. Theory and Application of Trapdoor Functions. In *23rd FOCS*, pages 80–91, 1982.

[28] D. Zuckerman. Randomness-Optimal Sampling, Extractors, and Constructive Leader Election. In *28th STOC*, pages 286–295, 1996.

Appendix: On the reducibility properties of the matrix problem

For simplicity, we work with finite fields of characteristic 2, rather than with finite fields of large prime cardinality. The issue at hand is that random self-reducibility for n-by-n matrices requires working with a field with more than n elements, and so the finite field cannot be fixed (but should rather depend on the parameter n). The first question is which field should we use (as a function of n), and the second question is how to support downward self-reducibility in case the field changes (with the dimension). One solution is to work with fields of prime cardinality, make the prime number an auxiliary input of the problem, and move among different primes via Chinese Remainder Theorem (cf. [9]). Here we take a different approach.

We consider the computation of the following function $f_n(A_1, ..., A_\ell) = \sum_{S\subseteq[\ell]} \det(\sum_{i\in S} A_i)$, where $\ell \stackrel{\rm def}{=} \ell(n)$ is logarithmic in n (i.e., $\ell(n) = \lceil c\log_2 n\rceil$ for some c), the A_i's are n-by-n matrices over $\mathrm{GF}(2^m)$, and $m \stackrel{\rm def}{=} m(n) = 2^{\lceil \log_2 n\rceil}$. (We ignore the issue of representing the field $\mathrm{GF}(2^m)$; that is, specifying an irreducible polynomial of degree m over $\mathrm{GF}(2)$. Such a polynomial can be added as an auxiliary parameter to f_n, or alternatively be fixed for specific values of m.)[15]

Random self-reducibility. We first show that the value of f_n at any instance can be computed from the value of f_n at $\mathrm{poly}(n)$ random instances. Specifically, we show that $f_n(A_1, ..., A_\ell)$ can be computed based on the values of $f_n(A_1 + \gamma_j R_1, \cdots, A_\ell + \gamma_j R_\ell)$, for $j = 1, ..., n+1$, where the R_i's are random matrices (and the γ_j's are distinct non-zero elements of $\mathrm{GF}(2^m)$). Recall that, for every two matrices A and B, it holds that $\det(A) = \sum_{j=1}^{n+1} c_j \det(A + \gamma_j B)$, where the c_j's depend only on the γ_j's. Using this fact (in the second equality), we obtain:

$$\begin{aligned}
f_n(A_1, ..., A_\ell) &= \sum_{S\subseteq[\ell]} \det\left(\sum_{i\in S} A_i\right) \\
&= \sum_{S\subseteq[\ell]} \sum_{j=1}^{n+1} c_j \cdot \det\left(\left(\sum_{i\in S} A_i\right) + \gamma_j \left(\sum_{i\in S} R_i\right)\right) \\
&= \sum_{j=1}^{n+1} c_j \cdot \sum_{S\subseteq[\ell]} \det\left(\sum_{i\in S} (A_i + \gamma_j R_i)\right) \\
&= \sum_{j=1}^{n+1} c_j \cdot f_n(A_1 + \gamma_j R_1, ..., A_\ell + \gamma_j R_\ell)
\end{aligned}$$

and the claim follows. It follows that given a procedure that computes f_n correctly on at least a $1-(1/3n)$ fraction of the instances, we can compute f_n correctly (w.v.h.p) on any instance. Actually, combined with the list-decoding algorithm of Sudan [23], the above random self-reducibility process implies a very strong worst-case/average-case connection.

Proposition 15 *Given a procedure that computes f_n correctly on an ϵ fraction of the instances, we can obtain a procedure that on input any instance outputs a list of size* $\mathrm{poly}(1/\epsilon)$ *containing the correct value of f_n.*

Proof: For fix sequences $A_1, ..., A_\ell$ and $R_1, ..., R_\ell$ of n-by-n matrices, and a variable x (ranging in the field), the function $f_n(A_1 + xR_1, ..., A_\ell + xR_\ell)$ is a degree n polynomial in x with free-term

[15] Specifically, for $m = 2 \cdot 3^i$, the polynomial $x^m + x^{m/2} + 1$ is irreducible over $\mathrm{GF}(2)$. Using these cases requires defining $m(n) = 2 \cdot 3^{\lceil \log_3 n\rceil}$ and slightly modifying the downwards self-reducibility process described below.

equal $f_n(A_1, ..., A_\ell)$. Selecting random matrices $R_1, ..., R_\ell$, the probability that the given procedure provides the correct answer to at least an $\epsilon/2$ fraction of the instances in $\{A_1 + eR_1, ..., A_\ell + eR_\ell\}$ is at least $\epsilon/2$. In such a case, using [23], we obtain a list of polynomials containing the correct one. ■

