Pseudorandom Generators For Group Products

Presenter: Aeren

May 31, 2021

Derandomization Problem Of Permutation Branching Program

- Randomness is an important computational resource.

Derandomization Problem Of Permutation Branching Program

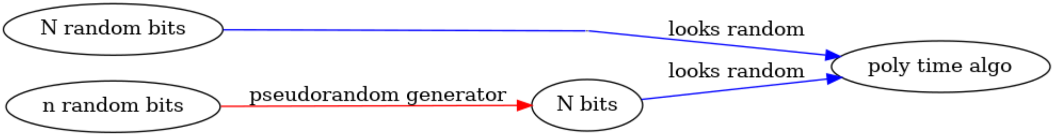
- Randomness is an important computational resource.
- Many computational problems are known to have much more efficient randomized algorithms than the deterministic ones.

Derandomization Problem Of Permutation Branching Program

- Randomness is an important computational resource.
- Many computational problems are known to have much more efficient randomized algorithms than the deterministic ones.
- Thus it is natural to attempt to reduce the amount of "randomness", i.e. the number of completely random bits used.

Derandomization Problem Of Permutation Branching Program

- Let $n \ll N$. A **pseudorandom generator** accepts a bitstring of length n generated by the n truly random bits, called the **seed**, and yields, in polynomial time, a bitstring of length N, which must be indistinguishable from N truly random bits for any polynomial time algorithm.



Derandomization Problem Of Permutation Branching Program

- Unfortunately, they're only known to exist under the assumption that one-way functions exist, which is even stronger assumption than $P \neq NP$.

Derandomization Problem Of Permutation Branching Program

- Unfortunately, they're only known to exist under the assumption that one-way functions exist, which is even stronger assumption than $P \neq NP$.
- Therefore we restrict our interest into some subclass of algorithms, such as space-bounded ones.

Derandomization Problem Of Permutation Branching Program

- Unfortunately, they're only known to exist under the assumption that one-way functions exist, which is even stronger assumption than $P \neq NP$.
- Therefore we restrict our interest into some subclass of algorithms, such as space-bounded ones.
- As an example, it is known that we only need $O(\log^2 N)$ truly random bits to construct a pseudorandom generator producing $O(poly(N))$ bits in $SPACE(\log(N))$

Derandomization Problem Of Permutation Branching Program

- Unfortunately, they're only known to exist under the assumption that one-way functions exist, which is even stronger assumption than $P \neq NP$.
- Therefore we restrict our interest into some subclass of algorithms, such as space-bounded ones.
- As an example, it is known that we only need $O(\log^2 N)$ truly random bits to construct a pseudorandom generator producing $O(poly(N))$ bits in $SPACE(\log(N))$
- Constructing a pseudorandom generator for a class of algorithms is called the **derandomization problem**.

Derandomization Problem Of Permutation Branching Program

DEFINITION

A **branching program** with n variables $x_1, \cdots, x_n$ is a directed acyclic multigraph where

1. one of the node is marked as an **input node**,
2. outdegree of each nodes is either 2, called an **internal node**, or 0, called a **terminal**,
3. each outward edges of an internal node are labelled 0 and 1 respectively,
4. each internal nodes are labelled with one of the x_i, and
5. each terminals are marked as **accepting** or **rejecting**.

Derandomization Problem Of Permutation Branching Program

DEFINITION

A **branching program** with n variables $x_1, \cdots, x_n$ is a directed acyclic multigraph where

1. one of the node is marked as an **input node**,
2. outdegree of each nodes is either 2, called an **internal node**, or 0, called a **terminal**,
3. each outward edges of an internal node are labelled 0 and 1 respectively,
4. each internal nodes are labelled with one of the x_i, and
5. each terminals are marked as **accepting** or **rejecting**.

A branching program **accepts** the bitstring $S \in \{0,1\}^n$ if the terminal reachable by following the edge labelled S_i for each internal nodes with label x_i is accepting. Otherwise, it **rejects** the bitstring.

