Derandomization and interlacing polynomials
Constructive and nonconstructive methods in combinatorics and TCS

Lecturer: Andrew Drucker  Scribe: Nathan Mull
May, 2018

[Note: This document aims to be a helpful resource but has not received the level of formality or editorial scrutiny expected of formal publications. Please use accordingly.]

In the first part of these notes, we cover some results on derandomization and the construction of pseudorandom generators (PRGs). We look at Nisan’s PRG for space-bounded computation, which allows use to simulate poly(n) random bits with \(O(\log^2(n))\) random bits in \textsc{Logspace} \cite{11}. Roughly speaking, this is the strongest known unconditional derandomization result. We also look at the Nisan-Widgerson PRG \cite{12} which, in tandem with hardness amplification results \cite{4}, is strong enough to prove that \(\text{BPP} = \text{P}\) under the assumption \(\text{E}\) has a language whose decision problem requires \(2^{\Omega(n)}\)-sized circuits.

In the second part of these notes, we cover some recent results related to interlacing polynomials. We discuss some of the basic theory, and use results from this area to sketch the proof of a result of Marcus, Spielman, and Srivastava that there exist infinite families of bipartite Ramanujan graphs of arbitrary degree.

1 Basics of space-bounded computation

We won’t worry too much about the details of our Turing machine model, but will work roughly with the following definitions.

**Definition 1.** (Definition 4.1 from [1]) \(L \in \text{DSPACE}(S(n))\) if there is a constant \(c\) and Turing machine \(M\) deciding \(L\) such that for each \(\{0,1\}^n\), the head of \(M\) visits at most \(\leq c \times S(n)\) locations on its work tapes.

**Definition 2.** (Definition 4.5 from [1])

\[
\text{PSPACE} = \bigcup_{i=1}^{\infty} \text{DSPACE}(n^c)
\]

\[
\text{LOGSPACE} = \text{DSPACE}(\log n)
\]
The class $\text{NSPACE}(S(n))$ is defined analogously to its time-bounded counterpart, i.e. it has the same definition as above but with $M$ being a nondeterministic TM.

Since there is no restriction on running time in the definition of space-bounded TMs, space-bounded computation is very powerful. With $S$ space, it is possible to “test” $2^S$ possible strings.

**Theorem 1.**

- $\text{NP} \subseteq \text{PSPACE}$
- $\text{NLOGSPACE} \subseteq \text{P}$
- $\text{PSPACE} = \text{NPSPACE}$
- $\text{NSPACE}(S) \subseteq \text{DSPACE}(S^2)$ (see [13])
- For every space-constructible $S > \log n$, $\text{NSPACE}(S) = \text{co-NSPACE}(S)$ (see [3])

The classes $\text{RSPACE}(S(n))$ and $\text{BPSPACE}(S(n))$ are also defined analogously to their time-bounded counterparts, but we have to be a bit careful about the how to provide randomness to space-bounded TMs. We will use the model in which random bits are given as a sequence that the TM can read in a one-directional pass. In particular, a TM cannot query particular random bits, so if it wants to “remember” a random bit it sees, it has to write it down. This means that space-bounded computation can be modeled by finite state machines (FSMs) with the random bits as input.

**Fact 1.** A randomized TM with space bounded by $S$ may be equivalently thought of as an FSM with $2^S$ states over the alphabet $\{0, 1\}$.

Nisan’s generator is explicitly designed to work for such FSMs.

## 2 Nisan’s generator

The goal of Nisan’s generator is to transform a small number of random bits into a large number of bits that “look” random to FSMs. In particular, an FSM should have negligible advantage on these transformed bits when compared to the same number of purely random bits.

**Definition 3.** (Definition 1 from [11]) A **PRG for $\text{DSPACE}(S)$ with parameter $\epsilon$** is a function $G : \{0, 1\}^m \rightarrow \{0, 1\}^k$ such that for every FSM $M$ with $2^S$ states over $\{0, 1\}$,

$$\left| \Pr_{x \in \{0, 1\}^m} (M \text{ accepts } G(x)) - \Pr_{x \in \{0, 1\}^k} (M \text{ accepts } x) \right| \leq \epsilon.$$
The idea of the generator is to apply a bunch of specially chosen functions to a given random seed in different combinations, and then concatenate all the results. As is typical in these constructions, we want the functions to have some form of independence, in that it should be sufficiently hard to tell the input and output from randomly chosen strings from the domain and range. Naturally, this idea leads to the construction of PRGs via one-way functions, or functions for which it is sufficiently difficult to guess the input given the output [6]. Part of the innovation of Nisan’s construction is that it uses the weaker form of independence found in hash families.

