CS 860: Space Bounded Computation **University of Waterloo: Winter 2026**

Lecture 6: Time-Space Tradeoffs on Branching Programs I

February 24, 2026

Instructor: Sepehr Assadi *Scribe: Max Jiang*

Topics of this Lecture

In this lecture, we explore time-space tradeoffs for branching programs with *large* outputs. In particular, we review Beame's lower bounds for the unique elements and sorting problems[Bea91], and then adapt this framework to provide an alternate proof for the matrix multiplication lower bound of Abrahamson [Abr86].

1 The Unique Elements Problem

In previous lectures, we primarily explored the limits of computation when we solely focused on space complexity. In this lecture, our focus will be on time-space tradeoffs: when *both* space and time are limited simultaneously (we already saw an algorithmic question of this nature for STCONN in Lecture 4). We want to ask what can and cannot be solved, on inputs of length n, when space S and time T satisfy $S \cdot T = O(f(n))$ for a given function $f : \mathbb{N} \to \mathbb{N}$. For now, we are interested in problems where the output is large, i.e., the output size is $\Omega(n)$. Time-space tradeoffs for decision problems will be discussed in future lectures.

Our first problem of interest will be the unique elements problem, which is defined below.

Problem 1 (Unique Elements). In the unique elements problem, we have as input a list of n numbers $x_1, x_2, \ldots, x_n$ in $[n]$. The goal is to output the indices of all unique elements in the input, i.e., output all $i \in [n]$ such that $x_i \neq x_j$ for all $j \neq i$.

There are two contrasting ways to solve this problem in a standard RAM model (with word size $O(\log n)$).

- In $T = \Theta(n^2)$ time and $S = \Theta(1)$ space: For each index $i \in [n]$, we can store x_i and scan the input to check if there exists another index $j \neq i$ such that $x_i = x_j$. If not, we output i as a unique element. We do not need more than a constant number of registers of size $O(\log n)$ to implement this algorithm, but will spend $\Theta(n)$ time per index (in the worst-case) and thus the overall runtime is $\Theta(n^2)$.
- In $T = \Theta(n)$ time and $S = \Theta(n)$ space: Keep one bit vector of length n to mark which values have appeared in the input, and one for which values have appeared more than once. We scan the input and update these bit vectors accordingly. At the end, we can output the indices that have not appeared more

than once. This requires storing $O(n)$ bits for the bit-vectors and making a constant number of passes over the input in total, so $O(n)$ time as well.

There are also "middle grounds" between these two algorithms. As a rough sketch, we can use bit vectors of length s and make $O(n/s)$ passes over the input, focusing on the elements in the range $((i-1)\cdot s, i\cdot s]$ on the i-th pass. This allows us to solve the problem in $T = \Theta(n^2/s)$ time and $S = \Theta(s)$ space.

In all these algorithms, the product of time and space is $\Omega(n^2)$. A natural question arises: is it possible to solve the problem with $S \cdot T = o(n^2)$? It turns out the answer is *No*, as proven in [Bea91].

Theorem 1 ([Bea91])**.** *Any algorithm with S in $\Omega(\log n)$ that solves unique elements has $S \cdot T = \Omega(n^2)$.*

Theorem 1 is proven more generally in the non-uniform model of branching programs with outputs that we shall review below (and already defined in Lecture 1). This model is powerful enough to simulate a RAM machine algorithm with the same time and space parameter (up to constant factors), so this lower bound also applies to RAM machines. We defined the decision version of branching programs in Lecture 1. Here, we extend the definition to multi-output problems.

Definition 2. For parameters $w, d, n \in \mathbb{N}$ and a function $f : [m]^n \to [o]$, a **branching program with output** is a layered directed acyclic graph (DAG) with d layers with the following properties:

- There are $d+1$ layers of vertices, with exactly w vertices in each layer. Each vertex except for the last layer has exactly m outgoing edges labelled from 1 to m.
- Any edge must go from a vertex in layer i to a vertex in layer $i+1$ for $i \in [d]$.
- The first layer has a single vertex called the **root**.
- Every vertex v except for the last layer is labeled by an integer $i_v \in [n]$, and each edge e has a value $out_e \in [o] \cup \{\perp\}$.

