Texts in Computational Complexity: Proving that Undirected Connectivity is in $\mathcal{L}$ (with a long appendix on expander graphs)

Oded Goldreich
Department of Computer Science and Applied Mathematics
Weizmann Institute of Science, Rehovot, Israel.

December 15, 2005

Preface. This text consists of two parts. The main part (i.e., Section 1) provides a presentation of Reingold's log-space algorithm for testing connectivity of (undirected) graphs. This algorithm relies heavily on the notion of expander graphs and on a specific construct that was developed in their stuty. The second part (i.e., Section 2) provides an overview of expander graphs, which extends beyond the very minimum needed in Section 1. Specifically, Section 1 only relies on the algebraic definition of expanders (presented in §2.1.1) and on the *zig-zag product* (defined in §2.2.2).

Preliminaries

We will use and/or refer to the following two composition lemmas.

Lemma 1 (naive composition): *Let $f_1 : \{0,1\}^* \to \{0,1\}^*$ and $f_2 : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be computable in space s_1 and s_2, respectively.*[1] *Then f defined by $f(x) \stackrel{\rm def}{=} f_2(x, f_1(x))$ is computable in space s such that*

$$s(n) = \max(s_1(n), s_2(n + \ell(n))) + \ell(n) + O(1)\,,$$

where $\ell(n) = \max_{x\in\{0,1\}^n}\{|f_1(x)|\}$.

Lemma 1 is useful when ℓ is relatively small, but in many cases $\ell \gg \max(s_1, s_2)$. In these cases, the following composition lemma is more useful.

Lemma 2 (emulative composition): *Let f_1, f_2, s_1, s_2, ℓ and f be as in Lemma 1. Then f is computable in space s such that*

$$s(n) = s_1(n) + s_2(n + \ell(n)) + O(\log(n + \ell(n)))\,.$$

1 Main text: Reingold's log-space algorithm

Exploring a graph (e.g., towards determining its connectivity) is one of the most basic and ubiquitous computational tasks regarding graphs. The standard graph exploration algorithms (e.g.,

[1] Here (and throughout the chapter) we assume, for simplicity, that all complexity bounds are monotonically non-decreasing.

BFS and DFS) require temporary storage that is linear in the number of vertices. In contrast, the algorithm presented in this section uses temporary storage that is only logarithmic in the number of vertices. In addition to demonstrating the power of log-space computation, this algorithm (or rather its actual implementation) provides a taste of the type of issues arising in the design of sophisticated log-space algorithms.

The intuitive task of "exploring a graph" is captured by the task of deciding whether a given graph is connected. In addition to the intrinsic interest in this natural computational problem, we note that related versions of the problem seem harder. For example, determining directed connectivity (in directed graphs) captures the essence of the class $\mathcal{NL}$. In view of this situation, we emphasize the fact that the computational problem considered here refers to undirected graphs by calling it undirected connectivity.

Theorem 3 *Deciding undirected connectivity* (UCONN) *is in* $\mathcal{L}$

The algorithm is based on the fact that UCONN is easy in the special case that the graph consists of a collection of constant degree expanders (see Section 2). In particular, if the graph has constant degree and logarithmic diameter then it can be explored using a logarithmic amount of space (which is used for determining a generic path from a fixed starting vertex).[2]

However, the input graph does not necessarily consist of a collection of constant degree expanders. The main idea is then to transform the input graph into one that does satisfy the aforementioned condition, while preserving the number of connected components of the graph. Needless to say, the key point is performing such a transformation in logarithmic space. The rest of this section is devoted to the description of such a transformation. We first present the basic approach and next turn to the highly non-trivial implementation details.

We first note that it is easy to transform the input graph $G_0 = (V_0, E_0)$ into a constant-degree graph G_1 that preserves the number of connected components in G_0. Specifically, each vertex $v \in V$ having degree $d(v)$ (in G_0) is represented by a cycle C_v of $d(v)$ vertices (in G_1), and each edge $\{u, v\} \in E_0$ is replaced by an edge having one end-point on the cycle C_v and the other end-point on the cycle C_u such that each vertex in G_1 has degree three (i.e., has two cycle edges and a single intra-cycle edge). This transformation can be performed using logarithmic space, and thus (relying on Lemma 2) we assume throughout the rest of the proof that the input graph has degree three. Our goal is to transform this graph into a collection of expanders, while maintaining the number of connected components. In fact, *we shall describe the transformation while pretending that the graph is connected, while noting that otherwise the transformation acts separately on each connected component.*

A couple of technicalities. For a constant integer $d > 2$ determined so as to satisfy some additional condition, we may assume that the input graph is actually d^2-regular (albeit is not necessarily simple). Furthermore, we shall assume that this graph is not bipartite. Both assumptions can be justified by augmenting the aforementioned construction of a 3-regular graph by adding $d^2 - 3$ self-loops to each vertex.

Prerequisites: Needless to say, the aforementioned transformation refers to the notion of an expander graph (as defined in §2.1.1). The transformation also relies on the *zig-zag product* defined in §2.2.2.

[2]For further details, see Exercise 14.

1.1 The basic approach

Recall that our goal is to transform G_1 into an expander. The transformation is gradual and consists of logarithmically many iterations, where in each iteration an adequate expansion parameter doubles while the graph becomes a constant factor larger and maintains the degree bound. The (expansion) parameter of interest is the gap between the relative second eigenvalue of the graph and 1 (see §2.1.1). A constant value of this parameter indicates that the graph is an expander. Initially, this parameter is lower-bounded by $1/O(n^2)$, where n is the size of the graph, and after logarithmically many iterations this parameter is lower-bounded by a constant (and the current graph is an expander).

The crux of the aforementioned gradual transformation is the transformation that takes place in each single iteration. This transformation combines the standard graph powering (to a constant power c) and the *zig-zag product* presented in §2.2.2. Specifically, for adequate positive integers d and c, we start with the d^2-regular graph $G_1 = (V_1, E_1)$, and 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. That is, in each iteration, we raise the current graph (i.e., G_i) to the power c and combine the resulting graph with the fixed graph G using the zig-zag product. Thus, G_i is a d^2-regular graph with $d^{(i-1)\cdot 2c} \cdot |V_1|$ vertices, where this invariant is preserved by definition of the zig-zag product.

The analysis of the improvement in the expansion parameter, denoted $\delta_2(\cdot) \stackrel{\text{def}}{=} 1 - \lambda_2(\cdot)$, relies on Eq. (7). Recall that Eq. (7) implies that if $\lambda_2(G) < 1/2$ then $1 - \lambda_2(G' ⓩ G) > (1 - \lambda_2(G'))/3$. Thus, the fixed graph G is selected such that $\lambda_2(G) < 1/2$, which requires a sufficiently large constant d. Thus, we have

$$\delta_2(G_{i+1}) \;=\; 1 - \lambda_2(G_i^c ⓩ G) \;>\; \frac{1 - \lambda_2(G_i^c)}{3} \;=\; \frac{1 - \lambda_2(G_i)^c}{3}$$

whereas, for sufficiently large constant c, it holds that $1 - \lambda_2(G_i)^c > \max(6 \cdot (1 - \lambda_2(G_i)), 1/2)$. It follows that that $\delta_2(G_{i+1}) > \max(2\delta_2(G_i), 1/6)$. Thus, setting $t = O(\log |V_1|)$ and using $\delta_2(G_1) = 1 - \lambda_2(G_1) = \Omega(|V_1|^{-2})$, we obtain $\delta_2(G_t) > 1/6$ as desired.

