Bravely, Moderately:
A Common Theme in Four Recent Works

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science
Rehovot, Israel.
`oded.goldreich@weizmann.ac.il`

August 28, 2005

Abstract

We highlight a common theme in four relatively recent works that establish remarkable results by an iterative approach. Starting from a trivial construct, each of these works applies an ingeniously designed sequence of iterations that yields the desired result, which is highly non-trivial. Furthermore, in each iteration, the construct is modified in a relatively moderate manner. The four works we refer to are

1. the polynomial-time approximation of the permanent of non-negative matrices (by Jerrum, Sinclair, and Vigoda, *33rd STOC*, 2001);
2. the iterative (Zig-Zag) construction of expander graphs (by Reingold, Vadhan, and Wigderson, *41st FOCS*, 2000);
3. the log-space algorithm for undirected connectivity (by Reingold, *37th STOC*, 2005);
4. and, the alternative proof of the PCP Theorem (by Dinur, *ECCC*, TR05-046, 2005).

Keywords: Approximation, Expander Graphs, Log-Space, Markov Chains, NP, Permanent, PCP, Space Complexity, Undirected Connectivity.

Contents

Introduction

Speude bradeos.[1]

The title of this essay employs more non-technical terms than one is accustomed to encounter in the title of a technical survey, let alone that some are rarely used in a technical context. Indeed, this is an unusual survey, written in an attempt to communicate a feeling that cannot be placed on sound ground. The feeling is that there is a common theme among the works to be reviewed here, and that this common theme is intriguing and may lead to yet additional important discoveries. We hope that also readers that disagree with the foregoing feeling may benefit from the perspective offered by lumping the said works together and highlighting a common theme.

We are going to review four celebrated works, each either resolving a central open problem or providing an alternative proof for such a central result. The common theme that we highlight is the (utmost abstract) attitude of these works towards solving the problem that they address. Rather than trying to solve the problem by one strong blow, each of these works goes through a long sequence of iterations, gradually transforming the original problem into a trivial one. (At times, it is more convenient to view the process as proceeding in the opposite direction; that is, gradually transforming a solution to the trivial problem into a solution to the original problem.) Anyhow, each step in this process is relatively simple (in comparison to an attempt to solve the original problem at one shot), and it is the multitude of iterated steps that does the job. Let us try to clarify the foregoing description by providing a bird's eye view of each of these works.

A bird's eye view of the four works

Following are very high level outlines of the aforementioned works. At this point we avoid almost all details (including crucial ones), and refrain from describing the context of these works (i.e., the history of the problems that they address). Instead, we focus on the iterative processes eluded to above. More detailed descriptions as well as comments about the history of the problems are to be found in corresponding sections of this essay.

Approximating the permanent of non-negative matrices. The probabilistic polynomial-time approximation algorithm of Jerrum, Sinclair, and Vigoda [17] is based on the follow-

[1]This Ancient Greek proverb, reading *hasten slowly*, is attributed to Augustus; see C. Suetonius Tranquillus, *D. Octavius Caesar Augustus*, paragraph XXV. The intention seems to be a calling for action that is marked by determination and thoroughness, which characterizes the "moderate revolution" of Rome under Augustus.

ing observation: knowing (approximately) certain parameters of a non-negative matrix M allows to approximate the same parameters for a matrix M', provided that M and M' are sufficiently similar. Specifically, M and M' may differ only on a single entry, and the ratio of the corresponding values must be sufficiently close to one. Needless to say, the actual observation (is not generic but rather) refers to specific parameters of the matrix, which include its permanent. Thus, given a matrix M for which we need to approximate the permanent, we consider a sequence of matrices $M_0, ..., M_t \approx M$ such that M_0 is the all 1's matrix (for which it is easy to evaluate the said parameters), and each M_{i+1} is obtained from M_i by reducing some adequate entry by a factor sufficiently close to one. This process of (polynomially many) gradual changes, allows to transform the dummy matrix M_0 into a matrix M_t that is very close to M (and hence has a permanent that is very close to the permanent of M). Thus, approximately obtaining the parameters of M_t allows to approximate the permanent of M.

The iterative (Zig-Zag) construction of expander graphs. The construction of constant-degree expander graphs by Reingold, Vadhan, and Wigderson [25] proceeds in iterations. Its starting point is a very good expander G of constant size, which may be found by exhaustive search. The construction of a large expander graph proceeds in iterations, where in the i^{th} iteration the current graph G_i and the fixed graph G are combined (via a so-called Zig-Zag product) to obtain the larger graph G_{i+1}. The combination step guarantees that the expansion property of G_{i+1} is at least as good as the expansion of G_i, while G_{i+1} maintains the degree of G_i and is a constant times larger than G_i. The process is initiated with $G_1 = G^2$, and terminates when we obtain a graph of approximately the desired size (which requires a logarithmic number of iterations). Thus, the last graph is a constant-degree expander of the desired size.

The log-space algorithm for undirected connectivity. The aim of Reingold's algorithm [24] is to (deterministically) traverse an arbitrary graph using logarithmic amount of space. Its starting point is the fact that any expander is easy to traverse in deterministic logarithmic-space, and thus the algorithm gradually transforms any graph into an expander, while maintaining the ability to map a traversal of the latter into a traversal of the former. Thus, the algorithm traverses a virtual graph, which being an expander is easy to traverse in deterministic logarithmic-space, and maps the virtual traversal of the virtual graph to a real traversal of the actual input graph. The virtual graph is constructed in (logarithmically many) iterations, where in each iteration the graph becomes easier to traverse. Specifically, in each iteration the expansion property of the graph improves by a constant factor, while the graph itself only grows by a constant factor, and each iteration can be performed (or rather emulated) in constant space. Since each graph has some noticeable expansion (i.e., expansion inversely related to the size of the graph), after logarithmically many steps this process yields a good expander (i.e., constant expansion).

The alternative proof of the PCP Theorem. Dinur's new approach [12] to the proof of the PCP Theorem is based on gradually improving the performance of PCP-like systems. The starting point is a trivial PCP-like system that detects error with very small but noticeable probability. Each iterative step increases the detection probability of the system by a constant factor, while incurring only a small overhead in other parameters (i.e., the randomness complexity increases by a constant term). Thus, the PCP Theorem (asserting constant detection probability for $\mathcal{NP}$) is obtained after logarithmically many such iterative steps. Indeed, the heart of this approach is the detection amplification step, which may be viewed as simple only in comparison to the original proof of the PCP Theorem.

An attempt to articulate the thesis

The current section will contain an attempt to articulate the thesis that there is a common theme among these works. Readers that do not care about philosophical discussions (and other attempts to say what cannot be said) are encouraged to skip this subsection. In order to emphasize the subjective nature of this section, it is written in first person singular.