**Definition 4.** (Definition 2 from [11]) A pairwise independent hash family from $U$ to $\sigma$ is a set of functions $H \subseteq \sigma^U$ such that for $x_1, x_2 \in U$ with $x_1 \neq x_2$ and $y_1, y_2 \in \Sigma$,

$$\Pr_{h \in H}(h(x_1) = y_1 \text{ and } h(x_2) = y_2) = \frac{1}{|\Sigma|^2}.$$ 

Before defining the generator, it is worth noting that the hash functions will become part of the seed in the construction. This means the hash functions should be somewhat succinctly representable. In particular, if $U = \{0, 1\}^n$ and $\Sigma = \{0, 1\}^m$, then each function should be representable by $O(n + m)$ bits.

**Definition 5.** (Nisan’s generator, section 3.3 of [11]) Fix a pairwise independent hash family $H$ from $\{0, 1\}^n$ to $\{0, 1\}^n$. For every $k \geq 0$, define the generator $G_k : \{0, 1\}^n \times H^k \to \{0, 1\}^{2^k n}$ recursively as follows.

$$G_0(x) = x$$

$$G_k(x, h_1, h_2, \ldots, h_k) = (G_{k-1}(x, h_1, h_2, \ldots, h_{k-1}), G_{k-1}(h_k(x), h_1, h_2, \ldots, h_k))$$

where $(\ast, \ast)$ is concatenation.

The first few generators look like:

$$G_0(x) = x$$

$$G_1(x, h_1) = (x, h_1(x))$$

$$G_2(x, h_1, h_2) = (x, h_1(x), h_2(x), h_1(h_2(x)))$$

$$G_3(x, h_1, h_2, h_3) = (x, h_1(x), h_2(x), h_1(h_2(x)), h_2(h_3(x)), h_1(h_3(x)), h_2(h_3(x)), h_1(h_2(h_3(x))))$$

$$\vdots$$

We can see from the above that for $G_k$, there is a composition of functions from $H$ corresponding to every element in $\{0, 1\}^k$, which provides the exponential growth in $k$ of the output. Informally, the result proved by Nisan is the following.

**Theorem 2.** (Informal version of Theorem 1 from [11]) The generator $G_{O(S \log R)}$ is a PRG for $\text{DSPACE}(S)$ with parameter $2^{-S}$. 


So $O(\log n^2)$ random bits is sufficient to simulate polynomially many random bits in LOGSPACE. The primary tool for the analysis is essentially a mixing lemma for pairwise independent hash families. Nisan proves that, with high probability, $x$ and $h(x)$ are likely to be roughly independent with respect to set containment.

**Definition 6.** (Definition 3 from [11]) Fix $A \subseteq \{0,1\}^n, B \subseteq \{0,1\}^m$ and $h : \{0,1\}^n \rightarrow \{0,1\}^m$. $h$ is $\epsilon$-independent for $A$ and $B$ if

$$\Pr_{x \in \{0,1\}^n} (x \in A \text{ and } h(x) \in B) \in [\alpha\beta - \epsilon, \alpha\beta + \epsilon]$$

for $\alpha = |A|/2^n$ and $\beta = |B|/2^m$.

**Lemma 1.** (Pairwise independence mixing lemma, Lemma 1 from [11]) For pairwise independent hash family $\mathcal{H}$,

$$\Pr_{h \in \mathcal{H}} (h \text{ is not } \epsilon\text{-independent for } A \text{ and } B) \leq \frac{\alpha\beta}{2^k\epsilon^2}.$$


### 3 $P = \text{BPP}$ if $E$ requires exponential circuits

Much of what is covered in this section follows [1].

**Definition 7.**

$$E = \text{DTIME}(2^{O(n)})$$

To make our lives simpler, we will simultaneously think of $E$ as a class of languages and a class of functions deciding those languages.

**Theorem 3.** (Theorem 20.7 from [1]) If there is a language $L \in E$ that requires circuits of size $2^{\Omega(n)}$, then $P = \text{BPP}$.

The proof is broken into three phases.

1. (Hardness amplification via error correcting codes) First, we convert the given hard function $f \in E$ into a new function $f'$ that is hard to compute correctly on more 99% of the inputs. This is done by putting the truth table of $f$, $\text{tt}(f)$, through an error correcting code $C$ and defining $f'(x) \triangleq C(\text{tt}(f))_x$. If $f'$ has small circuits that are correct on most of the inputs, then decoding procedures for $C$ allow us to construct small circuits for $f$.