Derandomization Problem Of Permutation Branching Program

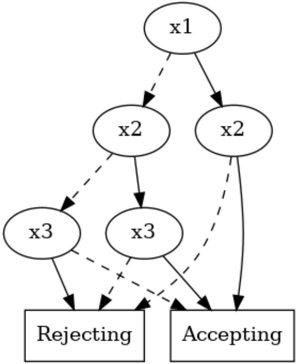


Figure: Branching program for $f(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$

Derandomization Problem Of Permutation Branching Program

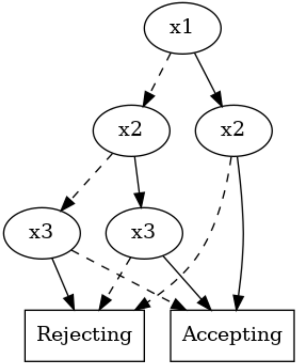


Figure: Branching program for $f(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$

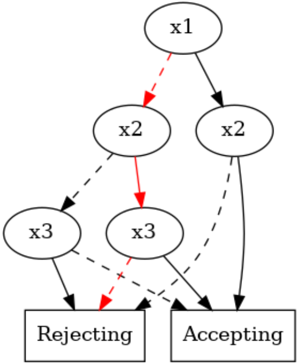


Figure: Rejects 010

Derandomization Problem Of Permutation Branching Program

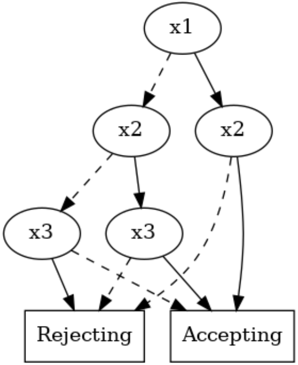


Figure: Branching program for $f(x_1, x_2, x_3) = (\neg x_1 \wedge \neg x_2 \wedge \neg x_3) \vee (x_1 \wedge x_2) \vee (x_2 \wedge x_3)$

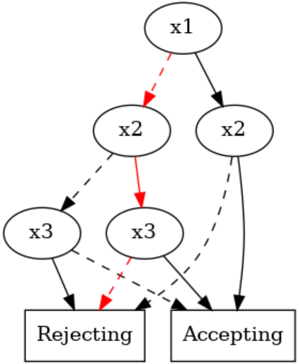


Figure: Rejects 010

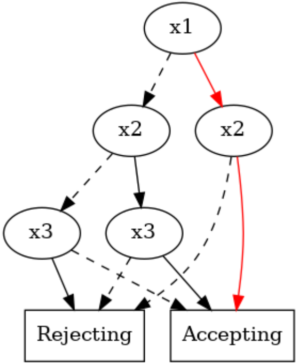


Figure: Accepts 110

Derandomization Problem Of Permutation Branching Program

- It is known that it's suffice to find the construction of a pseudorandom generator for polynomial size read-once branching programs in order to solve the derandomization problem for space bounded computations.

Derandomization Problem Of Permutation Branching Program

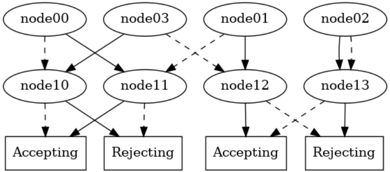
- It is known that it's suffice to find the construction of a pseudorandom generator for polynomial size read-once branching programs in order to solve the derandomization problem for space bounded computations.
- This paper proves that the **Impagliazzo-Nisan-Wigderson generator**(INW generator) solves this problem for a subclass of branching programs, called **permutation branching programs**.

Derandomization Problem Of Permutation Branching Program

DEFINITION

A **permutation branching program** with n variables $x_1, \cdots, x_n$ of width k is a branching program such that

1. nodes are divided into levels $0, 1, \cdots, n$, each containing k vertices numbered $1, 2, \cdots, k$, such that level $0 \leq i < n$ nodes are labelled x_{i+1},
2. one of the level 0 node is marked as an input vertex and level n nodes are terminals, and
3. edges are divided into levels $0, 1, \cdots, n-1$ such that level i edges of label $b \in \{0, 1\}$ forms a permutation from level i nodes to level $i+1$ nodes.



Pseudorandom Generators For Group Products

- Without the loss of generality, we may renumber each nodes so that each edges labelled 0 maps a node numbered i to a node numbered i (on the subsequent level).

Pseudorandom Generators For Group Products

- Without the loss of generality, we may renumber each nodes so that each edges labelled 0 maps a node numbered i to a node numbered i (on the subsequent level).
- Consider the problem of finding a pseudorandom generator such that for every fixed permutation $\pi \in S_k$ and every $1 \leq i \leq k$, it approximates the probability that a random input with input node i reaches the terminal $\pi(i)$ for a fixed read-once permutation branching program.