I will start by saying a few works about bravery and moderation. I consider as brave the attempt to resolved famous open problems or provide alternative proofs for central celebrated results.[2] To try a totally different approach is also brave, and so is realizing one's limitations and trying a moderate approach: rather than trying to resolve the problem in a single blow, one wisely designs a clever scheme that gradually progresses towards the desired goal. Indeed, this is the victory of moderation.

Getting to the main thesis of this essay (i.e., the existence of a common theme among the reviewed works), I believe that I have already supported a minimalistic interpretation of this thesis by the foregoing bird's eye view of the four works. That is, there is an obvious similarity among these bird's eye views. However, some researchers may claim (and indeed

[2] Consider the problems addressed by the four reviewed works: The problem of approximating the permanent was open since Valiant's seminal work [27] and has received considerable attention since Broder's celebrated work [11]. Constructions of expander graphs were the focus of much research since the 1970's, and were typically based on non-elementary mathematics (cf. [20, 15, 19]). The existence of deterministic log-space algorithms for undirected connectivity has been in the focus of our community since the publication of the celebrated randomized log-space algorithm of Aleliunas *et. al.* [1]. The PCP Theorem, proved in the early 1990's [5, 6], is closely related (via [14, 5]) to the study of approximation problems (which dates to the early 1970's).

have claimed) that this similarity extends also to numerous other works and to various other types of iterative procedures. This is the claim I wish to oppose here: I believe that the type of iterative input-modification process that underlies the aforementioned works is essentially novel and amounts to a new algorithmic paradigm.

Let me first give a voice to the skeptic. For example, Amnon Ta-Shma, playing the Devil's advocate, claims that many standard iterative procedures (e.g., repeated squaring) may be viewed as "iteratively modifying the input" (rather than iteratively computing an auxiliary function of it, as I view it). Indeed, the separation line between input-modification and arbitrary computation is highly subjective, and I don't believe that one can rigorously define it. Nevertheless, rejecting Wittgenstein's advice [28, §7], I will try to speak about it.

My claim is that (with the exception of the iterative expander construction of [25]) the reviewed works do not output the modified input, but rather a function of it, and they modify the input in order to ease the computation of the said function. That is, whereas the goal was to compute a function of the original input, they compute a function of the final modified input, and obtain the originally desired value (of the function evaluated at the original input) by a process that relies on the relatively simplicity of the intermediate modifications. The line that I wish to draw is between *iteratively producing modified inputs* (while maintaining a relation between the corresponding outputs) and *iteratively producing better refinements of the desired output while keeping the original input intact*. Indeed, I identify the latter with standard iterative processes (and the former with the common theme of the four reviewed works).

My view is that in each of these works, *the input itself undergoes a gradual transformation* in order to ease some later process. This is obvious in the case of approximating the permanent [17] and in the case of traversing a graph in log-space [24], but it is also true with respect to the other two cases: In Dinur's proof [12] of the PCP Theorem the actual iterative process consists of a sequence of Karp-reductions (which ends with a modified instance that has a simple PCP system), and in the iterative construction of expanders [25] the size of the desired expander increases gradually. In contrast, in typical proofs by induction, it is the problem itself that gets modified, whereas standard iterative procedures refer to sub-problems that relate to auxiliary constructs. Indeed, the separation line between the iterative construction of expanders and standard iterative analysis is the thinnest, but the similarity between it and the results of Reingold [24] and Dinur [12] may appeal to the skeptic.

I wish to stress that the aforementioned iterative process that gradually transforms the input is marked by the relative simplicity of each iteration, especially in comparison to the full-fledged task being undertaken. In the case of Reingold's log-space algorithm [24], each iteration needs to be implemented in constant amount of space, which is indeed a good indication to its simplicity. In the case of the approximation of the permanent [17], each iteration is performed by a modification of a known algorithm (i.e., of [16]). In the iterative construction of expanders [25], a graph powering and a new type of graph product are used and analyzed, where the analysis is simple in comparison to either of [20, 15, 19]. Lastly, in Dinur's proof [12] of the PCP Theorem, each iteration is admittingly quite complex, but not when compared to the original proof of the PCP Theorem [6, 5].

The similarity among the iterated Zig-Zag construction of [25], the log-space algorithm for undirected connectivity of [24], and the new approach to the PCP Theorem of [12] has been noted by many researchers (see, e.g., [24, 12] themselves). However, I think that the noted similarity was more technical in nature, and was based on the role of expanders and "Zig-Zag like" operations in these works. In contrast, my emphasis is on the sequence of gradual modifications, and thus I view the permanent approximator of [17] just as close in spirit to these works. In fact, as is hinted in the foregoing discussion, I view [24, 12] as closer in spirit to [17] than to [25].

1 Approximating the permanent of non-negative matrices

The permanent of a n-by-n matrix $(a_{i,j})$ is the sum, taken over all permutations $\pi : [n] \to [n]$, of $\prod_{i=1}^{n} a_{i,\pi(i)}$. Although defined very similarly to the determinant (i.e., just missing the minus sign in half the terms), the permanent seems to have a totally different complexity than the determinant. In particular, in a seminal work [27], Valiant showed that the permanent is $\#\mathcal{P}$-complete; that is, counting the number of solutions to any NP-problem is polynomial-time reducible to computing the permanent of 0-1 matrices, which in turn counts the number of perfect matchings in the corresponding bipartite graph. Furthermore, the reduction to the permanent of integer matrices preserves the number of solutions (when normalized by an easy to compute factor), and hence approximating the permanent of such matrices seems infeasible (as it will imply $\mathcal{P} = \mathcal{NP}$). It was noted that the same does not hold for 0-1 matrices (or even non-negative matrices). In fact, Broder's celebrated work [11] introduced an approach having the potential to yield efficient algorithms for approximating the permanent of non-negative matrices. Fifteen years later, this potential was fulfilled by Jerrum, Sinclair, and Vigoda, in a work [17] to be reviewed here.

The algorithm of Jerrum, Sinclair, and Vigoda [17] follows the general paradigm of Broder's work (which was followed by all subsequent works in the area): The approach is based on the relation between *approximating* the ratio of perfect and nearly perfect matching of a graph and *sampling* uniformly a perfect or nearly perfect matching of a graph, where nearly perfect matchings are matchings that leave unmatched a single pair of vertices. In order to perform the aforementioned sampling, one sets-up a Markov Chain with states corresponding to the set of perfect and nearly perfect matching of the graph. The transition probability of the Markov Chain maps each perfect matching to a nearly perfect matching obtained by omitting a uniformly selected edge (in the perfect matching). The transition from a nearly perfect matching that misses the vertex pair (u, v) is determined by selecting a random vertex z, adding (u, v) to the matching if $z \in \{u, v\}$ and (u, v) is an edge of the graph, and adding (u, z) to the matching and omitting (x, z) from it if $z \notin \{u, v\}$ and (u, z) is an edge of the graph. By suitable modification, the stationary distribution of the chain equals the uniform distribution over the set of perfect and nearly perfect matchings of the graph. The stationary distribution of the chain is approximately sampled by starting from an arbitrary state (e.g., any perfect matching) and taking a sufficiently long walk on the chain.

