Report submitted for the course
Complexity Theory II

by
Saswata Mukherjee & Somnath Bhattacharjee

on

$BP_H SPACE(s) \subseteq DSPACE(s^{3/2})$

by

Michael Saks & Shiyu Zhou

January 8, 2023

Contents

1 Introduction

Here we give the brief idea of the paper, which proves that any randomized algorithm with bounded two sided error protocol which requires $O(s)$ space, can be deramdomized in $O(s^{3/2})$ space. It improves the best known previous result which shows that $BP_H SPACE(s) \subseteq DSPACE(s^2)$.

2 Key Idea

At first let us define a problem.

Problem 2.1. [Approximate Substochastic Matrix Repeated Squaring (AMRS)]:

Input: $M \leftarrow d \times d$ substochastic matrix, $2^r, 2^a \leftarrow$ in unary.
Output: $d \times d$ substochastic matrix M' such that $||M^{2^r} - M'|| \leq 2^{-a}$.

(We call such M' to be $a-$approximation of M.)

We will mainly try to give randomized algorithm for this problem and then will de-randomize it.

2.1 Why AMRS:

Now we give idea how our main goal can be reduced to the above problem 2.1.

If a TM uses $s(n)$ space on input of size n, then we know that the number of configurations of the TM will be $d = 2^{O(s(n))}$, i.e., it uses d many steps. And also there is one **Accept** and one **Start** configuration. Therefore, we can associate a markov matrix $M_{d\times d}$ with this situation, in the following way:
$M(i,j) =$ probability of going from ith to jth configuration in one step.
$\implies M^d(i,j) =$ probability of going from ith to jth configuration in d-many steps.

Therefore, if (**Start, Accept**) entry of $a-$approximation matrix of M^d is "sufficiently good", we can say the input is in the language, and reject otherwise.

So, to solve our problem it is enough to give a deterministic algorithm for 2.1 (with paremeters $r = a = \lceil \log d \rceil$), which uses $O(\log^{3/2} d)$ space.

2.2 Enough to Find a Good Randomized Algorithm for AMRS

In this section we will see finding a *good enough* randomized algorithm for AMRS is sufficient to derandomized it. But before that we need to introduce the concept of *off-line* randomness. Then we can decide what is good what is not.

A randomized algorithm A on any input x will be called off-linr randomized if it will take some random string $y \in \{0,1\}^{R(x)}$ and after getting y the procedure is completely deterministic ,ie, after fixing x and y the algorithm $A(x,y)$ will note produce any random string. We will say the *random bit complexity* of A is $|y| = R(x)$ and the space required to run $A(x,y)$ once y is written down is *processing space complexity*. The over all space is sum of these two.

Let F be a substocastic matrix function. In this case can think F to be the actual function for AMRS ,ie, on input M,z (where M is the input matrix and z denotes r and a) F computes M^{2^r}. We will say an off-line randomized algorithm A approximates F with *accuracy* $a \in \mathbb{Z}$ and error probability $\beta \in [0,1]$ if there is polynomially bounded function R

$$\Pr_{y \xleftarrow{u.a.r.} \{0,1\}^{R(M,z)}} [||A(M,z,y) - F(M,z)||\ 2^{-a}] \leq \beta$$

Now for such offline randomized A approximating F, the *naive derandomization* of A will be compute $A(M,z,y)$ for each $y \in \{0,1\}^{R(M,z)}$ and then take the arithmetic average of all the outputs. If the random bit complexity and processing space complexity of A is R and S respectively. Note for each y, the processing space complexity can be reused and and the space nedded to compute the average and store $(A(M,z,y)$ is $O(S+R)$ (for any iteration we can store the average value for all the previous iterations and in the end of the iteration we can use the corresponding output to modify the current average). So overall space complexity will be $O(S+R)$ for the deterministic algorithm. The following lemma will prove the accuracy will not change that much.

Lemma 1. If B is the naive derandomisation of A and if A has accuracy a error probability $\beta \leq 2^{-(a+1)}$, the B has accuracy $a-1$

Hence finding an offline randomized algorithm for AMRS with small random bit complexity and processing space complexity and exponentially small error probability is sufficient.

3 Towards Construction and Analysis of the Main Algorithm

3.1 Recursive Use of PRS to Get Main Algorithm

Let us describe what the PRS algorithm does at first.