One detail of crucial importance is the ability to transform G_1 into G_t via a log-space computation. Indeed, the transformation of G_i to G_{i+1} can be performed in logarithmic space (see Exercise 15), but we need to compose a logarithmic number of such transformations. Unfortunately, the standard composition lemmas for space-bounded algorithms involve overhead that we cannot afford.[3] Still, taking a closer look at the transformation of G_i to G_{i+1}, one may note that it is highly structured and in some sense it can be implemented in constant space and supports a stronger composition result that incurs only a constant amount of storage per iteration. The resulting implementation (of the iterative transformation of G_1 to G_t) and the underlying formalism will be the subject of Section 1.2. (An alternative implementation, provided in [11], can be obtained by unraveling the composition.)

1.2 The actual implementation

The space-efficient implementation of the iterative transformation outlined in Section 1.1 is based on the observation that we do not need to explicitly construct the various graphs but merely provide "oracle access" to them. This observation is crucial when applied to the intermediate graphs; rather

[3]Needless to say, we cannot afford the naive composition (of Lemma 1), since it causes an overhead linear in the size of the intermediate output. As for the emulative composition (of Lemma 2), it sums up the space complexities of the composed algorithms (not to mention adding another logarithmic term), which would result in a log-squared bound on the space complexity.

than constructing G_{i+1}, when given G_i as input, we show how to provide oracle access to G_{i+1} (i.e., answer "neighborhood queries" regarding G_{i+1}) when given oracle access to G_i (i.e., an oracle that answers neighborhood queries regarding G_i). That is, we view G_i and G_{i+1} (or rather their incidence lists) as functions (to be evaluated) rather than as strings (to be printed), and show how to reduce the task of finding neighbors in G_{i+1} (i.e., evaluating the "incidence function" at a given vertex) to the task of finding neighbors in G_i.

A clarifying discussion. Note that here we are referring to oracle machines that access a finite oracle, which represents a *finite variable object* (which in turn is an instance of some computational problem), rather than following the convention by which the oracle represents a *fixed computational problem*. Still the mechanism (and/or operations) of these two types of oracle machines is the same: They both get an input (which here is a "query" regarding a variable object rather than an instance of a fixed computational problem), and produce an output (which here is the answer to the query rather than a "solution" for the given instance). Analogously, these machines make queries (which here are queries regarding another variable object rather than queries regarding another fixed computational problem), and obtain corresponding answers.

As usual, queries are made via a special write-only device and the answers are read from a corresponding read-only device, where the use of these devices is not charged in the space complexity. With these conventions in place, we claim that neighborhoods in the d^2-regular graph G_{i+1} can be computed by a constant-space oracle machine that is given oracle access to the d^2-regular graph G_i. That is, letting $g_i : V_i \times [d^2] \to V_i \times [d^2]$ (resp., $g_{i+1} : V_{i+1} \times [d^2] \to V_{i+1} \times [d^2]$) denote the edge rotation function[4] of G_i (resp., G_{i+1}), we have:

Claim 4 *There exists a constant-space oracle machine that evaluates g_{i+1} when given oracle access to g_i, where the state of the machine is counted in the space complexity.*

Proof Sketch: We first show that the two basic operation that underly the definition of G_{i+1} (i.e., powering and zig-zag product with a constant graph) can be performed in constant-space.

The edge rotation function of the square of a graph can be evaluated at $(v, \langle j_1, j_2\rangle)$, using a constant amount of space, by evaluating the edge rotation function of the original graph twice. First, by making the query (v, j_1) we obtain the edge rotation of (v, j_1), denoted (u, k_1), and next, making the query (u, j_2), we obtain (w, k_2) and output $(w, \langle k_2, k_1\rangle)$. We stress that we only use the temporary storage to record k_1, whereas u is directly copied from the oracle answer device to the oracle query device. Accounting also for a constant number of states needed for the various stages of the foregoing activity, we conclude that graph squaring can be performed in constant-space. The argument extends to the task of raising the graph to any constant power.

Turning to the zig-zag product (of G' with a fixed G), we note that the corresponding edge rotation function can be evaluated in constant-space (given oracle access to the edge rotation function of G'). This follows directly from Eq. (5), noting that the latter calls for a single evaluation of the edge rotation function of G' and two simple modifications that only depend on the constant-size graph G. Again, using the fact that it suffices to copy vertex names from the input (or oracle answer device) to the oracle query device (or output), we conclude that the aforementioned activity can be performed using constant space.

The argument extends to a sequential composition of a constant number of operations of the aforementioned type (i.e., graph squaring and zig-zag product with a constant graph). $\square$

[4] Recall that the edge rotation function of a graph maps the pair (v, j) to the pair (u, k) if vertex u is the j^{th} neighbor of vertex v and v is the k^{th} neighbor of u (see §2.2.2).

Recursive composition. Using Claim 4, we wish to obtain a log-space oracle machine that evaluates g_t by making oracle calls to g_1, where $t = O(\log|V_1|)$. Such an oracle machine will yield a log-space transformation of G_1 to G_t (by evaluating g_t at all possible values). It is tempting to hope that an adequate composition lemma, when applied to Claim 4, will yield the desired log-space oracle machine (reducing the evaluation of g_t to g_1). This is indeed the case, except that the adequate composition lemma is still to be developed (as we do next).

We first note that applying a naive composition (as in Lemma 1) amounts to an additive overhead of $O(\log|V_1|)$ *per each composition.* But we cannot afford more than an amortized constant additive overhead per composition. Applying the emulative composition (as in Lemma 2) amounts in multiplicative overhead, which is certainly unaffordable. The composition developed next is a variant of the naive composition, which is beneficial in the context of recursive calls. The basic idea is deviating from the paradigm that allocates separate input/output and query devices to each level in the recursion, and combining all these devices in a single ("global") device which will be used by all levels of the recursion. That is, rather than following the "structured programming" methodology of using (locally) designated space for passing information to the subroutine, we use the "bad programming" methodology of passing information through global variables. As usual, this notion is formulated by referring to the model of multi-tape Turing machine, but it can be formulated in any other reasonable model of computation.