This approach depends on the mixing time of the chain (i.e., the number of steps needed to get approximately close to its stationary distribution), which in turn is linearly related to the ratio of nearly perfect to perfect matchings in the underlying graph (see, [16]). (Note that the later ratio also determines the complexity of the reduction from approximating this ratio to sampling the stationary distribution of the chain.) When the latter ratio is polynomial, this approach yields a polynomial-time algorithms, but it is easy to see that there are graphs for which the said ratio is exponential. One key observation of [17] is that the latter problem can be fixed by introducing auxiliary weights that when applied to all nearly perfect matchings yield that the set of perfect matching has approximately the same probability mass (under the stationary distribution) as the set of nearly perfect matchings. Specifically, for each pair (u, v) we consider a weight $w(u, v)$ such that the probability mass assigned to perfect matchings approximately equals $w(u, v)$ times the probability mass assigned to nearly perfect matchings that leaves the vertices u and v unmatched. Needless to say, in order to determine the corresponding weights, one needs to know the corresponding ratios, which seems to lead to a vicious cycle.

Here is where the main idea of [17] kicks in: *knowing the approximate sizes of the perfect and nearly perfect matching in a graph G allows to obtain these parameters for a related graph G' that is closed to G, by running the Markov Chain that corresponds to G' using weights as determined for G.* This observation is the basis of the iterative process outlined in the Introduction: We start with a trivial graph G_0 for which the said quantities are easy to determine, and consider a sequence of graphs $G_1, ..., G_t$ such that G_{i+1} is sufficiently close to G_i, and G_t is sufficiently close to the input graph. We approximate the said quantities for G_{i+1} using the estimated quantities for G_i, and finally obtain an approximation of the number of perfect matchings in the input graph. The algorithm actually works with weighted graphs, where the weight of a matching is the product of the weights of the edges in the matching. We start with G_0 that is a complete graph (i.e., all edges are present, each at weight 1), and let G_{i+1} be a graph obtained from G_i by reducing the weight of one of the non-edges of the input graph by a factor of $\rho = 9/8$. Using such a sequence, for $t = \widetilde{O}(n^3)$, we can obtain a graph G_t in which the edges of the input graph have weight 1 while non-edges of the input graph have weight lower than $1/(n!)$. Approximating the total weight of the weighted perfect matchings in G_t provides the desired approximation to the input graph.

Digest. The algorithm of Jerrum, Sinclair, and Vigoda [17] proceeds in iterations, using a sequence of weighted graphs $G_0, ..., G_t$ such that G_0 is the complete (unweighted) graph, G_{i+1} is a sufficiently close approximation of G_i, and G_t is a sufficiently close approximation to the input graph. We start knowing the number of perfect and nearly perfect matchings in G_0 (which is easily determined by the number of vertices). In the i^{th} iteration, using approximations for the number of perfect and nearly perfect matchings in G_i, we compute such approximations for G_{i+1}. These approximations are obtained by running an adequate Markov Chain, and the fact that we only have approximations for the quantities of G_i merely effects the mixing time of the chain (in a non-significant way). Thus, gradually transforming a dummy graph G_0 into the input graph, we obtain approximations to relevant parameters of all the graphs, where the approximated parameters of G_i allow us to obtain the approximated parameters of G_{i+1}, and the approximated parameters of G_t include an approximation of the number of perfect matchings in the input graph.

Comment. We mention that a different iterative process related to the approximation of the permanent was previously studied in [18]. In that work, an input matrix is transformed to an approximately Doubly Stochastic (aDS) matrix, by iteratively applying row and column scaling operations, whereas for any aDS n-by-n matrix the permanent is at least $\Omega(\exp(-n))$ and at most 1.

2 The iterative (Zig-Zag) construction of expander graphs

By expander graphs (or expanders) of degree d and eigenvalue bound $\lambda < d$, we mean an infinite family of d-regular graphs, $\{G_n\}_{n\in S}$ ($S \subseteq \mathsf{N}$), such that G_n is a d-regular graph over n vertices and the absolute value of all eigenvalues, save the biggest one, of the adjacency matrix of G_n is upper-bounded by λ. This algebraic definition is related to the combinatorial definition of expansion in which one requires that any (not too big) set of vertices in the graph have relatively a large set of strict neighbors (i.e., is "expanding"): See [3] and [2]. It is often more convenient to refer to the relative eigenvalue bound defined as λ/d.

We are interested in *explicit constructions* of expander graphs, where the minimal notion of explicitness requires that the graph be constructed in time polynomial in its size (i.e., there exists a polynomial time algorithm that, on input 1^n, outputs G_n).[3] A *stronger notion of explicitness* requires that there exists a polynomial-time algorithm that on input n (in binary), a vertex $v \in G_n$ and an index $i \in [d] \stackrel{\rm def}{=} \{1,...,d\}$, returns the $i^{\rm th}$ neighbor of v. Many explicit constructions of expanders were given, starting in [20] (where S is the set of all quadratic integers), and culminating in the optimal construction of [19] (where $\lambda = 2\sqrt{d-1}$ and S is somewhat complex). These constructions are quite simple to present, but their analysis is based on non-elementary results from various branches of mathematics. In contrast, the following construction of Reingold, Vadhan, and Wigderson [25] is based on an iterative process, and its analysis is based on a relatively simple algebraic fact regarding the eigenvalues of matrices.

The starting point of the construction (i.e., the base of the iterative process) is a very good expander G of *constant size*, which may be found by an exhaustive search. The construction of a large expander graph proceeds in iterations, where in the $i^{\rm th}$ iteration the graphs G_i and G are combined to obtain the larger graph G_{i+1}. The combination step guarantees that the expansion property of G_{i+1} is at least as good as the expansion of G_i, while G_{i+1} maintains the degree of G_i and is a constant times larger than G_i. The process is initiated with $G_1 = G^2$ and terminates when we obtain a graph G_t of approximately the desired size (which requires a logarithmic number of iterations).

The heart of the combination step is a new type of "graph product" called *Zig-Zag product*. This operation is applicable to any pair of graphs $G = ([D], E)$ and $G' = ([N], E')$, provided that G' (which is typically larger than G) is D-regular. For simplicity, we assume that G is d-regular (where typically $d \ll D$). The Zig-Zag product of G' and G, denoted $G' \textcircled{z} G$, is defined as a graph with vertex set $[N] \times [D]$ and an edge set that includes an edge

[3] We also require that the set S for which G_n's exist is sufficiently "tractable": say, that given any $n \in \mathsf{N}$ one may efficiently find $s \in S$ so that $n \leq s < 2n$.