Problem 3.1.
Input: Integer m,r, substochastic matrix $M_{d\times d}$, offline random input $h \in \{0,1\}^{2mr}$.
Output: For an integer a it outputs $a-$approximation of $\Lambda^r(M) := M^{2^r}$, which is M', with error probability $2^{2a+3r+5\log d}/2^m$.
Space required $O(m+r+\log d)$

We shall see in section 4 that there is a PRS algorithm for 3.1.

The parameter m defined above will be called the *randomization parameter*. In the end we will set $m = O(s)$ $(s = \max\{r,a,\log d\})$to get exponentially small error so the naive derandomization will have space complexity $\Theta(rs)$. So to improve that the next idea will be apply PRS recursively. One can note if $r = r_1 \times r_2$ then

$$\Lambda^{(r)}(M) = \left(\Lambda^{(r_1)}\right)^{r_2}(M) \qquad \text{(ie, applying } \Lambda^{(r_1)}\ r_2 \text{ times on } M\text{)}$$

Now using that idea if we compute $\Lambda^{(r_1)}$ recursively r_2 times we will have total r_2 many iterations and each iteration will take $2mr_1$ length random string, so total random bit complexity will be $2r_1r_2s = rs$ and processing space complexity will be $O(r_2s)$. The random bit complexity increases because we are using different random strings in each iterations. So what if we use the same random string in each iteration? We will say a random string h is good for some randomised algorithm A if using h A gives the desired answer. Now clearly we don't know whether almost all random strings h are good for each iteration. We will discuss this case in the next section.

3.2 Use of "Perturbing" to get one single good random string

Let $M_0 = M, M_1, \ldots, M_{r_2}$ be the sequence of matricies we compute in the algorithm defined in section 3.1. And $N_0 = M, N_1, \ldots, N_{r_2}$ where $N_i = N_0^{2^{r_1}}$ be the sequence of matrices which ideally should be computed. One can view the problem in terms of approximating N_i to M_i.

We already said that we don't know whether almost all hs are good for all M_is or not. But note almost all hs are good for all N_is if we apply PRS to N_i. Now using this observation the paper gave a reasonable conjecture that if N_i and M_i are close and h is good for N_i then $PRS(N_i)$ and $PRS(M_i) = M_{i+1}$ is also close, hence applying induction one can prove almost all hs are good for all M_i. But even if the conjecture is true the error growth is too high to handle.

To tackle this obstacle we will perturb and then truncate each M_i entries and then will apply PRS. the main idea behind this is by truncating low order bits of the entries, try to make N_i and modified M_i close enough. Now note roughly if the modified M_i and N_i are close enough then we can say almost all h are good for all M_i.

3.3 Main Algorithm

Before describing the main algorithm we have to define some notations.

Definition 3.1. For some non-negative real number δ the perturbation operator is $\Sigma_\delta : [0,1] \to [0,1]$, $\Sigma_\delta(z) = \max\{z - \delta, 0\}$
for some non negative integer t, the truncation operator is $\lfloor . \rfloor_z : [0,1] \to [0,1]$, $\lfloor z \rfloor_t = 2^{-t}\lfloor 2^t z \rfloor$
Applying these two operator on a matrix means applying these operators in each entries of the matrix.

We will call the algorithm MAIN which is just recursive application of PRS in a clever manner. We will take $r_1 = r_2 = \sqrt{r}$, if r is not a perfect square then we will compute for the largest perfect square smaler than r and from that we will compute manually.

MAIN will have additional parameters m, t, D, K of $\Theta(s)$ computed from the input and will not change through out the process. m is PRS randomization parameter and $t + D = K$.

MAIN will have offline random string $h \in \{0,1\}^{2mr_1}$ and $q =\in \{0,1\}^{Dr_2}$ where $q =$

$[q(1),\dots,q(r_2)]$ and each $q(i) \in \{0,1\}^D$. Let $\delta_i = q(i)2^{-K}$. MAIN will basically compute

$$M = M_0, M_1^P, M_1^\Sigma, M_1, M_2^P, M_2^\Sigma, M_2, \dots, M_{r_2}^P, M_{r_2}^\Sigma, M_{r_2}$$

Basically in each step it is computing M_i^P, M_i^Σ, M_i from M_{i-1} and each entries will have K bits, where

$$\begin{aligned} M_i^P &= PRS(M_{i-1}, r_1, m, h) && \text{(applying PRS on } M_{i-1}\text{)} \\ M_i^\Sigma &= \Sigma_{\delta_i}(M_i^P) && \text{(Perturb last } D \text{ bits of each entry randomly)} \\ M_i &= \lfloor M_i^\Sigma \rfloor && \text{(Truncate the entries upto first } t \text{ bits)} \end{aligned}$$

In the end it will return M_{r_2}.