The computation of on an input $x = (x_1, x_2, \ldots, x_n) \in [m]^n$ proceeds as follows, beginning from the root vertex. At vertex v, we read the i_v-th entry of the input. We take the edge e labelled x_{i_v} out of v to the next layer, appending out_e to the output if it is not $\perp$. Once we reach the last layer, we halt.

The parameter d is called the **depth** and w the **width** of the branching program. The space usage of the computation is $S = \log w$ and the time usage is $T = d$.

From here on, this is the computation model we work with, and for brevity we may refer to branching programs with output simply as just branching programs (or even BPs). We will use the same model throughout; only the input/output alphabets change by problem.

1.1 Unique Elements Lower Bound

The proof of Theorem 1 is roughly broken down into four steps, which we can summarize as follows:

1. Define a set of inputs that are considered *"hard"*. In this case, inputs that have $\Omega(n)$ output elements. Show that a large proportion of inputs are hard.
2. Show that on a hard input, if we break down the BP computation into b sub-computations of low depth BPs, one of these sub-computations must output $\Omega(n/b)$ elements.
3. Show that on a random hard input, the probability that a low depth sub-program completes step 2 is exponentially small.

4. Union bound over *all* sub-programs of the original BP to argue that on some hard inputs, the entire BP cannot output enough elements correctly.

This is a general framework for proving time-space tradeoffs for any problem with large output size, and we will follow it below to prove Theorem 1. For the rest of this proof, fix a BP $\mathcal{B}$ of space S and depth T for the unique elements problem.

Step 1. Let U^n denote the uniform distribution over $[n]^n$. The following claim allows us to define our "hard" inputs. For any $x \in [n]^n$, define unique (x) as the number of unique elements in x.

Claim 3. *For input $x = (x_1, x_2, \ldots, x_n)$ drawn from U^n,*

$$\Pr_{x \sim U^n}\left(\text{unique}\,(x) \geqslant \frac{n}{2e}\right) \geqslant \frac{1}{10}.$$

Proof. Let $Y_i = 1$ if x_i is unique and 0 otherwise, and let $Y = \sum_{i=1}^{n} Y_i = \text{unique}\,(x)$. Element x_i is unique when no other element is equal to it, so the expectation of Y is

$$\mathbb{E}[Y] = \sum_{i=1}^{n} \Pr(Y_i = 1) = n \cdot \left(1 - \frac{1}{n}\right)^{n-1} \geqslant \frac{n}{e}.$$

If, for contradiction, the claim is false and $\Pr_{x \sim U^n}\left(\text{unique}\,(x) \geqslant \frac{n}{2e}\right) < \frac{1}{10}$, then because Y is at most n, a Markov type argument proves that

$$\mathbb{E}[Y] < \frac{1}{10} \cdot n + \frac{9}{10} \cdot \frac{n}{2e} < \frac{n}{e} \leqslant \mathbb{E}[Y],$$

which is a contradiction. □

From hereon, we refer to any input $x \in [n]^n$ with unique $(()\,x) \geqslant n/2e$ as a **hard** input. By Claim 3, a constant fraction of inputs are hard.

Step 2. We break the computation of BP $\mathcal{B}$ down into b sub-computations, where the i-th sub-computation is the sub-program obtained by restricting the original branching program to layers $(i-1)\cdot\frac{T}{b}+1$ to $i\cdot\frac{T}{b}+1$. The i-th sub-computation begins at the vertex reached by the computation from the $(i-1)$-th sub-computation. In other words, we partition the time T spent by the BP into b blocks of T/b consecutive time steps.