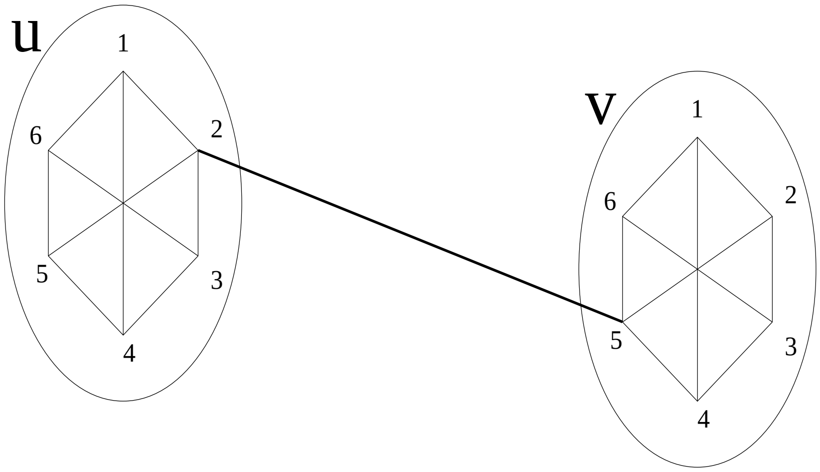

Figure 1: Extract of a zig-zag produce of G' with the 3-regular graph G having six vertices. In G' the 2nd edge of u is incident at v, as its 5th edge.

between $\langle u,i\rangle \in [N]\times[D]$ and $\langle v,j\rangle$ if and only if $(i,k),(\ell,j)\in E$ and the $k^{\rm th}$ edge incident at u equals the $\ell^{\rm th}$ edge incident at v. (See Figure 1, where vertex $\langle u,3\rangle$ is connected in $G'ⓩG$ to $\langle v,2\rangle$, using the G-edges $(3,2)$ and $(5,2)$.) It will be convenient to represent graphs like G' by their edge rotation function[4], denoted $R' : [N]\times[D] \to [N]\times[D]$, such that $R'(u,i) = (v,j)$ if (u,v) is the $i^{\rm th}$ edge incident at u as well as the $j^{\rm th}$ edge incident at v. For simplicity, we assume that G is edge-colorable with d colors, which in turn yields a natural edge rotation function (i.e., $R(i,\alpha) = (j,\alpha)$ if the edge (i,j) is colored α). We will denote by $E_\alpha(i)$ the vertex reached from $i\in[D]$ by following the edge colored α (i.e., $E_\alpha(i) = j$ iff $R(i,\alpha) = (j,\alpha)$). The Zig-Zag product of G' and G, denoted $G'ⓩG$, is then defined as a graph with the vertex set $[N]\times[D]$ and the edge rotation function

$$(\langle u,i\rangle,\langle\alpha,\beta\rangle) \mapsto (\langle v,j\rangle,\langle\beta,\alpha\rangle) \quad \text{if } R'(u,E_\alpha(i)) = (v,E_\beta(j)). \tag{1}$$

That is, edges are labeled by pairs over $[d]$, and the $\langle\alpha,\beta\rangle^{\rm th}$ edge out of vertex $\langle u,i\rangle\in[N]\times[D]$ is incident at the vertex $\langle v,j\rangle$ (as its $\langle\beta,\alpha\rangle^{\rm th}$ edge) if $R(u,E_\alpha(i)) = (v,E_\beta(j))$. (That is, based on $\langle\alpha,\beta\rangle$, we take a G-step from $\langle u,i\rangle$ to $\langle u,E_\alpha(i)\rangle$, then viewing $\langle u,E_\alpha(i)\rangle \equiv (u,E_\alpha(i))$ as an edge of G' we rotate it to $(v,j') \stackrel{\rm def}{=} R'(u,E_\alpha(i))$, and take a G-step from $\langle v,j'\rangle$ to $\langle v,E_\beta(j')\rangle$, while defining $j = E_\beta(j')$ and using $j' = E_\beta(E_\beta(j')) = E_\beta(j)$.)

[4]In [25] (and [24]) these functions are called rotation maps. As these functions are actually involutions (i.e., $R(R(x)) = x$ for every $x\in[N]\times[D]$), one may prefer terms as "edge rotation permutations" or "edge rotation involutions".

Clearly, the graph $G' \textcircled{z} G$ is d^2-regular and has $D \cdot N$ vertices. The key fact, proved in [25], is that the relative eigenvalue of the zig-zag produce is upper-bounded by the sum of the relative eigenvalues of the two graphs (i.e., $\lambda(G' \textcircled{z} G) \leq \lambda(G') + \lambda(G)$, where $\lambda(\cdot)$ denotes the relative eigenvalue of the relevant graph).[5]

The iterated expander construction uses the aforementioned zig-zag product as well as graph squaring. Specifically, the construction starts with the d^2-regular graph $G_1 = G^2 = ([D], E^2)$, where $D = d^4$ and $\lambda(G) < 1/4$, and proceeds in iterations such that $G_{i+1} = G_i^2 \textcircled{z} G$ for $i = 1, 2, ..., t-1$. That is, in each iteration, the current graph is first squared and then composed with the fixed graph G via the zig-zag product. This process maintains the following two invariants:

1. The graph G_i is d^2-regular and has D^i vertices.

   (This follows from the fact that a zig-zag product with the d-regular graph always yields a d^2-regular graph.)

2. The relative eigenvalue of G_i is smaller than one half.

   (Here we use the fact that $\lambda(G_{i-1}^2 \textcircled{z} G) \leq \lambda(G_{i-1}^2) + \lambda(G)$, which in turn equals $\lambda(G_{i-1})^2 + \lambda(G) < (1/2)^2 + (1/4)$. Note that graph squaring is used to reduce the relative eigenvalue of G_i before increasing it by zig-zag product with G.)

To ensure that we can construct G_i, we should show that we can actually construct the edge rotation function that correspond to its edge set. This boils down to showing that, given the edge rotation function of G_{i-1}, we can compute the edge rotation function of G_{i-1}^2 as well as of its zig-zag product with G. Note that this computation amounts to two recursive calls to computations regarding G_{i-1} (and two computations that correspond to the constant graph G). But since the recursion is logarithmic in the size of the final graph, the time spend in the recursive computation is polynomial in the size of the final graph. This suffices for the minimal notion of explicitness, but not for the stronger one.