Definition 5 (global-tape oracle machines): *A* global-tape oracle machine *is defined as an oracle machine, except that the input, output and oracle tapes are replaced by a single* global tape. *In addition, the machine has a constant number of work tapes, called the* local tapes. *The machine obtains its input from the global tape, writes each query on this very tape, obtains the corresponding answer from this tape, and writes its final output on this tape. The space complexity of such a machine is stated when referring separately to the use of the global tape and to the use of the local tapes.*

Clearly, any ordinary oracle machine can be converted into an equivalent global-tape oracle machine. The resulting machine uses a global tape of length at most $n + \ell + m$, where n denotes the length of the input, ℓ denote the length of the longest query or oracle answer, and m denotes the length of the output. However, combining the different tapes into one global tape seems to require holding separate pointers for each of the original tapes, which means that the local tape has to be used to store corresponding counters (in addition to the original work-tape). A key observation is that these counters can be avoided in the case that the original machine can be described as a sequence of transformations (of the input into the first query, and of the i^{th} answer to the $i+1^{\text{st}}$ query or the output), while maintaining auxiliary information on the work-tape. Indeed, the machine presented in the proof of Claim 4 has this form, and thus can be implemented by a global-tape oracle machine that uses a global-tape not longer than its input and a local-tape of constant length.

Claim 6 (Claim 4, revisited): *There exists a global-tape oracle machine that evaluates g_{i+1} when given oracle access to g_i, while using global tape of length $\log_2(d^2|V_{i+1}|)$ and a local tape of constant length.*

Proof Sketch: Following the proof of Claim 4, we merely indicate the exact use of the two tapes. For example, recall that the edge rotation function of the square of G_i is evaluated at $(v, \langle j_1, j_2\rangle)$ by evaluating the edge rotation function of the original graph first at (v, j_1) and then at (u, j_2), where $(u, k_1) = g_i(v, j_1)$. This means the global-tape machine first reads $(v, \langle j_1, j_2\rangle)$ from the global tape and replaces it by the query (v, j_1), while storing j_2 on the local tape. Thus, the machine merely deletes a constant number of bits from the global tape (and leaves its prefix intact). After invoking the oracle, the machine copies k_1 from the global tape (which currently holds (u, k_1)) to its local tape, and copies j_2 from its local tape to the global tape (such that it contains (u, j_2)). After invoking the oracle for the second time, the global tape contains $(w, k_2) = g_i(u, j_2)$, and the machine merely modifies it to $(w, \langle k_2, k_1\rangle)$, which is the desired output.

Similarly, the edge rotation function of the zig-zag product of G' with a fixed G is evaluated at $(\langle u, i\rangle, \langle \alpha, \beta\rangle)$ by querying G' at $(u, E_\alpha(i))$ and outputting $(\langle v, E_\beta(j')\rangle, \langle \beta, \alpha\rangle)$, where (v, j') denotes the oracle answer (see Eq. (5)). This means that the global-tape oracle machine first copies α, β from the global-tape to the local-tape, transforms the contents of the global tape from $(\langle u, i\rangle, \langle \alpha, \beta\rangle)$ to $(u, E_\alpha(i))$, and makes an analogous transformation after the oracle is invoked. □

Composing global-tape oracle machines. In the proof of Claim 6, we implicitly used sequential composition of computations conducted by global-tape oracle machines.[5] In general, when sequentially composing such computations the global-tape and local-tape usage are the maximum among all composed computations; that is, the current formalism offers a tight bound on naive composition (as opposed to Lemma 1). Furthermore, global-tape oracle machines are beneficial in the context of recursive composition, as indicated by Lemma 7 (which relies on this model in a crucial way). The key observation is that all levels in the recursive composition may re-use the same global storage, and only the local storage gets added. Consequently we have the following composition lemma, where n denotes the length of the input to f_1 (and t may depend on n).

Lemma 7 (recursive composition in the global-tape model): *Suppose that, for every $i = 1, ..., t-1$, there exists a global-tape oracle machine that computes f_i by making oracle calls to f_{i+1} while using a global-tape of length $L(n)$ and a local-tape of length $l_i(n)$, which also accounts for the machine's state. Then f_1 can be computed by a standard oracle machine that makes calls to f_t and uses space $L(n) + \sum_{i=1}^{t-1} O(l_i(n))$.*

We shall apply this lemma with $f_i = g_{t+1-i}$, using the bounds $L(n) = O(n)$ and $l_i(n) = O(1)$ (as guaranteed by Claim 6).

Proof Sketch: We compute f_1 by allocating space for the emulation of the global-tape and the local-tapes of each level in the recursion. We emulate the recursive computation by capitalizing on the fact that all recursive levels use the same global-tape (for making queries and receiving answers). In the actual recursion, each level may use the global-tape arbitrarily as long as when it returns control to the invoking machine the global-tape contains the right answer. Thus, the emulation may do the same, and emulate each recursive call by using the space allocated for the global-tape as well as the space designated for the local-tape of this level. □

Conclusion. Combining Claim 6 and Lemma 7, we conclude that the evaluation of $g_{O(\log |V_1|)}$ can be reduced to the evaluation of g_1 in space $O(\log |V_1|)$. Recalling that G_1 can be constructed in log-space (based on the input graph G_0), we infer that $G' = G_{O(\log |V_1|)}$ can be constructed in log-space. Theorem 3 follows by recalling that G' (which has constant degree and logarithmic diameter) can be tested for connectivity in log-space (see Exercise 14). Using a similar argument, we can test whether a given pair of vertices are connected in the input graph (see Exercise 16).

[5] A similar composition took place in the proof of Claim 4, but here we assert a stronger feature of this specific computation.

2 Appendix: Expander Graphs

This appendix is more elaborate than necessary for the main text. The latter merely relies on the algebraic definition of expanders (as in §2.1.1) and on the *zig-zag product* defined in §2.2.2. Still, we believe that the reader may not mind having the wider perspective provided below.

Loosely speaking, expander graphs are graphs of small degree that exhibit various properties of cliques. In particular, we refer to properties such as the relative sizes of cuts in the graph, and the rate at which a random walk converges to the uniform distribution (relative to the logarithm of the graph size to the base of its degree).

Some technicalities. Typical presentation of expander graphs refer to one of several variants. For example, in some sources, expanders are defined as bipartite graphs, whereas in others they are very far from being bipartite. We shall follow the latter convention. Furthermore, at times we implicitly consider an augmentation of these graphs where self-loops are added to each vertex. For simplicity, we also allow parallel edges.

We often talk of expander graphs while we actually mean an infinite collection of graphs such that each graph in this collection satisfies the same property (which is informally attributed to the collection). For example, when talking of a d-regular expander (graph) we actually refer to an infinite collection of graphs such that each of these graphs is d-regular. Typically, such a collection (or family) contains a single N-vertex graph for every $N \in S$, where S is an infinite subset of $\mathbb{N}$. Throughout this section, we denote such a collection by $\{G_N\}_{N \in S}$, with the understanding that G_N is a graph with N vertices and S is an infinite set of natural numbers.

2.1 Definitions and Properties

We consider two definitions of expander graphs, two different notions of explicit constructions, and two useful properties of expanders.

2.1.1 Two Mathematical Definitions

We start with two different definitions of expander graphs. These definitions are qualitatively equivalent and even quantitatively related. We start with the algebraic definition, and later present the combinatorial definition.