2. (Further hardness amplification via an XOR lemma) Next, we convert $f'$ to an extremely hard function $f''$ that is hard to compute correctly on slightly more than half of the inputs. This conversion uses the principle that uncertainty in $f'(x)$ should imply even more uncertainty in $\oplus_{i=1}^n f'(x_i)$. The challenge of this phase is ensuring that the newfound hardness of $f''$ is not saturated by its larger relative input size. This is done by derandomizing the classical XOR lemma.
3. (PRG from extremely hard function) Finally, we use \( f'' \) to build a PRG for large circuits. This is done by applying our hard function \( f'' \) to a collection of sufficiently independent subsets of our seed, and then concatenating the results. If this sequence of bits is not pseudorandom, then the circuit witnessing this fact will be able to compute \( f'' \) well enough to contradict its hardness.

Before continuing, we present the following definitions that we will use throughout the rest of this section.

**Definition 8.** A function \( f \) is \((S(n), \epsilon)-\text{hard}\) if for any circuit \( C \) of size at most \( S(n) \),
\[
\Pr_{x \in \{0,1\}^n} (C(x) = f(x)) < \epsilon.
\]

**Definition 9.** (Roughly Definition 20.2 from [1]) A generator \( G : \{0,1\}^m \rightarrow \{0,1\}^n \) is \((S(n), \epsilon)-\text{pseudorandom}\) if for any circuit \( C \) of size at most \( S(n) \),
\[
\left| \Pr_{x \in \{0,1\}^n} (C(x) = 1) - \Pr_{x \in \{0,1\}^m} (C(G(x)) = 1) \right| \leq \epsilon.
\]

**Phase 1: Hardness amplification via local decoding**

The principles of this phase as described above are fairly straightforward. The only issue is that standard decoding procedures for ECCs typically output an entire message from a given codeword, which is a problem when the message size is exponential in \( n \) (as it is in our case). To fix this, we use local decoding.

**Definition 10.** (Definition 19.16 from [1]) For ECC \( E : \{0,1\}^n \rightarrow \{0,1\}^m \), a local decoder for handling error \( \epsilon \) is given
- \( j \in \mathbb{Z}^+ \),
- random access to \( y \in \{0,1\}^m \) within Hamming distance \( \epsilon \) of \( E(x) \) for some \( x \in \{0,1\}^n \),
and outputs \( x_j \) with probability \( \geq 2/3 \) in time \( \text{polylog}(m) \).

In our setting, computing a single bit \( \text{tt}(f)_x \) is equivalent to determining \( f(x) \).

**Theorem 4.** (Theorem 19.21 from [1]) There is a constant \( c > 0 \) and ECC \( C : \{0,1\}^N \rightarrow \{0,1\}^{Nc} \) such that \( f' \), as defined above, is in \( \mathcal{E} \) and is \((2^{-1}n/\text{poly}(n), 0.99)-\text{hard}\).

**Proof.** (sketch of proof from [1]) Most of the proof of this fact is showing that there is some ECC that has a local-decoding algorithm with good parameters (Reed-Muller concatenated with Walsh-Hadamard is sufficient). We skip this part and work directly with the properties of the ECC that we need:
1. \( C(\tt(f)) \) is computable in time \( \text{poly}(N) = 2^{O(n)} \), and

2. the local decoding algorithm for \( C \) runs in time \( \text{poly log}(N) = \text{poly}(n) \).

The fact that \( f' \in E \) follows from the first property. Now suppose that there is some circuit \( D \) of size at most \( 2^n / \text{poly}(n) \) that computes \( f' \) on \( cn \) bits that is correct on more than 99% of all inputs. Then we can compute \( f \) on \( n \) bits exactly by building a circuit version \( B \) of the local decoder (even though the decoder has randomness, it can be made deterministic in the nonuniform setting by amplifying the success probability and then hard-coding a good choice of random bits) where \( D \) is used whenever the decoder needs random access to \( C(\tt(f)) \). This circuit for \( f \) will have size at most \( |D| \ast |B| \), which will be smaller that \( 2^{\Omega(n)} \) given that the ECC has good parameters.

The first property from the proof sketch implicitly takes advantage of the fact that \( 2^{O(n)} \ast 2^{O(n)} = 2^{O(n)} \). Namely, we can compute every bit of \( f \) and still sit in the class \( E \). This closure property is one of the main reasons we are working with \( E \) as opposed to a weaker class. It is also worth noting that the above proof sketch explicitly takes advantage of non-uniformity to deterministically simulate the randomness inherent in the local decoder.