To achieve a *strongly explicit construction*, we slightly modify the iterative construction. Rather than letting $G_{i+1} = G_i^2 \textcircled{z} G$, we let $G_{i+1} = (G_i \times G_i)^2 \textcircled{z} G$, where $G' \times G'$ denotes the tensor product of G' with itself (i.e., if $G' = (V', E')$ then $G' \times G' = (V' \times V', E'')$, where $E'' = \{(\langle u_1, u_2 \rangle, \langle v_1, v_2 \rangle) : (u_1, v_1), (u_2, v_2) \in E'\}$ with an edge rotation function $R''(\langle u_1, u_2 \rangle, \langle i_1, i_2 \rangle) = (\langle v_1, v_2 \rangle, \langle j_1, j_2 \rangle)$ where $R'(u_1, i_1) = (v_1, j_1)$ and $R'(u_2, i_2) = (v_2, j_2)$). (We still use $G_1 = G^2$.) Using the fact that tensor product preserves the relative eigenvalue and using a d-regular $G = ([D], E)$ with $D = d^8$, we note that the modified $G_i = (G_{i-1} \times G_{i-1})^2 \textcircled{z} G$ is a d^2-regular graph with $(D^{2^{i-1}-1})^2 \cdot D = D^{2^i - 1}$

[5]In fact, a stronger upper-bound is proved in [25].

vertices, and $\lambda(G_i) < 1/2$ (because $\lambda((G_{i-1} \times G_{i-1})^2 \textcircled{z} G) \le \lambda(G_{i-1})^2 + \lambda(G)$). Computing the neighbor of a vertex in G_i boils down to a constant number of such computations regarding G_{i-1}, but due to the tensor product operation the depth of the recursion is only double-logarithmic in the size of the final graph (and hence logarithmic in the length of the description of vertices in it).

Digest. In the first construction, the zig-zag product was used both in order to increase the size of the graph and to reduce its degree. However, as indicated by the second construction (where the tensor product of graphs is the main vehicle for increasing the size of the graph), the primary effect of the zig-zag product is to reduce the degree, and the increase in the size of the graph is merely a side-effect (which is actually undesired in Section 3). In both cases, graph squaring is used in order to compensate for the modest increase in the relative eigenvalue caused by the zig-zag product. In retrospect, the second construction is the "correct" one, because it decouples three different effects, and uses a natural operation to obtain each of them: Increasing the size of the graph is obtained by tensor product of graphs (which in turn increases the degree), a degree reduction is obtained by the zig-zag product (which in turn increases the relative eigenvalue), and graph squaring is used in order to reduce the relative eigenvalue.

A second theme. In continuation to the previous comment, we note that the successive application of several operations, each improving a different parameter (while not harming too much the others), reappears in the works of Reingold [24] and Dinur [12]. This theme has also appeared before in several other works (including [6, 5, 13]).[6]

[6] We are aware of half a dozen of other works, but guess that they are many more. We choose to cite here only works that were placed in the reference list for other reasons. Indeed, this second theme appears very clearly in PCP constructions (e.g., first optimizing randomness at the expense of number of queries and then reducing the latter at the expense of a bigger alphabet (not to mention the very elaborate combination in [13])).

3 The log-space algorithm for undirected connectivity

For more than two decades, undirected connectivity was one of the most appealing examples of the computational power of randomness. Whereas every graph can be efficiently traversed by a deterministic algorithm, the classical (deterministic) linear-time algorithms (e.g., BFS and DFS) require an extensive use of (extra) memory (i.e., space linear in the size of the graph). On the other hand, in 1979 Aleliunas *et. al.* [1] showed that, with high probability, a random walk of polynomial length visits all vertices (in the corresponding connected component). Thus, the randomized algorithm requires a minimal amount of auxiliary memory (i.e., logarithmic in the size of the graph). In the early 1990's, Nisan [21, 22] showed that any graph can be traversed in polynomial-time and poly-logarithmic space, but despite more than a decade of research attempts (see, e.g., [4]), a significant gap remained between the space complexity of randomized and deterministic polynomial-time algorithms for this natural and ubiquitous problem. This gap was recently closed by Reingold, in a work [24] reviewed next.

Reingold presented a deterministic polynomial-time algorithm that traverses any graph while using a logarithmic amount of auxiliary memory. His algorithm is based on a novel approach that departs from previous attempts, which tried to derandomize the random-walk algorithm. Instead, Reingold's algorithm traverses a virtual graph, which (being an expander) is easy to traverse (in deterministic logarithmic-space), and maps the virtual traversal of the virtual graph to a real traversal of the actual input graph. The virtual graph is constructed in (logarithmically many) iterations, where in each iteration the graph becomes easier to traverse. Specifically, in each iteration, each connected component of the graph becomes closer to a constant-degree expander in the sense that (the graph has constant degree and) the gap between its relative eigenvalue and 1 doubles.[7] Hence, after logarithmically many iterations, each connected component becomes a constant-degree expander, and thus has logarithmic diameter. Such a graph is easy to traverse deterministically using logarithmic space (e.g., by scanning all paths of logarithmic length going out of a given vertex, while noting that each such path can be represented by a binary string of logarithmic length).

The key point is to maintain the connected components of the graph while making each of them closer to an expander. Towards this goal, Reingold applies a variant of the iterated zig-zag construction (presented in Section 2), starting with the input graph, and iteratively composing the current graph with a constant-size expander. Details follow.

For adequate positive integers d and c, we first transform the actual input graph into a d^2-regular graph (e.g., by replacing each vertex v with a (multi-edge) cycle C_v and using

[7] See Section 2 for definition of expander and its relative eigenvalue.

each vertex on C_v to take care of an edge incident to v). Denoting the resulting graph by $G_1 = (V_1, E_1)$, we go through a logarithmic number of iterations letting $G_{i+1} = G_i^c ⓩ G$ for $i = 1, ..., t-1$, where G is a fixed d-regular graph with d^{2c} vertices. Thus, G_i is a d^2-regular graph with $d^{2c \cdot i} \cdot |V_1|$ vertices, and $1 - \lambda(G_i) > \max(2(1 - \lambda(G_{i-1})), 1/2)$, where the latter upper-bound on $\lambda(G_i)$ is due to [25]. We infer that $1 - \lambda(G_i) > \max(2^i \cdot (1 - \lambda(G_1)), 1/2)$, and using the fact that $\lambda(G_1) < 1 - (1/\text{poly}(|V_1|))$, which holds for any connected and non-bipartite graph, it follows that $\lambda(G_t) < 1/2$ for $t = O(\log |V_1|)$. (Indeed, it is instructive to assume throughout the analysis that (the original input and thus) G_1 is connected, and to guaranteed that it is non-bipartite (e.g., by adding self-loops).)