The algebraic definition (spectral gap). Identifying graphs with their adjacency matrix, we consider the eigenvalues (and eigenvectors) of a graph (or rather of its adjacency matrix). Any d-regular graph $G = (V, E)$ has the uniform vector as an eigenvector corresponding to the eigenvalue d, and if G is connected and not bipartite then all other eigenvalues are strictly smaller than d. The second eigenvalue, denoted $\lambda_2(G) < d$, of such a graph G is thus a tight upper-bound on the *absolute value* of all the other eigenvalues. Using the connection to the combinatorial definition, it follows that $\lambda_2(G) < d - \Omega(1/|V|^2)$ holds (for every connected non-bipartite d-regular graph G). The algebraic definition of expanders refers to an infinite family of d-regular graphs and requires the existence of a *constant* eigenvalue bound that holds for all the graphs in the family.

Definition 8 *An infinite family of d-regular graphs, $\{G_N\}_{N \in S}$, where $S \subseteq \mathbb{N}$,* satisfies the eigenvalue bound λ *if for every $N \in S$ it holds that $\lambda_2(G_N) \leq \lambda$.*

In such a case we say that the family has spectral gap $d-\lambda$. It will be often convenient to consider relative (or normalized) versions of these quantities, obtained by division by d. This technical definition is the one typically used in complexity theoretic applications, since it directly implies various "mixing properties" (see §2.1.3).

The combinatorial definition (expansion). This algebraic definition is related to the combinatorial definition of expansion. Loosely speaking, expansion requires that any (not too big) set of vertices of the graph has a relatively large set of neighbors. Specifically, a graph $G=(V,E)$ is c-expanding if, for every set $S \subset V$ of cardinality at most $|V|/2$, it holds that

$$\Gamma_G(S) \stackrel{\text{def}}{=} \{v : \exists u \in S \text{ s.t. } (u,v) \in E\} \tag{1}$$

has cardinality at least $(1+c)\cdot|S|$. Equivalently (assuming the existence of self-loops on all vertices), we may require that $|\Gamma_G(S)\setminus S| \geq c\cdot|S|$. Clearly, every connected graph $G=(V,E)$ is $(1/|V|)$-expanding. The combinatorial definition of expanders refers to an infinite family of d-regular graphs and requires the existence of a *constant* expansion bound that holds for all the graphs in the family.

Definition 9 *An infinite family of d-regular graphs, $\{G_N\}_{N\in S}$* is c-expanding *if for every $N \in S$ it holds that G_N is c-expanding.*

The two definitions of expander graphs are related (see [3, Sec. 9.2]).

Theorem 10 *Let G be a non-bipartite d-regular graph.*

1. *The graph G is c-expanding for $c \geq (d-\lambda_2(G))/2d$.*
2. *If G is c-expanding then $d-\lambda_2(G) \geq c^2/(4+2c^2)$.*

Thus, any non-zero bound on combinatorial expansion of a family of d-regular graphs yields a non-zero bound on its spectral gap, and vice versa. Note, however, that the back-and-forth translation between these definitions is not tight. Our applications refer to the algebraic definition, and the loss incurred in Theorem 10 is immaterial for them.

Amplification. The quality of expander graphs improves by raising them to any power $t>1$ (i.e., raising their adjacency matrix to the t^{th} power), which corresponds to considering graphs in which t-paths are replaced by edges. Using the algebraic definition, we have $\lambda_2(G^t)=\lambda_2(G)^t$, but indeed the degree also gets raised to the power t. Still, the ratio $\lambda_2(G^t)/d^t$ deceases with t. An analogous phenomenon occurs also under the combinatorial definition, provided that some suitable modifications are applied. For example, if $G=(V,E)$ is c-expanding (i.e., for every $S \subseteq V$ it holds that $|\Gamma_G(S)| \geq \min((1+c)\cdot|S|,|V|/2))$, then for every $S \subseteq V$ it holds that $|\Gamma_{G^t}(S)| \geq \min((1+c)^t\cdot|S|,|V|/2)$.

The optimal eigenvalue bound. For every d-regular graph $G=(V,E)$, it holds that $\lambda_2(G) \geq 2\gamma_G\cdot\sqrt{d-1}$, where $\gamma_G = 1-O(1/\log_d|V|)$. Thus, $2\sqrt{d-1}$ is a lower-bound on the eigenvalue bound of any infinite family of d-regular graphs.

2.1.2 Two levels of explicitness

A mild level of explicit constructions refer to the complexity of constructing the entire graph. An infinite family of graphs $\{G_N\}_{N\in S}$ is said to be explicitly constructible if there exists a polynomial-time algorithm that, on input 1^N (where $N \in S$), outputs the list of the edges in the N-vertex graph G_N.

The aforementioned level of explicitness suffices when the application requires holding the entire graph and/or runs in time that is lower-bounded by the size of the graph. In contrast, other applications only refer to a huge virtual graph (which is much bigger than their running time), and only require the computation of the neighborhood relations in such a graph. In this case, the following stronger level of explicitness is relevant.

A strongly explicit construction of an infinite family of (d-regular) graphs $\{G_N\}_{N\in S}$ is a polynomial-time algorithm that on input N (in binary), a vertex v in the N-vertex graph G_N and an index i ($i \in \{1, ..., d\}$), returns the i^{th} neighbor of v. That is, the neighbor is determined in time that is polylogarithmic in the size of the graph. Needless to say, the strong level of explicitness implies the basic level.

An additional requirement, which is often forgotten but is very important, refers to the "tractability" of the set S. Specifically, we require the existence of an efficient algorithm that given any $n \in \mathbb{N}$ finds an $s \in S$ such that $n \le s < 2n$. Corresponding to the foregoing definitions, efficient may mean either running in time $\text{poly}(n)$ or running in time $\text{poly}(\log n)$. The requirement that $n \le s < 2n$ suffices in most applications, but in some cases a smaller interval (e.g., $n \le s < n+\sqrt{n}$) is required, whereas in others a larger interval (e.g., $n \le s < \text{poly}(n)$) suffices.

Greater flexibility. In continuation to the foregoing paragraph, we comment that expanders can be combined in order to obtain expanders for a wider range of sizes. For example, two d-regular c-expanding graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where $|V_1| \le |V_2|$ and $c \le 1$, can be combined into a $(d+1)$-regular $c/2$-expanding graph on $|V_1|+|V_2|$ vertices by connecting the two graphs with a perfect matching of V_1 and $|V_1|$ of the vertices of V_2 (and adding self-loops to the remaining vertices of V_2). More generally, the d-regular c-expanding graphs, $G_1 = (V_1, E_1)$ through $G_t = (V_t, E_t)$, where $N \stackrel{\text{def}}{=} \sum_{i=1}^{t-1} |V_i| \le |V_2|$, yield a $(d+1)$-regular $c/2$-expanding graph on $\sum_{i=1}^{t} |V_i|$ vertices by using a perfect matching of $\cup_{i=1}^{t-1} V_i$ and N of the vertices of V_2.

2.1.3 Two properties

The following two properties provide a quantitative interpretation to the statement that expanders approximate the complete graph. The deviation from the latter is represented by an error term that is linear in λ/d.