Pseudorandom Generators For Group Products

- Without the loss of generality, we may renumber each nodes so that each edges labelled 0 maps a node numbered i to a node numbered i (on the subsequent level).
- Consider the problem of finding a pseudorandom generator such that for every fixed permutation $\pi \in S_k$ and every $1 \leq i \leq k$, it approximates the probability that a random input with input node i reaches the terminal $\pi(i)$ for a fixed read-once permutation branching program.
- Note that each permutation given in each level by edges labelled 1 are just an element of S_k.

Pseudorandom Generators For Group Products

- Without the loss of generality, we may renumber each nodes so that each edges labelled 0 maps a node numbered i to a node numbered i (on the subsequent level).
- Consider the problem of finding a pseudorandom generator such that for every fixed permutation $\pi \in S_k$ and every $1 \leq i \leq k$, it approximates the probability that a random input with input node i reaches the terminal $\pi(i)$ for a fixed read-once permutation branching program.
- Note that each permutation given in each level by edges labelled 1 are just an element of S_k.
- Now we may attempt to restate the problem using finite groups.

Pseudorandom Generators For Group Products

- Let G be a finite group.

Pseudorandom Generators For Group Products

- Let G be a finite group.
- A string $w = (g_1, \cdots, g_n)$ of elements of G is called a **group word**.

Pseudorandom Generators For Group Products

- Let G be a finite group.
- A string $w = (g_1, \cdots, g_n)$ of elements of G is called a **group word**.
- Given a group word $w = (g_1, \cdots, g_n)$, we define the probability distribution Rnd^w on G as

$$\mathrm{Rnd}^w(g) = \frac{1}{2^n} |\{(x_1, \cdots, x_n) \in \{0,1\}^n : g = g_1^{x_1} \cdots g_n^{x_n}\}|$$

.

Pseudorandom Generators For Group Products

- Let G be a finite group.
- A string $w = (g_1, \cdots, g_n)$ of elements of G is called a **group word**.
- Given a group word $w = (g_1, \cdots, g_n)$, we define the probability distribution Rnd^w on G as
$$\mathrm{Rnd}^w(g) = \frac{1}{2^n}|\{(x_1, \cdots, x_n) \in \{0,1\}^n : g = g_1^{x_1} \cdots g_n^{x_n}\}|$$
.
- It's not hard to see that the derandomization problem for Rnd^w is equivalent to that of the permutation branching program.

INW Generator

DEFINITION

An (N, M, λ)-**expander** is an undirected M-regular multigraph on N vertices whose second largest absolute value of eigenvalues of its normalized adjacency matrix is at most λ.

INW Generator

DEFINITION

An (N, M, λ)-**expander** is an undirected M-regular multigraph on N vertices whose second largest absolute value of eigenvalues of its normalized adjacency matrix is at most λ.

Recall that an expander graph has property that each "small" subset of vertices has "large" boundary.

INW Generator

- Given a 2^d-regular multigraph, we may label (u, e), for each vertex u and an edge e incident to u so that the labels of (u, e) forms a permutation of $\{0, 1\}^d$ for u.

INW Generator

- Given a 2^d-regular multigraph, we may label (u, e), for each vertex u and an edge e incident to u so that the labels of (u, e) forms a permutation of $\{0, 1\}^d$ for u.
- For $y \in \{0, 1\}^r$ and $y' \in \{0, 1\}^d$, let $\nu(y, y')$ be a neighbor of y reachable by the edge e where (y, e) is labelled y'.

INW Generator

- Given a 2^d-regular multigraph, we may label (u, e), for each vertex u and an edge e incident to u so that the labels of (u, e) forms a permutation of $\{0, 1\}^d$ for u.
- For $y \in \{0, 1\}^r$ and $y' \in \{0, 1\}^d$, let $\nu(y, y')$ be a neighbor of y reachable by the edge e where (y, e) is labelled y'.