**Phase 2: Derandomized XOR lemma**

Now that we have a mildly-hard-on-average function \( f' \), we need an extremely-hard-on-average function \( f'' \). This can be done via the following classical XOR lemma. Define \( f \oplus_k : \{0,1\}^k \to \{0,1\} \) by

\[
 f \oplus_k(x_1, x_2, \ldots, x_k) \triangleq \bigoplus_{i=1}^{\ast} f(x_i). 
\]

**Theorem 5.** (Theorem 19.2 from [1], originally in [17]) If \( f \) is \((S(n),0.99)\)-hard, then \( f \oplus_k \) is \((0.99)^2 \ast S(n)/2^\ast n, 1/2 + 4(0.99)^k\)-hard.

The issue with this classical XOR lemma is that the input size to \( f'' \) with respect to that of \( f' \) is too large for our purposes. Impagliazzo and Wigderson noticed that this blow-up in input size occurs because the proof of this theorem uses the independence of XOR’d inputs. To make the input smaller, they replaced the independent inputs with a sequence of pseudorandom bits. That is, for PRG \( G : \{0,1\}^m \to \{0,1\}^{kn} \), they considered the hardness of a function \( f_{G}^{\oplus k} : \{0,1\}^m \to \{0,1\} \) defined by \( f_{G}^{\oplus k}(x) \triangleq f^{\oplus k}(G(x)) \). Furthermore, they give general conditions on PRGs so that \( f_{G}^{\oplus k} \) is extremely-hard-on-average. These conditions enforce sufficient but weaker forms of independence necessitated by their new proof of the XOR lemma. See [4] (in particular, Theorem 15) for further details.

**Phase 3: Nisan-Wigderson PRG**

Given \( f'' = (f')^{\oplus kn}_G \), we now want to design a PRG for very large circuits. We are roughly aiming for \((2^{cn}, \epsilon)\)-pseudorandomness since
1. we want our seed length to be \( O(\log n) \), and

2. we want our pseudorandomness to work against polynomial computation.

The first tool we need is a simple corollary of Yao’s theorem, which essentially states that unpredictability implies pseudorandomness. We provide the proof here because of its interesting use of the hybrid argument.

**Theorem 6.** (Theorem 20.10 from [1]) For distribution \( Y \) over \( \{0,1\}^m \) and \( \epsilon > 0 \), if there is a circuit of size \( S \) such that

\[
\left| \Pr_{x \sim Y}(C(x) = 1) - \Pr_{x \in_R \{0,1\}^m}(C(x) = 1) \right| \geq \epsilon
\]

then there is a circuit \( C' \) of size at most \( 3S \) and index \( i \in [m] \) such that

\[
\Pr_{x \sim Y}(C'(x_1, x_2, \ldots, x_{i-1}) = x_i) \geq \frac{1}{2} + \frac{\epsilon}{m}.
\]

**Proof.** Without loss of generality, suppose \( \Pr_{x \sim Y}(C(x) = 1) - \Pr_{x \in_R \{0,1\}^m}(C(x) = 1) \geq \epsilon \) (otherwise take the negation of \( C \)). Let \( D_i \) be the distribution on \( \{0,1\}^m \) induced by taking the first \( i \) bits of \( x \sim Y \) concatenated with the last \( m - i \) bits from \( z \in_R \{0,1\}^m \). Note that \( D_0 \) is the uniform distribution on \( \{0,1\}^m \) and \( D_m \) is \( Y \). By averaging, there is some \( i \) such that

\[
\Pr_{x \sim D_i}(C(x) = 1) - \Pr_{x \sim D_{i-1}}(C(x) = 1) \geq \frac{\epsilon}{m}.
\]

Written another way,

\[
\Pr_{x \sim Y, z \in_R \{0,1\}^m}(C(x_1, \ldots, x_i, z_{i+1}, \ldots, z_m) = 1) - \Pr_{x \in_R \{0,1\}^m}(C(x_1, \ldots, x_{i-1}, z_i, \ldots, z_m) = 1) \geq \frac{\epsilon}{m}.
\]

By another averaging argument, there is some \( z' \in \{0,1\}^m \) such that

\[
\Pr_{x \sim Y}(C(x_1, \ldots, x_i, z'_{i+1}, \ldots, z'_m) = 1) - \Pr_{z \sim \{0,1\}^m}(C(x_1, \ldots, x_{i-1}, z_i, \ldots, z'_m) = 1) \geq \frac{\epsilon}{m}.
\]

Hard-code these bits into \( C \), and call this new circuit \( C^* \). Then we have

\[
\Pr_{x \sim Y}(C^*(x_1, \ldots, x_i) = 1) - \Pr_{z \in \{0,1\}}(C^*(x_1, \ldots, x_{i-1}, z) = 1) \geq \frac{\epsilon}{m}.
\]