Note the random bit complexity will be $2r_1 m + r_2 D = O(r^{\frac{1}{2}} s)$ and processing complexity will be $O(r_2 s)$ since it is computing $3r_2$ many matrices and each matrix can be computed in $O(s)$ time from the previous matrix, so it will compute entry by entry (ie, to compute some entry (i,j) it will compute the entries of the previous matrix recursively). Hence the naive deranomization will give $O(r^{\frac{1}{2}} s)$ deterministic algorithm.

3.4 Glimps of Correctness

Let us modify the N_i sequence in similar way:
Define $M = N_0, N_1^P, N_1^\Sigma, N_1, N_2^P, N_2^\Sigma, N_2, \dots, N_{r_2}^P, N_{r_2}^\Sigma, N_{r_2}$ where

$$\begin{aligned} N_i^P &= N_{i-1}^{2^{r_1}} && \text{(ARMS on } N_{i-1}\text{)} \\ N_i^\Sigma &= \Sigma_{\delta_i}(N_i^P) && \text{(Perturb last } D \text{ bits of each entry randomly)} \\ N_i &= \lfloor N_i^\Sigma \rfloor && \text{(Truncate the entries upto first } t \text{ bits)} \end{aligned}$$

Now the following lemma will say that N_i and M_i sequences are close enough

Lemma 2. For any choice of $q \in \{0,1\}^{Dr_2}$ N_r approximates $\Lambda^{(r)}(M)$ with accuracy $K - D - 2r - \log d$.

Now note for a fixed q, N_i does not depends on h. We will say a q is good if for almost h, $N_i = M_i$. And then we can prove for any q, almost all h are good for N_i, and a large fraction of qs are good. Hence we can say $N_i = M_i$ with high probability. Note error can occur in two ways, either q is bad or for good q, h is bad. Now with calculation we can prove the sum is very small.

4 Matrix Pseudorandom Repeated Squaring: PRS

In previous section, we have defined PRS, and in this section we are going to show a brief proof idea and analysis of the algorithm vi a Nisan's generator.

Take substochastic matrix $M_{d\times d}$ = Normalized adjacency matrix of graph G.
Where G has d vertices and it is 2^m−regular graph and for some $i \in V(G)$, all edges from i

is enumerated by elements of $\{0,1\}^m$. We can interpret this as,

$$M[i,j] = \text{Probability of going from } i \text{ to } j\text{th configuration using } m \text{ random bits.}$$

So, $M^p[i,j] =$ Probability of going from i to jth configuration using mp random bits.

Similarly define M' ,$M'[i,j] = \{a \in \{0,1\}^m :$ there is an edge a from i to $j\}$.
$(M')^p[i,j] =$ Set of all strings in $(\{0,1\}^m)^p$ so that it defines a $i \to j$ path of length p in G.

For any such M', $(M')^*[i,j] = |M'[i,j]|2^{-m}$, and for any M, we can construct $Q(M) = M'$ so that $(Q(M))^* = M$.

We are going to construct an explicit PRG $G : \{0,1\}^t \to \{0,1\}^n$ where t n with a very small error. More precisely, $t = O(rm) = O(rs)$ as, we know $m = O(s)$ and we want to find approximate M^{2^r}. And $n \approx O(s2^r)$. Instead of using large number of true random bits, if we use pseudorandom bit, it will not cost us so much and moreover by 2.2, using small amount of space we can reach our goal.

4.1 Some More Definitions

Take a map $g : \{0,1\}^m \to (\{0,1\}^m)^p$, call it $(m,p)-$generator.
Define $(M'_g)_{d\times d}$ so that $x \in M'_g[i,j] \iff g(x)$ defines a path of length p from i to j.
Say, such g is $\epsilon-$PR w.r.t $M' \iff ||((M')^p)^* - (M'_g)^*|| \leq \epsilon$.

Composition of PRGs:

If g is a $(m,p)-$generator and g' is a $(m,p')-$generator.
then define $g \circ g'(x) = g(g'(x))_1, \ldots, g(g'(x))_{p'}$. Clearly, $g \circ g'$ is a $(m,pp')-$generator.
And it can be easily proved that $M'_{g\circ g'} = (M'_g)_{g'}$.