One detail of crucial importance is the ability to transform G_1 into G_t via a log-space computation. It is easy to see that the transformation of G_i to G_{i+1} can be performed in constant-space, but the standard composition lemma for space complexity incurs a logarithmic space overhead per each composition (and thus cannot be applied here). Still, taking a closer look at the transformation of G_i to G_{i+1}, one may note that it is highly structured and supports a stronger composition result that incurs only a constant space overhead per composition. An alternative implementation, outlined in [24], is obtained by unraveling the composition. The details of these alternative implementations are beyond the scope of the current essay.[8]

A minor variant. It is simpler to present a direct implementation of a minor variant of the above process. Specifically, rather than using the zig-zag product $G' ⓩ G$ (of Section 2), one may use the replacement product $G' ⓡ G$ defined as follows for a D-regular graph $G' = (V', E')$ and a d-regular graph $G = ([D], E)$:[9] The resulting $2d$-regular graph has vertex set $V' \times [D]$ and the following edge rotation function (which actually induces an

[8]We cannot refrain from saying that we prefer an implementation based on composition, and provide a few hints regarding such an implementation. Firstly, we suggest to consider the task of computing the neighbor of a given vertex in G_i, where the original graph is viewed as an oracle and the actual input is the aforementioned vertex. This computation can be performed by a constant-space oracle machine provided that its queries are answered by a similar machine regarding G_{i-1}. (A proper formulation requires read-only and write-only oracle tapes of length bounded by the actual input (i.e., vertex) length, where these tapes are not accounted in the space complexity.) Applying standard composition for this computation introduces an overhead that is logarithmic in the length of the input (i.e., the vertex name), which is double-logarithmic in the size N of the input graph. Thus, we obtain a deterministic algorithm using space $O(\log N \log\log N)$. Getting rid of the excessive double-logarithmic factor requires introducing a model of "shared memory for procedural calls" (including "shared pointers" for such calls).

[9]Since this product yields a $2d$-regular graph, in the context of the log-space algorithm one should set $D = (2d)^c$.

edge coloring)

$$
\begin{array}{c}
(\langle u,i\rangle,\langle 0,\alpha\rangle) \mapsto (\langle u,E_\alpha(i)\rangle,\langle 0,\alpha\rangle) \\
\text{and} \\
(\langle u,i\rangle,\langle 1,\alpha\rangle) \mapsto (R'(u,i),(1,\alpha)),
\end{array}
\tag{2}
$$

where E_α is as in Section 2. That is, every $\langle u,i\rangle \in V' \times [D]$ has d incident edges that correspond to the edges incident at i in G, and d parallel copies of the i^{th} edge of u in G'. It can be shown that, in the relevant range of parameters, the replacement product effect the eigenvalues in a way that is similar to the affect of the zig-zag product (because the two resulting graphs are sufficiently related).

A major variant. A more significant variant on the construction was recently presented in [26]. As a basic composition, they utilize a derandomized graph squaring of a large D-regular graph $G' = (V', E')$ using a d-regular (expander) graph $G = ([D], E)$: Unlike the previous composition operations, the resulting graph, which is a subgraph of the square of G', has V' itself as the vertex set but the edge density of the resulting graph is larger than that of G'. Specifically, the edge rotation function is

$$
(u,\langle i,\alpha\rangle) \mapsto (v,\langle j,\alpha\rangle) \quad \text{if } R'(u,i)=(w,k) \text{ and } R'(w,E_\alpha(k))=(v,j). \tag{3}
$$

where E_α is as in Section 2. That is, the edge set contains a subset of the edges of the standard graph square, where this subset corresponds to the edges of the small (expander) graph G. It can be shown that the derandomized graph squaring effect the eigenvalues in a way that is similar to the combination of squaring and zig-zag product, but the problem is that the edge density does not remain constant through the iterated procedure. Nevertheless, two alternatives ways of obtaining a log-space algorithm are known, one of which is presented in [26].

4 The alternative proof of the PCP Theorem

The PCP Theorem [5, 6] is one of the most influential and impressive results of complexity theory. Proven in the early 1990's, the theorem asserts that membership in any NP-set can be verified, with constant error probability (say 1%), by a verifier that probes a polynomially long (redundant) proof at a constant number of randomly selected locations. The PCP Theorem led to a breakthrough in the study of the complexity of combinatorial approximation problems (see, e.g., [14, 5]). Its original proof is very complex and involves the composition of two highly non-trivial proof systems, each minimizing a different parameter of the asserted PCP system (i.e., proof length and number of probed locations). An alternative approach to the proof of the PCP Theorem was recently presented by Dinur [12], and is reviewed below. In addition to yielding a simpler proof of the PCP Theorem, Dinur's approach resolves an important open problem regarding PCP systems (i.e., constructing a PCP system having proofs of almost-linear rather than polynomial length).

The original proof of the PCP Theorem focuses on the construction of two PCP systems that are highly non-trivial and interesting by themselves, and combines them in a natural manner. Loosely speaking, this combination (via proof composition) preserves the good features of each of the two systems; that is, it yields a PCP system that inherits the (logarithmic) randomness complexity of one system and the (constant) query complexity of the other. In contrast, Dinur's approach is focused at the "amplification" of PCP systems, via a gradual process of logarithmically many steps. It start from a trivial "PCP" system that rejects false assertions with probability inversely proportional to their length, and double the rejection probability in each step. In each step, the constant query complexity is preserved and the length of the PCP oracle is increased only by a constant factor. Thus, the process gradually transforms a very weak PCP system into a remarkable PCP system as postulated in the PCP Theorem.

In order to describe the aforementioned process we need to redefine PCP systems so to allow arbitrary soundness error. In fact, for technical reasons it is more convenient to describe the process as an iterated reduction of a "constraint satisfaction" problem to itself. Specifically, we refer to systems of 2-variable constraints, which are readily represented by (labeled) graphs.

Definition 4.1 (CSP with 2-variable constraints): *For a fixed finite set Σ, an instance of* `CSP` *consists of a graph $G = (V, E)$* (which may have parallel edges and self-loops) *and a sequence of 2-variable constraints $\Phi = (\phi_e)_{e\in E}$ associated with the edges, where each constraint has the form $\phi_e : \Sigma^2 \to \{0,1\}$. The value of an assignment $\alpha : V \to \Sigma$ is the number of constraints satisfied by α; that is, the value of α is $|\{(u,v) \in E :$*

$\phi_{(u,v)}(\alpha(u),\alpha(v)) = 1\}|$. *We denote by* $\mathtt{vlt}(G,\Phi)$ *the fraction of unsatisfied constraints under the best possible assignment; that is,*

$$\mathtt{vlt}(G,\Phi) = \min_{\alpha:V\to\{0,1\}} \{|\{(u,v)\in E : \phi_{(u,v)}(\alpha(u),\alpha(v)) = 0\}|/|E|\} \tag{4}$$