Our goal now is to construct \( C' \) using \( C^* \). We see that \( C^* \) outputs 1 more often when the last index of its input comes from \( x \) instead of \( z \). So we can use \( C^* \) as a test for the value of \( x_i \). For \( a \in \{0,1\} \) define \( C^*_a(x_1, \ldots, x_{i-1}) \triangleq C^*(x_1, \ldots, x_{i-1}, a) \). By another averaging argument, there are \( a, a' \in \{0,1\} \) such that \( C^*_a \lor \neg C^*_a' \) has the desired advantage in predicting \( x_i \). Here, we essentially build the deterministic nonuniform version of the following randomized algorithm: given \( x_1, x_2, \ldots, x_{i-1} \) for \( x \sim Y \), choose \( a \in_R \{0,1\} \), return \( a \) if \( C^*(x_1, x_2, \ldots, x_{i-1}, a) = 1 \) and \( 1 - a \) otherwise.
The definition of the generator is fairly simple: we apply our hard function $f''$ to a collection of subsets of bits from our seed and append all the results together. With the above tool in hand, the principle of the generator is that predicting the next bit in the sequence is akin to computing $f''$.

**Definition 11.** (Nisan-Widgerson generator, Definition 20.12 from [1]) Let $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ be a family of subsets of $[u]$, each of size $n$, and let $f$ be a boolean function on $n$ inputs. Define the generator $\text{NW}_f^I: \{0,1\}^u \to \{0,1\}^m$ by

$$\text{NW}_f^I(x) \triangleq (f(x_{I_1}), f(x_{I_2}), \ldots, f(x_{I_m}))$$

where $f(x_I) = f(x_{i_1}, x_{i_2}, \ldots, x_{i_n})$ for $I = \{i_1, i_2, \ldots, i_n\}$ with $i_1 < i_2 < \cdots < i_n$.

The idea of the following analysis is that if we assume $\text{NW}_f^I$ is not pseudorandom, then it is predictable at some bit, and being able to predict that bit from previous bits should give us non-negligible advantage in computing $f''$. Therefore, it must be that our ability to predict a given bit is not simply because the previous bits “leak” too much information about $f''$. In the extremal case, if $I_j = I_k$, for $j < k$, then it would be very easy to predict the bit $f''(I_k)$ from previous bits, and predicting this bit does not give much insight on how to compute $f''$. So again, we are reaching for an independence condition, but this time on our choice of subsets. For this, we use combinatorial designs.

**Definition 12.** (Definition 20.13 from [1]) Fix $l, n, u \in \mathbb{Z}^+$ such that $l < n < u$. A $(u,n,l)$-design is a family of subsets $\mathcal{I} = \{I_1, I_2, \ldots, I_m\}$ such that for distinct $j,k \in [m]$,

- $I_j \subseteq [u]$,
- for every $|I_j| = n$, and
- $|I_j \cap I_k| = l$.

There is an efficient greedy deterministic algorithm for constructing combinatorial designs with good parameters. For further details, see Lemma 20.14 from [1].

Finally, we can put everything together to prove that $\text{NW}_f^I$ is a pseudorandom generator.

**Theorem 7.** (Lemma 20.15 from [1]) If $\mathcal{I}$ is a $(u,n,l)$-design of size $2^{d/10}$ and $f$ is $(S,1/2 + 1/S)$-hard for $S > 2^{2d}$, then $\text{NW}_f^I$ is $(S/10,1/10)$-pseudorandom.

**Proof.** Suppose $\text{NW}_f^I$ is not pseudorandom. Then there is some $i$, and circuit $C$ of size at most $3S/10$ such that

$$\Pr_{x \in \{0,1\}^u} (C(f(x_{I_1}), \ldots, f(x_{I_{i-1}})) = f(x_{I_i})) \geq \frac{1}{2} + \frac{1}{10 * 2^{2d/10}}.$$
Now split each \( x_I \) for \( j < i \) by its intersection with \( I_i \), i.e. \( x_1^{(j)} \triangleq x_{I_j \cap I_i} \) and \( x_2^{(j)} \triangleq x_{I_i - I_j} \). Then we have

\[
\Pr_{x \in \{0,1\}^n} \left( C(f(x_1^{(1)}, x_2^{(1)}), \ldots, f(x_1^{(i-1)}, x_2^{(i-1)})) = f(x_I) \right) \geq \frac{1}{2} + \frac{1}{10 + 2^{d/10}}.
\]