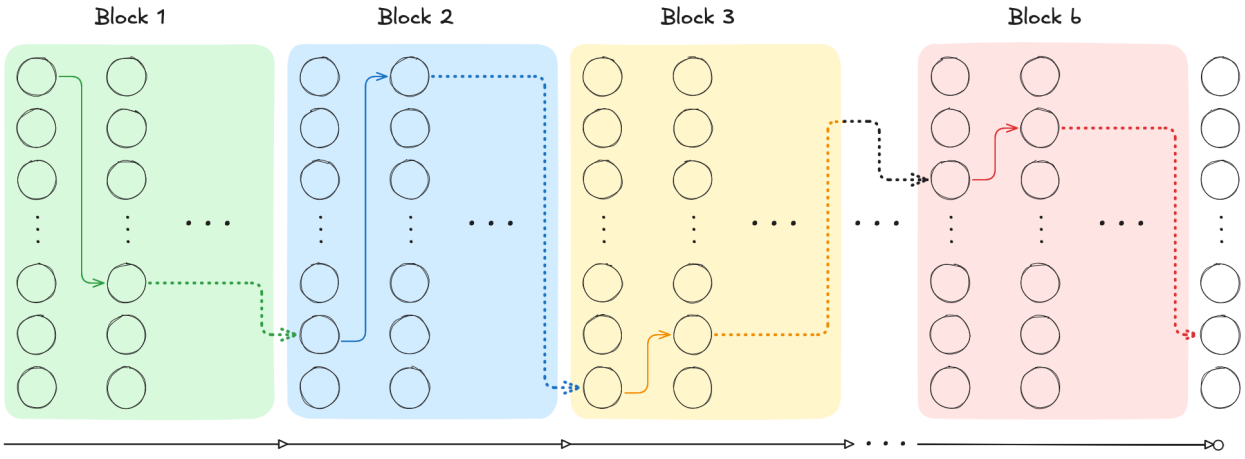

Figure 1: The computation of a branching program broken into b blocks. Each block has $\frac{T}{b}$ non-final layers.

We have the following easy claim.

Claim 4. *For each hard input x (namely, when* $\text{unique}(x) \geqslant \frac{n}{2e}$*), there exists a sub-computation in $\mathcal{B}$ that outputs at least $\dfrac{n}{2eb}$ indices.*

Proof. This is a simple application of the pigeonhole principle. There are at least $n/2e$ indices that needs to be output and b sub-programs in total, so one of them must output at least $1/b$ fraction of the output, concluding the proof. □

Step 3. The main technical step in this lower bound framework is this one, wherein one examines the power of *low-depth* BPs in correctly outputting a large number of elements (while entirely ignoring the width of the program in the proof). For the unique elements problem, even this step is not particularly technical.

Claim 5. *For* any *BP $\widetilde{\mathcal{B}}$ of depth $d \leqslant n/10$ and any $m \leqslant d/2$,*

$$\Pr_{x \sim U^n}(\widetilde{\mathcal{B}}(x) \text{ correctly outputs at least } m \text{ indices}) \leqslant 2^{-m}.$$

Proof. The BP $\widetilde{\mathcal{B}}$ can read at most $\frac{n}{10}$ indices and also output at most $\frac{n}{10}$ indices of x. Thus, there exists $\frac{8n}{10}$ indices I that are neither read nor outputted by $\widetilde{\mathcal{B}}$.