The mixing lemma. The following lemma is folklore and has appeared in many papers. Loosely speaking, the lemma asserts that expander graphs (for which $d \gg \lambda$) have the property that the fraction of edges between two large sets of vertices approximately equals the product of the densities of these sets. This property is called *mixing*.

Lemma 11 (Expander Mixing Lemma): *For every d-regular graph $G = (V, E)$ and for every two subsets $A, B \subseteq V$ it holds that*

$$\left|\frac{|(A \times B) \cap E_2|}{|E_2|} - \frac{|A|}{|V|} \cdot \frac{|B|}{|V|}\right| \le \frac{\lambda_2(G)\sqrt{|A| \cdot |B|}}{d \cdot |V|} \le \frac{\lambda_2(G)}{d}$$

where E_2 denotes the set of directed edges that correspond to the undirected edges of G (i.e., $E_2 = \{(u,v) : \{u,v\} \in E\}$ and $|E_2| = d|V|$).

Proof: Let $N \stackrel{\rm def}{=} |V|$ and $\lambda \stackrel{\rm def}{=} \lambda_2(G)$. For any subset of the vertices $S \subseteq V$, we denote its density in V by $\rho(S) \stackrel{\rm def}{=} |S|/N$. Hence, the claim of the lemma is restated as

$$\left|\frac{|(A\times B)\cap E_2|}{d\cdot N} - \rho(A)\cdot\rho(B)\right| \leq \frac{\lambda\sqrt{\rho(A)\cdot\rho(B)}}{d}.$$

We proceed by providing bounds on the value of $|(A\times B)\cap E_2|$. To this end we let $\overline{a}$ denote the N-dimensional Boolean vector having 1 in the $i^{\rm th}$ component if and only if $i \in A$. The vector $\overline{b}$ is defined similarly. Denoting the adjacency matrix of the graph G by $M = (m_{i,j})$, we note that $|(A\times B)\cap E_2|$ equals $\overline{a}^\top M \overline{b}$ (because $(i,j) \in (A\times B)\cap E_2$ if and only if it holds that $i \in A$, $j \in B$ and $m_{i,j} = 1$). We consider the *orthogonal eigenvector basis*, $\overline{e_1}, ..., \overline{e_N}$, where $\overline{e_1} = (1,...,1)^\top$ and $\overline{e_i}^\top \overline{e_i} = N$ for each i, and write each vector as a linear combination of the vectors in the basis. Specifically, we denote by a_i the coefficient of $\overline{a}$ in the direction of $\overline{e_i}$; that is, $a_i = (\overline{a}^\top \overline{e_i})/N$ and $\overline{a} = \sum_i a_i \overline{e_i}$. Note that $a_1 = (\overline{a}^\top (1,...,1)^\top)/N = \rho(A)$ and $\sum_{i=1}^N a_i^2 = (\overline{a}^\top \overline{a})/N = \rho(A)$. Similarly for $\overline{b}$. It now follows that

$$\begin{aligned} |(A\times B)\cap E_2| &= \overline{a}^\top M \left(b_1\overline{e_1} + \sum_{i=2}^N b_i \overline{e_i}\right) \\ &= \rho(B)\cdot d\cdot |A| + \sum_{i=2}^N b_i \lambda_i \overline{a}^\top \overline{e_i} \end{aligned}$$

where λ_i denotes the $i^{\rm th}$ eigenvalue of M (and indeed $\lambda_1 = d$). Thus,

$$\begin{aligned} \frac{|(A\times B)\cap E|}{dN} &= \rho(B)\rho(A) + \sum_{i=2}^N \frac{\lambda_i a_i b_i}{d} \\ &\in \left[\rho(B)\rho(A) \pm \frac{\lambda}{d}\cdot\sum_{i=2}^N a_i b_i\right] \end{aligned}$$

Using $\sum_{i=1}^N a_i^2 = \rho(A)$ and $\sum_{i=1}^N b_i^2 = \rho(B)$, and applying Cauchy-Schwartz Inequality, we bound $\sum_{i=2}^N a_i b_i$ by $\sqrt{\rho(A)\rho(B)}$. The lemma follows. ■

The random walk lemma. Loosely speaking, the first part of the following lemma asserts that, as far as remaining trapped in some subset of the vertex set is concerned, a random walk on an expander approximates a random walk on the complete graph.

Lemma 12 (Expander Random Walk Lemma): *Let $G = ([N], E)$ be a d-regular graph, and consider walks on G that start from a uniformly chosen vertex and take $\ell - 1$ additional random steps, where in each such step we uniformly selects one out of the d edges incident at the current vertex and traverses it.*

1. *Let W be a subset of $[N]$ and $\rho \stackrel{\rm def}{=} |W|/N$. Then the probability that such a random walk stays in W is at most*

$$\rho\cdot\left(\rho + (1-\rho)\cdot\frac{\lambda_2(G)}{d}\right)^{\ell-1}. \qquad (2)$$

2. *For any $W_0, ..., W_{\ell-1} \subseteq [N]$, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is at most*
$$\sqrt{\rho_0} \cdot \prod_{i=1}^{\ell-1} \sqrt{\rho_i + (\lambda/d)^2}, \tag{3}$$
where $\rho_i \stackrel{\rm def}{=} |W_i|/N$.

The basic principle underlying Lemma 12 was discovered by Ajtai, Komlos, and Szemerédi [2], who proved a bound as in Eq. (3). The better analysis yielding Part 1 is due to Kahale [6, Cor. 6.1]. More general bounds that refer to the probability of staying in W for a number of times that approximates $|W|/N$ are given in [5], which actually considers an even more general problem (i.e., obtaining Chernoff-type bounds for random variables that are generated by a walk on a Markov Chain).

Proof of Equation (3): The basic idea is to view the random walk as the evolution of a corresponding probability vector under suitable transformations. The transformations correspond to taking a random step in G and to passing through a "sieve" that keeps only the entries that correspond to the current set W_i. The key observation is that the first transformation shrinks the component that is orthogonal to the uniform distribution, whereas the second transformation shrinks the component that is in the direction of the uniform distribution. Details follow.

Let A be a matrix representing the random walk on G (i.e., A is the adjacency matrix of G divided by d). Let $\hat{\lambda}$ denote the absolute value of the second largest eigenvalue of A (i.e., $\hat{\lambda} \stackrel{\rm def}{=} \lambda_2(G)/d$), and note that $\overline{u} = (N^{-1}, ..., N^{-1})^\top$ (which represents the uniform distribution) is the eigenvector of A that is associated with the largest eigenvalue (which is 1). Let P_i be a 0-1 matrix that has 1-entries only on its diagonal, and furthermore entry (j, j) is set to 1 if and only if $j \in W_i$. Then, the probability that a random walk of length ℓ intersects $W_0 \times W_1 \times \cdots \times W_{\ell-1}$ is the sum of the entries of the vector
$$\overline{v} \stackrel{\rm def}{=} P_{\ell-1}A \cdots P_2AP_1AP_0\overline{u}. \tag{4}$$
We are interested in upper-bounding $\|\overline{v}\|_1$, and use $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$, where $\|\overline{z}\|_1$ and $\|\overline{z}\|$ denote the L_1-norm and L_2-norm of $\overline{z}$, respectively (e.g., $\|\overline{u}\|_1 = 1$ and $\|\overline{u}\| = N^{-1/2}$). The key observation is that the linear transformation P_iA shrinks every vector.