By averaging, there is some \( x' \in \{0,1\}^n \) such that

\[
\Pr_{x \in \{0,1\}^n} \left( C(f(x_1^{(1)}, (x')_2^{(1)}), \ldots, f(x_1^{(i-1)}, (x')_2^{(i-1)})) = f(x_I) \right) \geq \frac{1}{2} + \frac{1}{10 + 2^{d/10}}.
\]

Define \( f_i(x_1^{(i)} \triangleq f(x_1^{(i)}, (x')_2^{(i)}) \) so we can write this as

\[
\Pr_{x \in \{0,1\}^n} \left( C(f_1(x_1^{(1)}), \ldots, f_{i-1}(x_1^{(i-1)})) = f(x_I) \right) \geq \frac{1}{2} + \frac{1}{10 + 2^{d/10}}.
\]

Since \( I \) is a combinatorial design, \( f_i \) is a function of \( |I_j \cap I_i| = d \) inputs, which means it has a trivial circuit of size \( d2^d \). By hard-wiring these circuits into \( C \), we have a circuit \( B \) of size at most \( d2^d \cdot |I| + |I| \leq d2^{3d/10} + 3S/10 \leq S \) such that

\[
\Pr_{x \in \{0,1\}^n} \left( B(x) = f(x) \right) \geq \frac{1}{2} + \frac{1}{S}.
\]

Proving Theorem 3 is now a matter of chasing down parameters. By combining the three phases, we get a pseudorandom generator that can be used to simulate the polynomial time random TM on all possible seeds, all in polynomial time.

4 Interlacing Polynomials

The second part of these notes focuses on the study of interlacing polynomials. Interlacing polynomials have been studied for several decades, but was only until recently they have been used more extensively to prove the existence of combinatorial structures [7, 8, 10, 9]. The primary objects of study here are real-rooted polynomials, and the first observation is that the set of real-rooted polynomials is not closed under addition or averaging: the polynomials \( (X^2 + 1) \) and \( (X - 1)^2 \) are real-rooted, but their average \( X^2 + 1 \) does not have any real roots. Our goal is to understand under what conditions convex combinations of real-rooted polynomials are real-rooted.

**Definition 13.** (in Section 22.3 from [15]) Let \( P, Q \in \mathbb{R}[X] \) be real-rooted polynomials such that \( \deg(P) = n \) and \( \deg(Q) = n - 1 \). Let

\[
\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \quad \text{and} \quad \mu_1 \geq \mu_2 \geq \cdots \geq \mu_{n-1}
\]

be the roots of \( P \) and \( Q \), respectively. Say \( P \) and \( Q \) **interlace** if

\[
\lambda_1 \geq \mu_1 \geq \lambda_2 \geq \mu_2 \geq \cdots \geq \mu_{n-1} \geq \lambda_n.
\]
Fact 2. For $P \in \mathbb{R}[X]$, $P$ and $\frac{dP}{dX}$ interlace.

Definition 14. (in Section 22.3 from [15]) For real-rooted $P, Q \in \mathbb{R}[X]$ such that $\deg(P) = \deg(Q) = n$, let

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n \quad \text{and} \quad \mu_1 \geq \mu_2 \geq \cdots \geq \mu_n$$

be their respective roots. Say $P$ and $Q$ interlace with $P$ on top, written $Q \rightarrow P$, if

$$\lambda_1 \geq \mu_1 \geq \lambda_2 \geq \mu_2 \geq \cdots \geq \mu_{n-1} \geq \lambda_n \geq \mu_n.$$ 

Say $R \in \mathbb{R}[X]$ of degree $n$ is a common interlacing of $P$ and $Q$ if $P \rightarrow R$ and $Q \rightarrow R$.

It turns out that having a common interlacing is a sufficient condition for a set of polynomials to have real-rooted convex combinations. Assume now that all polynomials are real-rooted, degree $n$, and have positive leading coefficient unless otherwise stated.

Lemma 2. (Lemma 22.3.1 from [15]) Let $P, Q \in \mathbb{R}[X]$ be interlacing polynomials, where $\deg(P) = n$ and $\deg(Q) = n - 1$. Then for $t > 0$, $R_t = P - t \ast Q$ is real-rooted.

Lemma 3. (Lemma 22.3.4 from [15]) Let $P, Q \in \mathbb{R}[X]$ have common interlacing $R$. Then for $t \in [0, 1]$, the convex combination $C_t \triangleq tP + (1 - t)Q$ is real-rooted.

For our application, as well as many others, we care about largest roots. For example, the largest root of the characteristic polynomial of a matrix is the same the largest eigenvalue of the matrix. Let $\max \text{root}(P)$ denote the largest root of the polynomial $P$.