Let R be the read indices by $\widetilde{\mathcal{B}}$ and we condition on any choice x_R of these indices in the input x sampled from U^n. For the outputted indices of $\mathcal{B}$ to be correct, the $\frac{8n}{10}$ indices in I that are neither read nor outputted must not contain any of the outputted values (otherwise, we will have a duplicate). We have,

$$\begin{aligned}\Pr\left(\widetilde{\mathcal{B}}(x) \text{ correctly outputs} \geqslant m \text{ indices}\right) &= \mathop{\mathbb{E}}_{I,R,x_R} \Pr\left(\widetilde{\mathcal{B}}(x) \text{ correctly outputs} \geqslant m \text{ indices} \mid I, R, x_R\right) \\ &\qquad \text{(by the law of conditional probabilities)} \\ &\leqslant \left(1 - \frac{m}{n}\right)^{\frac{8n}{10}} \leqslant e^{-\frac{8m}{10}} \leqslant 2^{-m},\end{aligned}$$

where we used the fact that conditioned on X_I and I, the remaining entries of x, in particular, x_I are chosen uniformly at random, and each one should choose a different value than the m outputs. □

Step 4. The final step is to just count the number of sub-programs of depth $n/10$ in the BP $\mathcal{B}$.

Claim 6. *There are at most $2^S \cdot T$ sub-programs of depth $n/10$ that can ever be executed in $\mathcal{B}$.*

Proof. The sub-programs of depth $n/10$ can be uniquely identified with their root vertices, and there only $2^S \cdot T$ vertices in the branching program. □

Finally, we can tie everything together to obtain the desired lower bound.

Proof of Theorem 1. As we did earlier, fix $\mathcal{B}$ to be a branching program that solves unique elements with space S and time T. We break $\mathcal{B}$ down into $b = \frac{10T}{n}$ sub-computations of depth $\frac{n}{10}$. Certainly T must be at least n, as there are inputs that require $\mathcal{B}$ to output n indices.

For $x \sim U^n$ and a fixed sub-program $\widetilde{\mathcal{B}}$ of $\mathcal{B}$, the probability that the sub-program outputs at least $\frac{n}{2eb}$ indices is at most $2^{-\frac{n}{2eb}}$ by Step 3. By Step 4 and a union bound, we have,

$$\Pr_{x \sim U^n}(\textit{any sub-program } \widetilde{\mathcal{B}}(x) \text{ correctly outputs at least } n/2eb \text{ indices}) \leqslant 2^S \cdot T \cdot 2^{-\frac{n}{2eb}}.$$

On the other hand, Step 2 says it is necessary for some sub-program to output at least $\frac{n}{2eb}$ indices for $\mathcal{B}$ to output $\frac{n}{2e}$ indices. Thus,

$$\Pr_{x \sim U^n}(\mathcal{B}(x) \text{ correctly outputs at least } n/2e \text{ indices}) \leqslant 2^S \cdot T \cdot 2^{-\frac{n^2}{20eT}},$$

where we also used the bound $b = T/(10n)$. Finally, by Step 1, at least $\frac{1}{10}$ of the inputs require $\frac{n}{2e}$ output indices, so we must have

$$2^S \cdot T \cdot 2^{-\frac{n^2}{20eT}} \geqslant \frac{1}{10}.$$

Taking log of both sides, rearranging, and using the assumption that $S = \Omega(\log n)$ yields $S \cdot T = \Omega(n^2)$. □

1.2 Further Implication: A Sorting Lower Bound

A lower bound for sorting can be obtained by a reduction from unique elements. We first define the problem.

Problem 2 (Sorting). In the sorting problem, we have as input a list of n numbers $x_1, x_2, \ldots, x_n$ in $[n]$. The output is a permutation $\pi_1, \pi_2, \ldots, \pi_n$ of $[n]$ such that $x_{\pi_1} \leqslant x_{\pi_2} \leqslant \cdots \leqslant x_{\pi_n}$.

And below, we have a time-space tradeoff lower bound for sorting.

Corollary 7 ([Bea91])**.** *Any algorithm with S in $\Omega(\log n)$ that solves the sorting problem has $S \cdot T = \Omega(n^2)$.*

Proof. For contradiction, suppose algorithm $\mathcal{A}_{sort}$ solves sorting with $S_{sort} \cdot T_{sort} = o(n^2)$. We can solve unique elements as follows:

- Run $\mathcal{A}_{sort}$ on the input for a permutation $\pi_1, \pi_2, \ldots, \pi_n$. For convenience, define $x_{\pi_0} = x_{\pi_{n+1}} = \infty$.
- For each $1 \leqslant i \leqslant n$, if x_{π_i} is not equal to $x_{\pi_{i-1}}$ and x_{π_i} is not equal to $x_{\pi_{i+1}}$, then output index π_i as a unique element.