Main Claim. For every $\overline{z}$, it holds that $\|P_iA\overline{z}\| \leq (\rho_i + \hat{\lambda}^2)^{1/2} \cdot \|\overline{z}\|$.

Proof. Intuitively, A shrinks the component of $\overline{z}$ that is orthogonal to $\overline{u}$, whereas P_i shrinks the component of $\overline{z}$ that is in the direction of $\overline{u}$. Specifically, we decompose $\overline{z} = \overline{z_1} + \overline{z_2}$ such that $\overline{z_1}$ is the projection of $\overline{z}$ on $\overline{u}$ and $\overline{z_2}$ is the component orthogonal to $\overline{u}$. Then, using the triangle inequality and other obvious facts (i.e., $\|P_iA\overline{z_1}\| = \|P_i\overline{z_1}\|$ and $\|P_iA\overline{z_2}\| \leq \|A\overline{z_2}\|$), we have
$$\begin{aligned}
\|P_iA\overline{z_1} + P_iA\overline{z_2}\| &\leq \|P_iA\overline{z_1}\| + \|P_iA\overline{z_2}\| \\
&\leq \|P_i\overline{z_1}\| + \|A\overline{z_2}\| \\
&\leq \sqrt{\rho_i} \cdot \|\overline{z_1}\| + \hat{\lambda} \cdot \|\overline{z_2}\|
\end{aligned}$$
where the last inequality uses the fact that P_i shrinks any uniform vector by eliminating $1 - \rho_i$ of its elements, whereas A shrinks the length of any eigenvector except $\overline{u}$ by a factor of at least $\hat{\lambda}$. Using the Cauchy-Schwartz inequality[6], we get
$$\|P_iA\overline{z}\| \leq \sqrt{\rho_i + \hat{\lambda}^2} \cdot \sqrt{\|\overline{z_1}\|^2 + \|\overline{z_2}\|^2}$$

[6]That is, we get $\sqrt{\rho_i}\|z_1\| + \hat{\lambda}\|z_2\| \leq \sqrt{\rho_i + \hat{\lambda}^2} \cdot \sqrt{\|z_1\|^2 + \|z_2\|^2}$, by using $\sum_{i=1}^n a_i \cdot b_i \leq \left(\sum_{i=1}^n a_i{}^2\right)^{1/2} \cdot \left(\sum_{i=1}^n b_i{}^2\right)^{1/2}$.

$$= \sqrt{\rho_i + \hat{\lambda}^2} \cdot \|\overline{z}\|$$

where the equality is due to the fact that $\overline{z_1}$ is orthogonal to $\overline{z_2}$. $\square$

Recalling Eq. (4) and using the Main Claim (and $\|\overline{v}\|_1 \leq \sqrt{N} \cdot \|\overline{v}\|$), we get

$$\begin{aligned}
\|\overline{v}\|_1 &\leq \sqrt{N} \cdot \|P_{\ell-1}A \cdots P_2 A P_1 A P_0 \overline{u}\| \\
&\leq \sqrt{N} \cdot \left(\prod_{i=1}^{\ell-1} \sqrt{\rho_i + \hat{\lambda}^2} \right) \cdot \|P_0 \overline{u}\|.
\end{aligned}$$

Finally, using $\|P_0\overline{u}\| = \sqrt{\rho_0 N \cdot (1/N)^2} = \rho_0/\sqrt{N}$, we establish Eq. (3). $\blacksquare$

Rapid mixing. A property related to Lemma 12 is that a random walk starting at any vertex converges to the uniform distribution on the expander vertices after a logarithmic number of steps. Using notation as in the proof of Eq. (3), we claim that for every starting distribution $\overline{s}$ (including one that assigns all weight to a single vertex), it holds that $\|A^\ell\overline{s} - \overline{u}\|_1 \leq \sqrt{N} \cdot \hat{\lambda}^\ell$, which is meaningful for any $\ell > 0.5 \cdot \log_{1/\hat{\lambda}} N$. The claim is proved by recalling that $\|A^\ell\overline{s} - \overline{u}\|_1 \leq \sqrt{N} \cdot \|A^\ell\overline{s} - \overline{u}\|$ and using the fact that $\overline{s} - \overline{u}$ is orthogonal to $\overline{u}$ (because the former is a zero-sum vector). Thus, $\|A^\ell\overline{s} - \overline{u}\| \leq \hat{\lambda}^\ell \|\overline{s} - \overline{u}\|$ and using $\|\overline{s} - \overline{u}\| < 1$ the claim follows.

2.2 Constructions

Many explicit constructions of expanders were given, starting in [8] and culminating in the optimal construction of [7] where $\lambda = 2\sqrt{d-1}$. Most of these constructions are quite simple (see, e.g., §2.2.1), but their analysis is based on non-elementary results from various branches of mathematics. In contrast, the construction of Reingold, Vadhan, and Wigderson [12], presented in §2.2.2, is based on an iterative process, and its analysis is based on a relatively simple algebraic fact regarding the eigenvalues of matrices.

Before turning to these explicit constructions we note that it is relatively easy to prove the existence of 3-regular expanders, by using the probabilistic method and referring to the combinatorial definition of expansion.[7]

[7] As a warm-up, one may establish the existence of d-regular expanders, for some constant d. In particular, towards dealing with the case of $d = 3$, consider a random graph G on the vertex set $V = \{0, ..., n-1\}$ constructed by augmenting the fixed edge set $\{\{i, i+1 \bmod n\} : i = 0, ..., n-1\}$ with $d-2$ uniformly (and independently) chosen perfect matchings of the vertices of $F \stackrel{\text{def}}{=} \{0, ..., (n/2)-1\}$ to the vertices of $L \stackrel{\text{def}}{=} \{n/2, ..., n-1\}$. Noting that $|(\Gamma_G(S \cap F) \cap F)| \geq |S \cap F| - 1$ (and similarly for L), and assuming without loss of generality that $|S \cap F| \geq |S \cap L|$, we focus on upper-bounding the probability that, for some $\varepsilon > 0$, there exists a set $S \subset V$ of size at most $n/2$ such that $|(\Gamma_G(S \cap F) \cap L) \setminus \Gamma_G(S \cap L)| < \varepsilon|S|$. Fixing such a set S, the corresponding probability is upper-bounded by p_S^{d-2}, where

$$p_S \stackrel{\text{def}}{=} \frac{\binom{(n/2)-\ell}{\varepsilon|S|} \cdot \binom{\ell+\varepsilon|S|}{|S \cap F|}}{\binom{n/2}{|S \cap F|}}$$