Lemma 4. (Lemma 24.3.1 from [16]) Let $P, Q \in \mathbb{R}[X]$ have common interlacing $R$. Then for all $t \in [0, 1]$,

$$\min \{ \max \text{root}(P), \max \text{root}(Q) \} \leq \max \text{root}(C_t).$$

Furthermore, if $P_1, P_2, \ldots, P_k \in \mathbb{R}[X]$ have common interlacing $R$, then for any $t_1, t_2, \ldots, t_k \in [0, 1]$ such that $\sum t_i = 1$,

$$\min \{ \max \text{root}(P_i) \} \leq \max \text{root} \left( \sum t_i P_i \right)$$

noting that $\sum t_i P_i$ is real-rooted.

This can all be taken one step further to create a more flexible tree-like structure of interlacing polynomials.

Definition 15. (in Section 24.3 from [16]) A set of polynomials $P_1, P_2, \ldots, P_m \in \mathbb{R}[X]$, is an interlacing family with (rooted) tree $T$ if
• Every leaf of \( T \) is labeled by some polynomial \( P_i \),
• every non-leaf node of \( T \) is labeled by a convex combination of its children, and
• all the children of a non-leaf node have a common interlacing.

**Theorem 8.** (Lemma 24.3.2 from [16]) Let \( P_1, P_2, \ldots, P_m \) be an interlacing family with tree \( T \), and let \( P^* \) be the polynomial labeling the root of \( T \). Then

\[
\min_i \\{ \max \text{ root}(P_i) \} \leq \max \text{ root}(P^*).
\]

**Remark.** In everything above, we could have replaced \( \max \text{ root}(P) \) with “the \( j \)-th largest root of \( P \).”

Our goal now is to make the root of an interlacing family some polynomial with a clean representation and easily-bounded roots. This will give us bounds on the largest root of some polynomial in our interlacing family.

Generally speaking, it can be difficult to exhibit an interlacing of polynomials. However, we have the fact that interlacing is in some sense equivalent to having real-rooted convex combinations. So the problem exhibiting an interlacing reduces to proving that certain classes of polynomials are real-rooted, a problem for which there are tools from functional analysis.

**Theorem 9.** (Lemma 3.5 from [8]) Polynomials \( P_1, P_2, \ldots, P_k \) have a common interlacing if and only if all convex combinations are real-rooted.

The proofs the above facts, with the exception of Theorem 9, are actually fairly simple. We leave them out for sake of space. See the corresponding references for further details.

### 5 Bipartite Ramanujan graphs

We now apply the ideas above to construct bipartite Ramanujan graphs. The polynomials for this application are characteristic polynomials of adjacency matrices. As noted above, the largest root of such a polynomial is the maximum eigenvalue, so upper bounds on the maximum root give us information about the spectrum of the graph.

For graph \( G = (V, E) \), let \( A_G \) denote its adjacency matrix, and let \( P_G \) denote its characteristic polynomial, i.e. \( P_G(x) = \det(xI - A_G) \). Let

\[
\lambda_1(G) \geq \lambda_2(G) \geq \cdots \geq \lambda_{|V|}(G).
\]

denote the eigenvalues of \( A_G \). From spectral graph theory, the problem of finding graphs with good expansion can be reduced to finding graphs with sufficiently small \( \lambda_2 \). Likewise, we know that for \( d \)-regular graphs, \( \lambda_2 \geq 2\sqrt{d-1} - o(n) \), where \( n = |V| \) (see [5] for references). Ramanujan graphs are essentially those graphs that meet this lower bound. As hinted above, we will consider the bipartite case.
Lemma 5. (Definition 4.9 from [14]) A d-regular bipartite graph is called Ramanujan if \( \lambda_2(G) \leq 2\sqrt{d-1} \).

Theorem 10. (Theorem 4.8 from [14]) For all \( d \geq 2 \), there is an infinite family of \( d \)-regular bipartite Ramanujan graphs.

The first useful concept for this construction is the idea of a 2-lifts introduced in [2]. The goal of 2-lifts is to construct a new bipartite graph with a predictable well-behaved spectrum, so that it is easy to reason about whether or not it is Ramanujan.

Definition 17. (in Section 4.2 of [14]) A signing is a function \( s : E \to \{-1,1\} \). The 2-lift of \( G \) with respect to \( s \), written \( G^s \), is defined as follow. The vertex set of \( G^s \) is \( V \cup V = \{(v_i : v \in V, i \in \{1,2\})\} \). For \((u,v) \in E\), define

\[
    b_{(u,v)} \triangleq \begin{cases} 
        \{(u_1, v_1), (u_2, v_2)\} & s(u,v) = 1 \\
        \{(u_1, v_2), (u_2, v_1)\} & s(u,v) = -1.
    \end{cases}
\]

The edge set of \( G^s \) is \( \bigcup_{e \in E} b_e \). The sign matrix of \( G \) with respect to \( s \), written \( \hat{A}_G^s \), is defined by \((\hat{A}_G^s)_{u,v} = s(u,v) * A_{G^s} \).