The correctness of this algorithm is straightforward. In the present form, the above algorithm takes $S_{sort} + O(n)$ space and $T_{sort} \cdot O(1)$ time (noting that T_{sort} is certainly $\Omega(n)$). However, we can modify the above algorithm to run step 2 for i whenever a step of 1 produces π_{i+1}, to use the results of $\mathcal{A}_{sort}$ "on the fly". This way, we only ever need to store the three most recent indices outputted by $\mathcal{A}_{sort}$ along with a constant amount of bookkeeping registers, improving the space usage to $S_{sort} \cdot O(1)$. This is an algorithm for unique elements with $S \cdot T = O(S_{sort} \cdot T_{sort}) = o(n^2)$, contradicting Theorem 1. □

Remark. We shall note that to simplify the exposition, we have defined both unique elements and sorting problems to output *indices* in the answer instead of the input numbers. This only served to simplify some of the calculations and provide a slightly cleaner arguments – the paper of [Bea91] instead prove the lower bounds for outputting the numbers instead of their indices.

2 Matrix Multiplication Over $\mathbb{F}_2$

We now adapt Beame's framework to provide an alternative time-space tradeoff lower bound for matrix multiplication proven in [Abr86]. The variant of matrix multiplication we study is defined as follows.

Problem 3 (Matrix Multiplication). In the matrix multiplication problem, we have as input two $n \times n$ matrices A and B with entries in $\{0, 1\}$. The goal is to output tuples $(i, j, (AB)_{ij})$ for all $i, j \in [n]$, where the product AB is computed over $\mathbb{F}_2$.

We can solve this problem in $T = \Theta(n^3)$ time and $S = \Theta(\log n)$ space on a RAM machine (or even $S = \Theta(1)$ space in a branching program) by computing each entry of AB one by one, using $O(1)$ registers to store the counters and accumulators. There is also an active line of research on improving the runtime (independent of space restriction) with current best bounds of $O(n^{2.372})$ [ADV+25]. It has been a longstanding open problem to determine if there is an algorithm for matrix multiplication with $T = \Theta(n^2)$. Abrahamson's lower bound says that if such an algorithm exists, it ought to use $\Omega(n^2)$ space.

Theorem 8 ([Abr86])**.** *Any algorithm with time T and space $S = \Omega(\log n)$ for matrix multiplication satisfies*

$$S \cdot T^2 = \Omega(n^6).$$

We note that one can also obtain a tradeoff of $S \cdot T = \Omega(n^3)$ using a somewhat more direct applications of Beame's framework. However, the bound in Theorem 8 is stronger than a tradeoff of $S \cdot T = \Omega(n^3)$ as the latter asserts $S = \Omega(n^3/T)$ while the former asserts $S = \Omega((n^3/T)^2)$, a quadratic improvement. Given the proof is not particularly more difficulty and is also more illuminating, we opt to just provide the stronger result directly. Let us again review the four steps of Beame's framework, now in the context of matrix multiplication:

1. We define a class of *"good"* matrices, and say that an input is *"hard"* if both A and $B^\top$ are good. We then show that a large proportion of inputs are *hard*.
2. Prove that after we break the computation into b subcomputations, one of these sub-computations must output $\Omega(n^2/b)$ tuples.
3. Prove that on a random hard input, the probability that a low depth sub-computation outputs $\Omega(n^2/b)$ tuples is very small.
4. Prove that there are not enough sub-programs of the original branching program to solve a random *hard* input.