where $\ell = |\Gamma_G(S \cap L) \cap L|$. Indeed, we may focus on the case that $|S \cap F| \leq \ell + \varepsilon|S|$ (because in the other case $p_S = 0$), and observe that for sufficiently small $\varepsilon > 0$ it holds that $p_S < \binom{n}{|S|}^{-2/5}$. This suffices if $d \geq 5$. To deal with the case $d = 3$, we reduce the analysis of the expansion of sets S to the case that the set S consists of relatively long arithmetic sequences that use an increment of either 1 or 2, where relatively long means longer than some sufficiently large constant $t = 1/\sqrt{\varepsilon}$. (Note that if S contains more that $2\sqrt{\varepsilon} \cdot |S|$ elements in relatively short arithmetic sequences then $|\Gamma_G(S)| > 2\sqrt{\varepsilon}|S|/2t$, and prove expansion factor at least $\varepsilon' = 4\sqrt{\varepsilon}$ for the rest.) The reduction allows the application of the union bound, while considering only $2^{|S|/t} \cdot \binom{n}{|S|/t}$ possible sets S (rather than $\binom{n}{|S|}$ such sets).

2.2.1 The Margulis–Gabber–Galil Expander

For every natural number m, consider the graph with vertex set $\mathbb{Z}_m \times \mathbb{Z}_m$ and edge set in which every $\langle x, y\rangle \in \mathbb{Z}_m \times \mathbb{Z}_m$ is connected to the vertices $\langle x \pm y, y\rangle$, $\langle x \pm (y+1), y\rangle$, $\langle x, y \pm x\rangle$, and $\langle x, y \pm (x+1)\rangle$, where the arithmetic is modulo m. This yields an extremely simple and explicit 8-regular graph with second eigenvalue that is bounded by a constant $\lambda < 8$ that is independent of m. Thus we get:

Theorem 13 *For some constant $\lambda < 8$ there exists a strongly explicit construction of a family of $(8, \lambda)$-expanders for $\{m^2 : m \in \mathbb{N}\}$. Furthermore, the neighbors of a vertex can be computed in logarithmic-space.*[8]

An appealing property of Theorem 13 is that, for every $n \in \mathbb{N}$, it directly yields expanders with vertex set $\{0,1\}^n$. This is obvious in case n is even, but can be easily achieved also for odd n (e.g., use two copies of the graph for $n-1$, and connect the two copies by the obvious perfect matching).

Theorem 13 is due to Gabber and Galil [4], building on the basic approach suggested by Margulis [8]. It was later shown that $\lambda = 5\sqrt{2} < 7.1$. Recall, however, that the optimal construction of [7] achieves $\lambda = 2\sqrt{d-1}$ but there are restrictions on d (i.e., $d-1$ should be a prime congruent to 1 modulo 4) and on the graph sizes (i.e., the set S) for which this construction works.

2.2.2 The Iterated Zig-Zag Construction

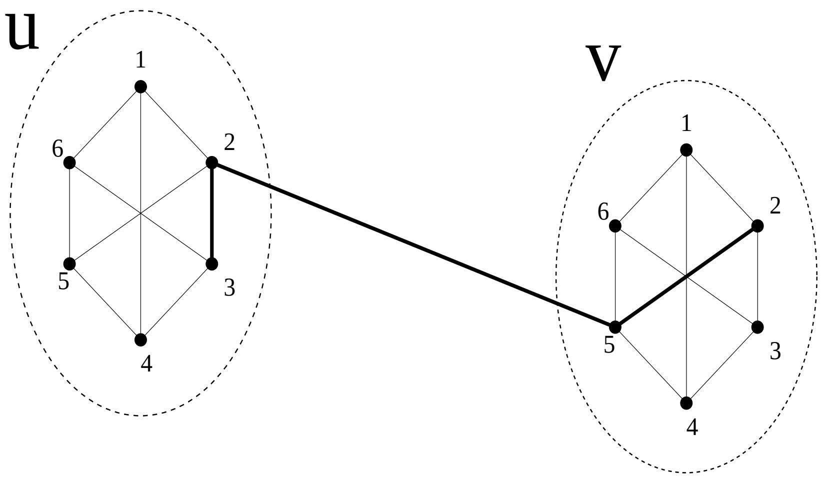
The starting point of the construction is a very good expander G of *constant size*, which may be found by an exhaustive search. The construction of a large expander graph proceeds in iterations, where in the i^{th} iteration the current graph G_i and the fixed graph G are combined, resulting in a larger 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 Zig-Zag product. 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 $G'ⓩG$, is defined as a graph with vertex set $[N] \times [D]$ and an edge set that includes an 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^{th} edge incident at u equals the ℓ^{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, denoted $R' : [N] \times [D] \to [N] \times [D]$, such that $R'(u, i) = (v, j)$ if (u, v) is the i^{th} edge incident at u as well as the j^{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

$$(\langle u, i\rangle, \langle \alpha, \beta\rangle) \mapsto (\langle v, j\rangle, \langle \beta, \alpha\rangle) \quad \text{if } R'(u, E_\alpha(i)) = (v, E_\beta(j)). \tag{5}$$

[8] In fact, under a suitable encoding of the vertices and for m that is a power of two, the neighbors can be computed by a on-line algorithm that uses a constant amount of space.



In this example G' is 6-regular and G is a 3-regular graph having six vertices. In the graph G' (not shown), the 2nd edge of vertex u is incident at v, as its 5th edge. The wide 3-segment line shows one of the corresponding edges of $G' \textcircled{z} G$.

Figure 1: Detail of the zig-zag product of G' and G.

That is, edges are labeled by pairs over $[d]$, and the $\langle \alpha, \beta \rangle^{\text{th}}$ edge out of vertex $\langle u, i \rangle \in [N] \times [D]$ is incident at the vertex $\langle v, j \rangle$ (as its $\langle \beta, \alpha \rangle^{\text{th}}$ 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 $(v, j') \stackrel{\text{def}}{=} R'(u, E_\alpha(i))$, 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 $j' = E_\beta(E_\beta(j')) = E_\beta(j)$.)

Clearly, the graph $G' \textcircled{z} G$ is d^2-regular and has $D \cdot N$ vertices. The key fact, proved in [12], is that the relative eigenvalue of the zig-zag product is upper-bounded by the sum of the relative eigenvalues of the two graphs (i.e., $\lambda_2(G' \textcircled{z} G) \leq \lambda_2(G') + \lambda_2(G)$, where $\lambda_2(\cdot)$ denotes the relative eigenvalue of the relevant graph).

The iterated construction. 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 $\lambda_2(G) < 1/4$, 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.

   (The degree bound 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 $\lambda_2(G_{i-1}^2 \textcircled{z} G) \leq \lambda_2(G_{i-1}^2) + \lambda_2(G)$, which in turn equals $\lambda_2(G_{i-1})^2 + \lambda_2(G) < (1/2)^2 + (1/4)$. 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 depth 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.