Downward self-reducibility. We next show that the value of f_n at any instance can be computed from the value of f_{n-1} at poly(n) instances. To prove that f_n is downward self-reducible, we expand the determinant (as usual) about the first row. Specifically, for every $n \times n$ matrix A it holds that $\det(A) = \sum_{j=1}^{n} a^j \det(A^j)$, where a^j denotes the $(1, j)$-entry of A, and A^j denotes the minor of A obtained by removing the first row and jth column. Thus:

$$
\begin{aligned}
f_n(A_1, ..., A_{\ell(n)}) &= \sum_{S \subseteq [\ell(n)]} \det\left(\sum_{i \in S} A_i\right) \\
&= \sum_{S \subseteq [\ell(n)]} \sum_{j=1}^{n} \left(\sum_{i \in S} a_i^j\right) \cdot \det\left(\sum_{i \in S} A_i^j\right) \\
&= \sum_{j=1}^{n} \sum_{i=1}^{\ell(n)} a_i^j \cdot \sum_{S \ni i} \det\left(\sum_{i' \in S} A_{i'}^j\right) \\
&= \sum_{j=1}^{n} \sum_{i=1}^{\ell(n)} a_i^j \cdot \left(\sum_{S \subseteq [\ell(n)]} \det\left(\sum_{i' \in S} A_{i'}^j\right) - \sum_{S \subseteq [\ell(n)] \setminus \{i\}} \det\left(\sum_{i' \in S} A_{i'}^j\right)\right) \\
&= \sum_{j=1}^{n} \sum_{i=1}^{\ell(n)} a_i^j \cdot \left(f_{n-1}(A_1^j, \ldots, A_{\ell(n)}^j) - f_{n-1}(A_1^j, \ldots, A_{i-1}^j, A_{i+1}^j, \ldots, A_{\ell(n)}^j)\right) \quad (1)
\end{aligned}
$$

where in Eq. (1) we have abused the notation f_{n-1}. Specifically, the function f_{n-1} has $\ell(n-1)$ arguments, whereas we have applied it once to $\ell(n)$ arguments and once to $\ell(n) - 1$ arguments. Furthermore, the function f_{n-1} is to be applied to matrices over $\mathrm{GF}(2^{m(n-1)})$, whereas we have applied it to matrices over $\mathrm{GF}(2^{m(n)})$. We address both problems next.

1. Note that $\ell(n-1) \in \{\ell(n), \ell(n) - 1\}$ (because $\ell(t) = \lceil c \log_2 t \rceil$). In case $\ell(n-1) = \ell(n)$, we replace the second term in Eq. (1) by $\frac{1}{2} f_{n-1}(A_1^j, \ldots, A_{i-1}^j, 0, A_{i+1}^j, \ldots, A_{\ell(n)}^j)$, using the fact that $\sum_{S \subseteq [\ell-1]} \det\left(\sum_{i \in S} M_i\right)$ equals $\frac{1}{2} \sum_{S \subseteq [\ell]} \det\left(\sum_{i \in S} M_i\right)$, where M_ℓ is the all-zero matrix. In case $\ell(n-1) = \ell(n) - 1$, we replace the first term in Eq. (1) by $f_{n-1}(A_1^j, A_2^j, \ldots, A_{\ell(n)-1}^j) + f_{n-1}(A_\ell^j, A_2^j, \ldots, A_{\ell(n)-1}^j) + f_{n-1}(A_1^j + A_\ell^j, A_2^j, \ldots, A_{\ell(n)-1}^j) - f_{n-1}(0, A_2^j, \ldots, A_{\ell(n)-1}^j)$, using the fact that

$$
\begin{aligned}
\sum_{S \subseteq [\ell]} \det\left(\sum_{i \in S} M_i\right) &= \sum_{S \subseteq \{2, \ldots, \ell-1\}} \left(\det\left(M_1 + \sum_{i \in S} M_i\right) + \det\left(\sum_{i \in S} M_i\right)\right) \\
&\quad + \sum_{S \subseteq \{2, \ldots, \ell-1\}} \left(\det\left(M_\ell + \sum_{i \in S} M_i\right) + \det\left(\sum_{i \in S} M_i\right)\right) \\
&\quad + \sum_{S \subseteq \{2, \ldots, \ell-1\}} \left(\det\left((M_1 + M_\ell) + \sum_{i \in S} M_i\right) + \det\left(\sum_{i \in S} M_i\right)\right)
\end{aligned}
$$

$$- \sum_{S \subseteq \{2,\ldots,\ell-1\}} \left(\det\left(0 + \sum_{i \in S} M_i\right) + \det\left(\sum_{i \in S} M_i\right)\right)$$

2. Note that $m(n-1) \in \{m(n), m(n)/2\}$ (because $m(t) = 2^{\lceil \log_2 t \rceil}$). In case $m(n-1) = m(n)$, which typically holds, the above description is accurate. The problematic case is of $m(n) = 2m(n-1)$, in which case we have to reduce determinants over $\mathrm{GF}(2^{2m})$ to determinants over $\mathrm{GF}(2^m)$. This can be done by viewing $\mathrm{GF}(2^{2m})$ as an extension field of $\mathrm{GF}(2^m)$. Specifically, we view $\mathrm{GF}(2^{2m})$ as the field of linear polynomials (in x) over $\mathrm{GF}(2^m)$, consider the determinant of an k-by-k matrix over $\mathrm{GF}(2^{2m})$ as a degree k polynomial over $\mathrm{GF}(2^m)$, and compute this polynomial by extrapolation from $k+1$ points in $\mathrm{GF}(2^m)$. Reducing the resulting polynomial modulo the irreducible polynomial representing $\mathrm{GF}(2^{2m})$ as an extension of $\mathrm{GF}(2^m)$, we obtain the value of the determinant.