DEFINITION

Let $\Gamma_1, \Gamma 2 : \{0, 1\}^r \to \{0, 1\}^n$ be functions and F be a 2^d-regular multigraph with vertex set $\{0, 1\}^r$. The **expander product** of Γ_1 and Γ_2 by F is the function $\Gamma_1 \otimes_F \Gamma_2 : \{0, 1\}^{r+d} \to \{0, 1\}^{2n}$ defined by

$$(\Gamma_1 \otimes_F \Gamma_2)(y, y') = (\Gamma_1(y), \Gamma_2(\nu(y, y')))$$

.

INW Generator

- The INW generator is obtained by recursively applying the expander product with a family of expanders of increasing sizes of vertices.

INW Generator

- The INW generator is obtained by recursively applying the expander product with a family of expanders of increasing sizes of vertices.
- In order to construct such family, we'll first look at some basic operations on expander graphs.

INW Generator

DEFINITION

For a D-regular undirected graph G, the **rotation map** $\mathrm{Rot}_G : [N] \times [D] \to [N] \times [D]$ is defined as follows: $\mathrm{Rot}_G(v, i) = (w, j)$ if the i-th edge incident to v leads to w and this edge is the j-th edge incident to w.

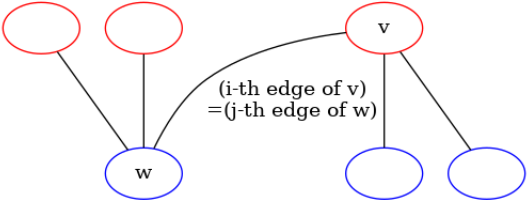


Figure: $\mathrm{Rot}_G(v, i) = (w, j), \mathrm{Rot}_G(w, j) = (v, i)$

INW Generator

First Operation(Power)

Let G be a D-regular multigraph on $[N]$. The t-th power of G is the D^t-regular multigraph G^t whose rotation map is given by $\mathrm{Rot}_{G^t}(v_0, (k_1, \cdots, k_t)) = (v_t, (l_t, \cdots, l_1))$ where these values are computed via the rule $\mathrm{Rot}_G(v_{i-1}, k_i) = (v_i, l_i)$.

INW Generator

First Operation(Power)

Let G be a D-regular multigraph on $[N]$. The t-th power of G is the D^t-regular multigraph G^t whose rotation map is given by $\mathrm{Rot}_{G^t}(v_0,(k_1,\cdots,k_t)) = (v_t,(l_t,\cdots,l_1))$ where these values are computed via the rule $\mathrm{Rot}_G(v_{i-1},k_i) = (v_i,l_i)$.

The t-th power is just the graph whose normalized adjacency matrix is the t-th power of the normalized adjacency matrix of the operand.

INW Generator

Since the t-th power of a matrix has eigenvalues powered by t, the following is immediate:

INW Generator

Since the t-th power of a matrix has eigenvalues powered by t, the following is immediate:

THEOREM

If G is an (N, D, λ)-expander, then G^t is an (N, D^t, λ^t)-expander. Moreover, Rot_{G^t} is computable in time $\mathrm{poly}(\log N, \log D, t)$ with t oracle queries to Rot_G.

INW Generator

Second Operation(Tensor Product)

Let G_1 be a D_1-regular multigraph on $[N_1]$ and G_2 a D_2-regular multigraph on $[N_2]$. The **tensor product** $G_1 \otimes G_2$ is the $D_1 \cdot D_2$-regular multigraph on $[N_1] \times [N_2]$ given by $\mathrm{Rot}_{G_1 \otimes G_2}((v,w),(i,j)) = ((v',w'),(i',j'))$ where $\mathrm{Rot}_{G_1}(v,i) = (v',i')$ and $\mathrm{Rot}_{G_2}(w,j) = (w',j')$.

INW Generator

Second Operation(Tensor Product)

Let G_1 be a D_1-regular multigraph on $[N_1]$ and G_2 a D_2-regular multigraph on $[N_2]$. The **tensor product** $G_1 \otimes G_2$ is the $D_1 \cdot D_2$-regular multigraph on $[N_1] \times [N_2]$ given by $\mathrm{Rot}_{G_1 \otimes G_2}((v, w), (i, j)) = ((v', w'), (i', j'))$ where $\mathrm{Rot}_{G_1}(v, i) = (v', i')$ and $\mathrm{Rot}_{G_2}(w, j) = (w', j')$.

The tensor product is just the graph whose normalized adjacency matrix is the product of the respective normalized adjacency matrices of operands.