For various functions $t : \mathsf{N} \to [0,1]$*, we will consider the promise problem* $\mathtt{gapCSP}_t^{\Sigma}$*, having instances as above, such that the* YES*-instances are fully satisfiable instances* (i.e., $\mathtt{vlt} = 0$) *and the* NO*-instances are pairs* (G,Φ) *satisfying* $\mathtt{vlt}(G,\Phi) > t(|G|)$*, where* $|G|$ *denotes the number of edges in* G*.*

Note that 3SAT (and analogously other sets in $\mathcal{NP}$) is reducible to $\mathtt{gapCSP}_t^{\{1,\ldots,7\}}$ for $t(m) = 1/m$. Our goal is to reduce 3SAT (or rather $\mathtt{gapCSP}_t^{\{1,\ldots,7\}}$) to $\mathtt{gapCSP}_c^{\Sigma}$, for some fixed finite Σ and constant $c > 0$. The PCP Theorem will follow by showing a simple PCP system for $\mathtt{gapCSP}_c^{\Sigma}$. The desired reduction is obtained by iteratively applying the following reduction logarithmically many times.

Lemma 4.2 (amplifying reduction of gapCSP to itself): *For some finite* Σ *and constant* $c > 0$*, there exists a polynomial-time reduction of* $\mathtt{gapCSP}^{\Sigma}$ *to itself such that the following conditions hold with respect to the mapping of any instance* (G,Φ) *to the instance* (G',Φ')*.*

1. *If* $\mathtt{vlt}(G,\Phi) = 0$ *then* $\mathtt{vlt}(G',\Phi') = 0$.
2. $\mathtt{vlt}(G',\Phi') \geq \min(2\cdot\mathtt{vlt}(G,\Phi), c)$.
3. $|G'| = O(|G|)$.

Proof Sketch: The reduction consists of three steps. We first apply a pre-processing step that makes the underlying graph suitable for further analysis. The value of vlt may decrease during this step by a constant factor. The heart of the reduction is the second step in which we may increase vlt by any desired constant factor. The latter step also increases the alphabet Σ, and thus a post-processing step is employed to regain the original alphabet (by using any inner PCP systems; e.g., the Hadamard-based one presented in [5]). Details follow.

We first note that the aforementioned Σ and c, as well as the auxiliary parameters d and t, are fixed constants that will be determined to satisfy various conditions that arise in the course of our argument.

We start with the pre-processing step. Our aim in this step is to reduce the input (G,Φ) of $\mathtt{gapCSP}^{\Sigma}$ to an instance (G_1,Φ_1) such that G_1 is a d-regular expander graph.

Furthermore, each vertex in G_1 will have at least $d/2$ self-loops, $|G_1| = O(|G|)$, and $\texttt{vlt}(G_1, \Phi_1) = \Theta(\texttt{vlt}(G, \Phi))$. This step is quite simple.

The main step is aimed at increasing the fraction of violated constraints by a sufficiently large constant factor. This is done by reducing the instance (G_1, Φ_2) of $\texttt{gapCSP}^{\Sigma}$ to an instance (G_2, Φ_2) of $\texttt{gapCSP}^{\Sigma'}$ such that $\Sigma' = \Sigma^{d^t}$. Specifically, the vertex set of G_2 is identical to the vertex set of G_1, and each t-edge long path in G_1 is replaced by a corresponding edge in G_2, which is thus a d^t-regular graph. The constraints in Φ_2 are the natural ones, viewing each element of Σ' as a Σ-labeling of the ("distance $\leq t$") neighborhood of a vertex, and checking that two such labelings are consistent and satisfy Φ_1. That is, suppose that there is a path of length at most t in G_1 going from vertex u to vertex v and passing through vertex w. Then, there is an edge in G_1 between vertices u and v, and the constraint associated with it with mandates that the entries corresponding to vertex w in the Σ'-labeling of vertices u and v are identical. In addition, if the G_1-edge (w, w') is on a path of length at most t starting at v then the corresponding edge in G_2 is associated a constraint that enforces the constraint that is associated to (w, w') in Φ_1.

Clearly, if $\texttt{vlt}(G_1, \Phi_1) = 0$ then $\texttt{vlt}(G_2, \Phi_2) = 0$. The interesting fact is that the fraction of violated constraints increases by a factor of $\Omega(\sqrt{t})$; that is, $\texttt{vlt}(G_2, \Phi_2) \geq \min(\Omega(\sqrt{t} \cdot \texttt{vlt}(G_1, \Phi_1)), c)$. The intuition is that any Σ'-labeling to the vertices of G_2 must either be consistent with a Σ-labeling of G_1 or violate many edges in G_2. Focusing on the first case and relying on the hypothesis that G_1 is an expander, it follows that the set of violated edge-constraints (of Φ_1) with respect to the aforementioned Σ-labeling causes many more edge-constraints of Φ_2 to be violated. The point is that a set F of edges of G_1 is likely to appear on a $\min(\Omega(t) \cdot |F|/|G_1|, \Omega(1))$ fraction of the edges of G_2 (i.e., t-paths of G_1). (Note that the claim is obvious if G_1 were a complete graph, but it also holds for an expander.)[10]

The factor of $\Omega(\sqrt{t})$ gained in the second step, makes up for the constant factor lost in the first step (as well as the constant factor to be lost in the last step), while allowing us a constant factor amplification, for a suitable choice of the constant t. However, we obtained an instance of $\texttt{gapCSP}^{\Sigma'}$ rather than an instance of $\texttt{gapCSP}^{\Sigma}$, where $\Sigma' = \Sigma^{d^t}$. The purpose of the last step is to reduce the latter instance to an instance of $\texttt{gapCSP}^{\Sigma}$. This is done by viewing the instance of $\texttt{gapCSP}^{\Sigma'}$ as a (weak) PCP system and composing it with an inner-verifier, using the proof composition paradigm (of [9, 13], which in turn follow [6]). We stress that the inner-verifier used here needs only handle instances of constant size (i.e., having description length $O(d^t \log |\Sigma|)$), and so the one presented in [5] (or [8]) will do. The resulting PCP-system uses randomness $r \stackrel{\text{def}}{=} \log_2 |G_2| + (d^t \log |\Sigma|)^2$ and a constant number

[10] We also note that due to a technical difficulty it is easier to establish the claimed bound of $\Omega(\sqrt{t} \cdot \texttt{vlt}(G_1, \Phi_1))$ rather than $\Omega(t \cdot \texttt{vlt}(G_1, \Phi_1))$.

of binary queries, and has rejection probability $\Omega(\texttt{vlt}(G_2, \Phi_2))$, which is independent of the choice of the constant t. For $\Sigma = \{0,1\}^{O(1)}$, we obtain an instance of $\texttt{gapCSP}^{\Sigma}$ that has a $\Omega(\texttt{vlt}(G_2, \Phi_2))$ fraction of violated constraints. Furthermore, the size of the resulting instance is $O(2^r) = O(|G_2|)$, because d and t are constants. This completes the last step as well as the proof of the entire lemma. $\square$