Fact 3. (Lemma 4.9 from [14])

\[
    A_{G^s} = \frac{1}{2} \begin{pmatrix} A_G + \hat{A}_G^s & A_G - \hat{A}_G^s \\ A_G - \hat{A}_G^s & A_G + \hat{A}_G^s \end{pmatrix}
\]

Lemma 5. (Lemma 4.9 from [14]) \( \text{Spec}(G^s) = \text{Spec}(G) \cup \text{Spec}(\hat{A}_G^s) \). In particular, \( \lambda_2(G^s) = \max\{\lambda_2(G), \lambda_1(\hat{A}_G^s)\} \).

The above lemma tells us that if we have a bipartite Ramanujan graph \( G \), then it suffices to show that there is always some signing \( s \) such that \( \lambda_1(\hat{A}_G^s) \leq 2\sqrt{d-1} \) so that \( G^s \) is bipartite Ramanujan. This process can be repeated iteratively to construct an infinite family.

To connect this back to interlacing polynomials, \( \lambda_1(\hat{A}_G^s) = \max \text{root}(P_{\hat{A}_G^s}) \), so it suffices to give an upper bound on this largest root. By observation,

\[
    \hat{A}_G^s = (-d)I + \sum_{(u,v) \in E} (e_u + s(u,v) * e_v)(e_u + s(u,v) * e_v)^T
\]

where \( e_v \) is a standard basis vector, so it is (almost) a sum of rank-1 PSD matrices. To prove Theorem 10, we need the following (nontrivial) facts.

Fact 4. Let \( A_1, A_2, \ldots, A_m \) be a set of random rank-1 PSD matrices, each with finite support, and define \( \mathcal{P} \triangleq \{ \sum_{i=1}^m P_{B_i} \mid \forall i(B_i \in \text{supp}(A_i)) \} \). Then \( \mathcal{P} \) is an interlacing family for a tree with \( \mathbb{E}[\sum_{i=1}^m A_i] \) labeling the root. In particular, convex combinations of characteristic polynomials of rank-1 PSD matrices are real-rooted (note that the characteristic polynomial of a rank-1 PSD matrix is naturally real-rooted) and we can apply Theorem 9. This is proved in Section 2.1 from [14].
Fact 5. The polynomial $E_s \in R \{−1, 1\}^E[\hat{A}^s_G]$ has a nice characterization as the matching polynomial

$$\mu_G(X) \triangleq \sum_{k=0}^{\lfloor n/2 \rfloor} X^{n-2k}(-1)^k m_k(G).$$

where $m_k(G)$ is the number of matchings in $G$ with $k$ edges. Furthermore, if $G$ is $d$-regular, then $\max \text{ root}(\mu_G) \leq 2\sqrt{d-1}$. This is proved in Section 4.3 from [14].

Putting everything together, we have the following proof sketch of theorem 10.

Proof. (sketch) First, we prove that there is always some 2-lift of a bipartite Ramanujan graph that is bipartite Ramanujan (note that 2-lifts are clearly bipartite). In order to apply Fact 4, we focus our attention to $A_s(G) + dI$, which is exactly a sum of rank-1 PSD matrices. Then we have the following inequalities.

$$\min_s\{\lambda_1(A_s(G))\} = \min_s\{\lambda_1(A_s(G) + dI)\} - d = \min_s\{\max \text{ root}(P_{A_s(G) + dI})\} - d \leq \max \text{ root}(\mathbb{E}[P_{A_s(G) + dI}]) - d = \max \text{ root}(\mathbb{E}[\det((x-d)I - A_s(G))]) - d = \max \text{ root}(\mathbb{E}[P_{A_s(G)}]) + d - d = \max \text{ root}(\mathbb{E}[P_{A_s(G)}]) \leq 2\sqrt{d-1} \quad \text{(by Fact 5)}$$

Therefore there is some $s^*$ such that $\lambda_2(G^{(s^*)}) = \min\{\lambda_1(A_{s^*}(G)), \lambda_2(G)\} \leq 2\sqrt{d-1}$, which implies $G^{(s^*)}$ is bipartite Ramanujan. Therefore, it suffices to find a bipartite Ramanujan graph to start this iterative process. The graph $K_{d,d}$ happens to be bipartite Ramanujan, so we are done. 

References