INW Generator

Tensor product of matrices has a nice property that its eigenvalues are the multiset of pairwise products of respective eigenvalues of operands. Therefore, the largest eigenvalue $1 \cdot 1 = 1$ and the second largest is $\max(\lambda_1 \cdot 1, 1 \cdot \lambda_2)$. Therefore, the following holds:

INW Generator

Tensor product of matrices has a nice property that its eigenvalues are the multiset of pairwise products of respective eigenvalues of operands. Therefore, the largest eigenvalue $1 \cdot 1 = 1$ and the second largest is $\max(\lambda_1 \cdot 1, 1 \cdot \lambda_2)$. Therefore, the following holds:

THEOREM

If G_1 is an (N_1, D_1, λ_1)-expander and G_2 is an (N_2, D_2, λ_2)-expander, then $G_1 \otimes G_2$ is an $(N_1 \cdot N_2, D_1 \cdot D_2, \max(\lambda_1, \lambda_2))$-expander. Moreover, $\mathrm{Rot}_{G_1 \otimes G_2}$ is computable in time $\mathrm{poly}(\log N_1 N_2, \log D_1 D_2)$ with one oracle queries to each Rot_{G_1} and Rot_{G_2}.

INW Generator

Third Operation(Zig-zag Product)

If G_1 is a D_1-regular graph on $[N]$ and G_2 is a D_2-regular graph on $[D_1]$, then their **zig-zag product** $G_1 \text{ⓩ} G2$ is a D_2^2-regular graph on $[N] \times [D_1]$ whose rotation map is as follows:
$\mathrm{Rot}_{G_1 \text{ⓩ} G_2}((v,k),(i,j))$:

1. Let $(k', i') = \mathrm{Rot}_{G_2}(k, i)$.
2. Let $(w, l') = \mathrm{Rot}_{\mathrm{G}_1}(v, k')$.
3. Let $(l, j') = \mathrm{Rot}_{\mathrm{G}_2}(l', j)$.
4. Output $((w, l), (j', i'))$.

INW Generator

THEOREM

If G_1 is an (N, D_1, λ_1)-expander and G_2 is a (D_1, D_2, λ_2)-expander, then $G_1 ⓩ G_2$ is a $(N_1 D_1, D_2^2, \lambda_1 + \lambda_2 + \lambda_2^2)$-expander. Moreover, $\mathrm{Rot}_{G_1 ⓩ G_2}$ can be computed in time $\mathrm{poly}(\log N, \log D_1, \log D_2)$ with one oracle query to Rot_{G_1} and two oracle queries to Rot_{G_2}.

INW Generator

Let H be a (D^8, D, λ)-expander for some D and λ. For $t \geq 1$, we define a (D^{8t}, D^2, λ_t)-expander G_t as follows:

1. $G_1 = H^2$.
2. $G_2 = H \otimes H$.
3. For $t \geq 3$,
$$G_t = \left(G_{\lceil \frac{t-1}{2} \rceil} \otimes G_{\lceil \frac{t-1}{2} \rceil}\right)^2 \text{Ⓩ} H$$
.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

- The number of vertices and degree can be shown with a straightforward induction.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

- The number of vertices and degree can be shown with a straightforward induction.
- To analyze the eigenvalue, let $\mu_t = \max\{\lambda_1, \cdots, \lambda_t\}$.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

- The number of vertices and degree can be shown with a straightforward induction.
- To analyze the eigenvalue, let $\mu_t = \max\{\lambda_1, \cdots, \lambda_t\}$.
- Then we have $\mu_t \leq \max\{\mu_{t-1}, \mu_{t-1}^2 + \lambda + \lambda^2\}$ for all $t \geq 2$.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

- The number of vertices and degree can be shown with a straightforward induction.
- To analyze the eigenvalue, let $\mu_t = \max\{\lambda_1, \cdots, \lambda_t\}$.
- Then we have $\mu_t \leq \max\{\mu_{t-1}, \mu_{t-1}^2 + \lambda + \lambda^2\}$ for all $t \geq 2$.
- Solving this recurrence gives $\mu_t \leq \lambda + O(\lambda^2)$ for all t.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

- The number of vertices and degree can be shown with a straightforward induction.
- To analyze the eigenvalue, let $\mu_t = \max\{\lambda_1, \cdots, \lambda_t\}$.
- Then we have $\mu_t \leq \max\{\mu_{t-1}, \mu_{t-1}^2 + \lambda + \lambda^2\}$ for all $t \geq 2$.
- Solving this recurrence gives $\mu_t \leq \lambda + O(\lambda^2)$ for all t.
- For the time complexity, note that the depth of the recursion is $\log t$ and evaluation of Rot_{G_t} requires 4 evaluations of rotation maps for smaller graphs.