Lemma 3.
If g is $\epsilon-$PR and g' is $\epsilon'-$PR w.r.t. M', then $g \circ g'$ is $(\epsilon' + p'\epsilon)-$PR w.r.t M'.

4.2 Family of Pairwise Independent Hash Functions Helps

Definition 4.1.
$\mathcal{H}_k = \{f : f\{0,1\}^k \to \{0,1\}^k\}$ be a pairwise independent family of hash functions, if $\forall x_1 \neq x_2, y_1, y_2 \in \{0,1\}^k$,

$$Pr_{h\sim\mathcal{H}_k}[h(x_1) = y_1 \wedge h(x_2) = y_2] = 2^{-2k}$$

We have proved expander mixing lemma for Expander graphs. We can prove that kind of lemma for pairwise independent hash family.

Take $A, B \subseteq \{0,1\}^m$, then, we say $h \in \mathcal{H}$ is $\epsilon-$good if

$$|Pr_x[x \in A \wedge h(x) \in B] - \rho(A)\rho(B)| \leq \epsilon.$$

Where $\rho(.)$ is the density of the set.

Lemma 4.
For any A, B, $Pr_{h \sim \mathcal{H}_m}[h \text{ is not } \epsilon - good] \leq \frac{1}{\epsilon^2 2^m}$.

M' is as defined before, then, we say $h \in \mathcal{H}_m$ is (M', ϵ) good if,

$$\forall i, j, k \in [d],\ h \text{ is } \epsilon - \text{good w.r.t. } (A = Q[i,j], B = Q[j,k])$$

Lemma 5.
$Pr_{h \sim \mathcal{H}_m}[h \text{ is not } (M', \epsilon) \text{ good}] \leq \frac{d}{\epsilon^2 2^m}$.

4.3 Construction of Generator via Composition of Hash Functions

Pick $h_1 \leftarrow \mathcal{H}_m$ uniformly at random, and define $G_1(x) = (x, h_1(x))$ for $x \in \{0,1\}^m$.

Lemma 6.
If h_1 is $(M' = Q(M), \epsilon)-$good, then the $(m, 2)$ generator G_1 is ϵd^2-PR w.r.t. M'.

Proof:
. $M'_{G_1}[i,k] = \{x : \exists j \text{ such that } x \in M'[i,j], h_1(x) \in M'[j,k]\}$.
Now, $|((M')^2)^*[i,k] - (M'_{G_1})^*[i,k]|$
$= |\sum_j \rho(M'[i,j])\rho(M'[j,k]) - Pr_x[x \in M'[i,j \wedge h_1(x) \in M'[j,k]$
$\leq \sum_j |\rho(M'[i,j])\rho(M'[j,k]) - Pr_x[x \in M'[i,j \wedge h_1(x) \in M'[j,k]|$ [h_1 is $(M', \epsilon)-$good.]
$\leq \epsilon d.$ ■

Now, to construct our generator we randomly sample $h_1, \ldots, h_r \leftarrow \{\mathcal{H}_m\}$.
Take $G_r = G_1 \circ G_2 \circ \cdots \circ G_r$, it will be $(m, 2^r)-$generator.
Define $g_1 \circ \cdots \circ g_r$ to be ϵ well behaved if for all i, g_i is $M'_{g_1 \circ \ldots g_{i-1}}, \epsilon$ good.

- We can prove that if $h_1, \ldots, h_r \in \mathcal{H}_m$ is picked randomly, then, with probability $\frac{rd}{\epsilon^2 2^m}$, G_r is not ϵ well behaved.
- Also, by induction, if G_r is $(M', \epsilon)-$good then, it is $(2^r \epsilon d^2)-$ PR.

So, by randomly picking $h_1, \ldots, h_r$, with prob $\geq 1 - \frac{rd^5 2^{2r}}{\epsilon^2 2^m}$, G_r is $\epsilon-$PR, w.r.t. M', M.

PRS Algo:
```
Randomly sample $(h_1, \ldots, h_r) \in \{0,1\}^{O(mr)}$.
c=0
for all $\alpha \in \{0,1\}^M$:
    if $G_r(\alpha)$ defines a path from $i$ to $j$:
        c += 1.
    end if.
end for.
Output $M[i,j] = c2^{-m}$.
```

Analysis:
From 4.3, to achieve accuracy 2^{-a}, we have to allow error probability $2^{2a+3r+5\log d}/2^{m}$.
And for space analysis, observe that it uses $O(mr)$ random bits and using $\approx O(m+r+\log d)$ space. So, combining these, we are done.