The strongly explicit version. To achieve a *strongly explicit construction*, we slightly modify the iterative construction. Rather than letting $G_{i+1} = G_i^2 \circledz G$, we let $G_{i+1} = (G_i \times G_i)^2 \circledz G$, where $G' \times G'$ denotes the *tensor product of G' with itself*; that is, 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 $G_i = (G_{i-1} \times G_{i-1})^2 \circledz G$ is a d^2-regular graph with $(D^{2^{i-1}-1})^2 \cdot D = D^{2^i-1}$ vertices, and $\lambda_2(G_i) < 1/2$ (because $\lambda_2((G_{i-1} \times G_{i-1})^2 \circledz G) \leq \lambda_2(G_{i-1})^2 + \lambda_2(G)$). 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 1). 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.

Stronger bound regarding the effect of the zig-zag product. In the foregoing description we relied on the fact, proved in [12], that the relative eigenvalue of the zig-zag product is upper-bounded by the sum of the relative eigenvalues of the two graphs. Actually, a stronger upper-bound is proved in [12]: For $g(x, y) = (1 - y^2) \cdot x/2$, it holds that

$$\begin{aligned} \lambda_2(G' \circledz G) &\leq g(\lambda_2(G'), \lambda_2(G)) + \sqrt{g(\lambda_2(G'), \lambda_2(G))^2 + \lambda_2(G)^2} \\ &\leq 2g(\lambda_2(G'), \lambda_2(G)) + \lambda_2(G) \\ &= (1 - \lambda_2(G)^2) \cdot \lambda_2(G') + \lambda_2(G). \end{aligned} \tag{6}$$

Thus, we get $\lambda_2(G'ⓩG) \leq \lambda_2(G') + \lambda_2(G)$. Furthermore, Eq. (6) yields a non-trivial bound for any $\lambda_2(G'), \lambda_2(G) < 1$, even in case $\lambda_2(G')$ is very close to 1 (as in Reingold's constriction [11]; see proof of Theorem 3). Specifically, Eq. (6) is upper-bounded by

$$\begin{aligned} & g(\lambda_2(G'), \lambda_2(G)) + \sqrt{\left(\frac{1-\lambda_2(G)^2}{2}\right)^2 + \lambda_2(G)^2} \\ = \; & \frac{(1-\lambda_2(G)^2)\cdot\lambda_2(G')}{2} + \frac{1+\lambda_2(G)^2}{2} \\ = \; & 1 - \frac{(1-\lambda_2(G)^2)\cdot(1-\lambda_2(G'))}{2} \end{aligned} \quad (7)$$

Thus, $1 - \lambda_2(G'ⓩG) \geq (1-\lambda_2(G)^2)\cdot(1-\lambda_2(G'))/2$. In particular, if $\lambda_2(G) < 1/\sqrt{3}$ then $1 - \lambda_2(G'ⓩG) > (1-\lambda_2(G'))/3$. This fact plays an important role in the proof of Theorem 3).

Notes

Before turning to the actual credit, we mention that some people tend to be discouraged by the impression that "decades of research have failed to answer any of the famous open problems of complexity theory." In addition to the fact that substantial progress towards the understanding of many fundamental issues has been achieved, people tend to forget that some famous open problems were actually resolved. The current text is indeed a superb example.

For more than two decades, undirected connectivity was one of the most appealing examples of the computational power of randomness. Recall that 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, it was known (since 1979, see [1]) 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, this algorithm (as well as the entire class $\mathcal{BPL}$), was derandomized in polynomial-time and poly-logarithmic space (see [9, 10]), but despite more than a decade of research attempts, a significant gap remained between the space complexity of randomized and deterministic polynomial-time algorithms for this natural and ubiquitous problem. This gap was closed by Reingold [11], who established Theorem 3 in 2004. Our presentation follows his ideas, but the specific formulation in Section 1.2 is new.

Exercise 14 (UCONN in constant degree graphs of logarithmic diameter) Present a log-space algorithm for deciding the following promise problem, which is parameterized by constants c and d. The input graph satisfies the promise if each vertex has degree at most d and every pair of vertices that reside in the same connected component are connected by a path of length at most $c\log_2 n$, where n denotes the number of vertices in the input graph. The task is to decide whether the input graph is connected.

Guideline: For every pair of vertices in the graph, we check whether these vertices are connected in the graph. (Alternatively, we may just check whether each vertex is connected to the first vertex.) Relying on the promise, it suffices to inspect all paths of length at most $\ell \stackrel{\text{def}}{=} c\log_2 n$, and these paths can be enumerated using $\ell \cdot \lceil \log_2 d \rceil$ bits of storage.

Exercise 15 (warm-up towards Section 1.2) In continuation to Section 1.1, present a log-space transformation of G_i to G_{i+1}.

Guideline: Given the graph G_i as input, we may construct G_{i+1} by first constructing $G' = G_i^c$ and then constructing $G' \textcircled{z} G$. To construct G', we scan all vertices of G_i (holding the current vertex in temporary storage), and for each such vertex construct its neighborhood in G' (by using $O(c)$ space for enumerating all possible neighbors). Similarly, we can construct the vertex neighborhoods in $G' \textcircled{z} G$ (by storing the current vertex name and using a constant amount of space for indicating incident edges in G).

Exercise 16 (st-UCONN) In continuation to Section 1, prove that the following computational problem is in $\mathcal{L}$: Given an undirected graph $G = (V, E)$ and two designated vertices, s and t, determine whether there is a path from s to t in G.

Guideline: Note that the transformation described in Section 1 can be easily extended such that it map vertices in G_0 to vertices in $G_{O(\log|V|)}$ while preserving the connectivity relation (i.e., u and v are connected in G_0 if and only if their images under the map are connected in $G_{O(\log|V|)}$).

Exercise 17 (finding paths in undirected graphs) In continuation to Exercise 16, present a log-space algorithm that given an undirected graph $G = (V, E)$ and two designated vertices, s and t, finds a path from s to t in G (in case such a path exists).

Guideline: In continuation to Exercise 16, we may finds and store a logarithmic path in $G_{O(\log|V|)}$ that connects a representative of s and a representative of t. Focusing on the task of finding a path in G_0 that corresponds to an edge in $G_{O(\log|V|)}$, we note that such a path can be found by using the reduction underlying the combination of Claim 6 and Lemma 7. (Again, a direct description appears in [11].)

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] M. Ajtai, J. Komlos, E. Szemerédi. Deterministic Simulation in LogSpace. In *19th ACM Symposium on the Theory of Computing*, pages 132–140, 1987.

[3] N. Alon and J.H. Spencer. *The Probabilistic Method.* John Wiley & Sons, Inc., 1992.

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

[5] D. Gillman. A chernoff bound for random walks on expander graphs. In *34th IEEE Symposium on Foundations of Computer Science*, pages 680–691, 1993.

[6] N. Kahale, Eigenvalues and Expansion of Regular Graphs. *Journal of the ACM*, Vol. 42 (5), pages 1091–1106, September 1995.

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

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

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

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

[11] O. Reingold. Undirected ST-Connectivity in Log-Space. In *37th ACM Symposium on the Theory of Computing*, pages 376–385, 2005.

[12] 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.