INW Generator

THEOREM

For every $t \geq 1$, G_t is an (D^{8t}, D^2, λ_t)-expander with $\lambda_t \in \lambda + O(\lambda^2)$. Moreover, Rot_{G_t} can be computed in time $\mathrm{poly}(t, \log D)$ with $\mathrm{poly}(t)$ oracle queries to Rot_H.

PROOF)

- The number of vertices and degree can be shown with a straightforward induction.
- To analyze the eigenvalue, let $\mu_t = \max\{\lambda_1, \cdots, \lambda_t\}$.
- Then we have $\mu_t \leq \max\{\mu_{t-1}, \mu_{t-1}^2 + \lambda + \lambda^2\}$ for all $t \geq 2$.
- Solving this recurrence gives $\mu_t \leq \lambda + O(\lambda^2)$ for all t.
- For the time complexity, note that the depth of the recursion is $\log t$ and evaluation of Rot_{G_t} requires 4 evaluations of rotation maps for smaller graphs.
- Therefore, total number of recursive calls is at most $4^{\log t} = t^2$.

INW Generator

Corollary

There is a universal constant $c_0 > 0$ such that for every constant $0 < \lambda < 1$ and $d = c_0 \lceil \log 1/\lambda \rceil$, there exists a sequence F_m of $(2^{d \cdot m}, 2^d, \lambda)$-expanders, where neighbors in F_m are computable in $O(d \cdot m)$ space and $\mathrm{poly}(d \cdot m)$ time.

INW Generator

We're now ready to present the construction of INW generator.

- For $0 < \lambda < 1$ and an integer $n \geq 1$, (λ, n)-INW generator is obtained recursively as follows.
- Let $\Gamma_0 : \{0,1\}^d \to \{0,1\}^d$ be the identity mapping.
- Then $\Gamma_{i+1} = \Gamma_i \otimes_{F_i} \Gamma_i$ where F_i is the $(2^{d(i+1)}, 2^d, \lambda)$-expander from the previous corollary.
- This gives (λ, n)-INW generator for every $n = d2^k$ where $k > 0$.
- We obtain the generator for arbitrary n by taking first n bit from (λ, n')-INW generator where $n' = d2^k \geq n$ is taken to be the smallest.
- Hence, (λ, n)-INW generator giving n bits of output has seed length $O(\log n \cdot \log 1/\lambda)$.

INW Generator

Recall that we're trying to approximate the distribution Rnd^w.

INW Generator

Recall that we're trying to approximate the distribution Rnd^w.
With the INW generator Γ, we have the distribution

$$D_\Gamma^w(g) = \frac{1}{2^r}|\{y \in \{0,1\}^r | g = g_1^{\Gamma(y)_1} \cdots g_n^{\Gamma(y)_n}\}|$$

.

INW Generator

Recall that we're trying to approximate the distribution Rnd^w.
With the INW generator Γ, we have the distribution

$$D_\Gamma^w(g) = \frac{1}{2^r}|\{y \in \{0,1\}^r | g = g_1^{\Gamma(y)_1} \cdots g_n^{\Gamma(y)_n}\}|$$

.

THEOREM

Let G be any finite group of size at least 4 and $0 < \delta < 1$. Let $\lambda = \delta/(2^{c_1|G|^{12}} \cdot \sqrt{|G|})$ where c_1 is the universal constant from the corollary. Then (λ, n)-INW generator Γ uses seeds of length $O(\log n \cdot (|G|^{12} + \log 1/\delta))$ to product n bits such that for every group word w of length n,

$$\|\mathrm{Rnd}^w - D_\Gamma^w\| \leq \delta$$

. Moreover, the output of the generator is computable in space linear in the seed length.

Conclusion

- The author mentions that he didn't try to optimize the constant factor involved in the construction.
- The general derandomization problem for branching programs remains open.

The End