Application to short PCPs. Recall that the PCP Theorem asserts that membership in any NP-set can be verified, with constant error probability, by a verifier that probes a polynomially long (redundant) proof at a constant number of randomly selected locations. Denoting by N the length of the standard proof, the length of the redundant proof was reduced in [9] to $\exp((\log N)^{\epsilon}) \cdot N$, for any $\epsilon > 0$. An open problem, explicitly posed in [9], is whether the length of the redundant proof can be reduced to $\text{poly}(\log N) \cdot N$. Building on prior work of [10], this seemingly difficult open problem was resolved by Dinur [12]: Specifically, viewing the system of [10] (which makes $\text{poly}(\log N)$ queries into a proof of length $\text{poly}(\log N) \cdot N$) as a PCP system with rejection probability $1/\text{poly}(\log N)$, Dinur amplifies the rejection probability for a double-logarithmic number of times, thus deriving the desired PCP system.

Acknowledgments

First of all, I wish to thank Irit Dinur, Mark Jerrum, Omer Reingold, Alistair Sinclair, Salil Vadhan, Eric Vigoda, and Avi Wigderson (i.e., the authors of [12, 17, 24, 25]) for their fascinating works. Next, I wish to thank Amnon Ta-Shma and Avi Wigderson for playing the Devil's advocate, thus forcing me to better articulate my feelings. Finally, I am grateful to Omer Reingold, Salil Vadhan, and Avi Wigderson for critical comments regarding an early version of this essay.

References

[1] 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 IEEE Symposium on Foundations of Computer Science*, pages 218–223, 1979.

[2] N. Alon. Eigenvalues and expanders. *Combinatorica*, Vol. 6, pages 83–96, 1986.

[3] N. Alon and V.D. Milman. λ_1, Isoperimetric Inequalities for Graphs and Superconcentrators, *J. Combinatorial Theory, Ser. B*, Vol. 38, pages 73–88, 1985.

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

[5] S. Arora, C. Lund, R. Motwani, M. Sudan and M. Szegedy. Proof Verification and Intractability of Approximation Problems. *Journal of the ACM*, Vol. 45, pages 501–555, 1998. Preliminary version in *33rd FOCS*, 1992.

[6] S. Arora and S. Safra. Probabilistic Checkable Proofs: A New Characterization of NP. *Journal of the ACM*, Vol. 45, pages 70–122, 1998. Preliminary version in *33rd FOCS*, 1992.

[7] L. Babai, L. Fortnow, L. Levin, and M. Szegedy. Checking Computations in Polylogarithmic Time. In *23rd ACM Symposium on the Theory of Computing*, pages 21–31, 1991.

[8] M. Bellare, O. Goldreich and M. Sudan. Free Bits, PCPs and Non-Approximability – Towards Tight Results. *SIAM Journal on Computing*, Vol. 27, No. 3, pages 804–915, 1998. Extended abstract in *36th IEEE Symposium on Foundations of Computer Science*, 1995.

[9] E. Ben-Sasson, O. Goldreich, P. Harsha, M. Sudan, and S. Vadhan. Robust PCPs of proximity, Shorter PCPs and Applications to Coding. In *36th ACM Symposium on the Theory of Computing*, pages 1–10, 2004. Full version in *ECCC*, TR04-021, 2004.

[10] E. Ben-Sasson and M. Sudan. Simple PCPs with Poly-log Rate and Query Complexity. *ECCC*, TR04-060, 2004.

[11] A. Broder. How hard is it to marry at random? (On the approximation of the Permanent). In *18th ACM Symposium on the Theory of Computing*, pages 50–58, 1986.

[12] I. Dinur. The PCP Theorem by Gap Amplification. *ECCC*, TR05-046, 2005.

[13] I. Dinur and O. Reingold. Assignment-testers: Towards a combinatorial proof of the PCP-Theorem. In *45th IEEE Symposium on Foundations of Computer Science*, pages 155–164, 2004.

[14] U. Feige, S. Goldwasser, L. Lovász, S. Safra, and M. Szegedy. Approximating Clique is almost NP-complete. *Journal of the ACM*, Vol. 43, pages 268–292, 1996. Preliminary version in *32nd FOCS*, 1991.

[15] O. Gaber and Z. Galil. Explicit Constructions of Linear Size Superconcentrators. *Journal of Computer and System Science*, Vol. 22, pages 407–420, 1981.

[16] M. Jerrum and A. Sinclair. Approximating the Permanent. *SIAM Journal on Computing*, Vol. 18, pages 1149–1178, 1989.

[17] M. Jerrum, A. Sinclair, and E. Vigoda. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Non-Negative Entries. *Journal of the ACM*, Vol. 51 (4), pages 671–697, 2004. Preliminary version in *33rd STOC*, pages 712–721, 2001.

[18] N. Linial, A. Samorodnitsky, and A. Wigderson. A Deterministic Strongly Polynomial Algorithm for Matrix Scaling and Approximate Permanents. *Combinatorica*, Vol. 20 (4), pages 545–568, 2000.

[19] A. Lubotzky, R. Phillips, and P. Sarnak. Ramanujan Graphs. *Combinatorica*, Vol. 8, pages 261–277, 1988.

[20] G.A. Margulis. Explicit Construction of Concentrators. (In Russian.) *Prob. Per. Infor.* 9 (4) (1973), 71–80. English translation in *Problems of Infor. Trans.*, pages 325–332, 1975.

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

[22] N. Nisan. $\mathcal{RL} \subseteq \mathcal{SC}$. *Journal of Computational Complexity*, Vol. 4, pages 1-11, 1994.

[23] N. Nisan and D. Zuckerman. Randomness is Linear in Space. *Journal of Computer and System Science*, Vol. 52 (1), pages 43–52, 1996.

[24] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, 2005.

[25] O. Reingold, S. Vadhan, and A. Wigderson. Entropy Waves, the Zig-Zag Graph Product, and New Constant-Degree Expanders and Extractors. *Annals of Mathematics*, Vol. 155 (1), pages 157–187, 2001. Preliminary version in *41st FOCS*, pages 3–13, 2000.

[26] E. Rozenman and S. Vadhan. Derandomized Squaring of Graphs. In *Proceedings of the 8th International Workshop on Randomization and Computation* (RANDOM'05), LNCS Vol. 3624, Springer, pages 436–447, 2005.

[27] L.G. Valiant. The Complexity of Computing the Permanent. *Theoretical Computer Science*, Vol. 8, pages 189–201, 1979.

[28] L. Wittgenstein. Tractatus Logico-Philosophicus. 1922.