To obtain the T^2 dependence in the tradeoff, we carefully pick the number of blocks we break the computation into, and also be more careful in the analysis of Step 3, which is the main technical step. Beside this, the only other step that is truly problem-dependent is Step 1 which involves defining good matrices and showing they appear frequently enough. Steps 2 and 4 are rather mechanical applications of the framework.

Step 1. Let $0 < \delta < \frac{1}{2}$ be a sufficiently small constant (proof of Lemma 10 specifies how small is enough). Define $t := \delta n$.

Definition 9. A matrix $M \in \mathbb{F}_2^{n \times n}$ is **good** if for all sets $I, J \subseteq [n]$ such that $|I| \leqslant t$ and $|J| \geqslant n - t$, the submatrix of M with rows in I and columns in J, denoted $M[I, J]$, has full row rank, i.e.

$$\operatorname{rank}(M[I, J]) = |I|.$$

In other words, a good matrix is one that even if we remove any small proportion of columns, any small subset of rows is linearly independent. We prove that most matrices are good. Let $U^{n \times n}$ denote the uniform distribution over $\mathbb{F}_2^{n \times n}$.

Lemma 10 ("Most matrices are good")**.**

$$\Pr_{M \sim U^{n \times n}}(M \text{ is good}) \geqslant 1 - o(1).$$

Proof. Let M be a uniformly random matrix in $\mathbb{F}_2^{n \times n}$. Let us also fix some $I, J \subset [n]$ with $|I| \leqslant t$ and $|J| \geqslant n - t$. The probability that $M[I, J]$ does not have full rank is the probability that there exists a

nonzero vector $c \in \mathbb{F}_2^{|I|}$ such that $c^\top \cdot M[I,J] = 0$. In $\mathbb{F}_2$, for a fixed c, the distribution of $c^\top M[I,J]$ is uniform over $\mathbb{F}_2^{|J|}$, so the probability that $c^\top M[I,J] = 0$ is $2^{-|J|}$. By the union bound over all $2^{|I|}-1$ choices of non-zero c, we have,

$$\Pr(M[I,J] \text{ does not have full row rank}) \leqslant 2^{|I|} \cdot 2^{-|J|} = 2^{|I|-|J|} \leqslant 2^{2t-n}.$$

The number of choices of I and J is at most

$$\left(\sum_{k=0}^{t} \binom{n}{k}\right)^2 \leqslant \left(\frac{en}{t}\right)^{2t} = \left(\frac{e}{\delta}\right)^{2\delta n} = 2^{O(\delta n \log(1/\delta))},$$

where we used the inequality $\sum_{k=0}^{t} \binom{n}{k} \leqslant (en/t)^t$. By the union bound over all I, J, we have that

$$\Pr(M \text{ is not good}) \leqslant 2^{O(\delta n \log(1/\delta))} \cdot 2^{2\delta n - n} = 2^{-\Omega(n)}$$

for sufficiently small δ. Thus the probability that M is good is at least $1 - 2^{-\Omega(n)} = 1 - o(1)$. □

We say an input (A, B) to the problem is **hard** if both A and $B^\top$ are good. By Lemma 10 and a union bound, we have that for A, B sampled uniformly and independently from $\mathbb{F}_2^{n\times n}$,

$$\Pr\left(\text{input } (A,B) \text{ is hard}\right) \geqslant 1 - o(1). \tag{1}$$

(Note that distribution of $B^\top$ and B is the same so we can apply the lemma).

Step 2. Suppose $\mathcal{B}$ solves matrix multiplication. As before, we can divide $\mathcal{B}$'s computation into b sub-computations. For any input (A, B), there exists a sub-computation of $\mathcal{B}$ that outputs at least n^2/b tuples. This follows again from the pigeonhole principle, since $\mathcal{B}$ is always required to output n^2 tuples by the end of its computation.

Step 3. The main technical step is Step 3, where we will devote most of our attention to.

Lemma 11. *Let d and m be such that $(d/t)^2 \leqslant m/2$. Let $\widetilde{\mathcal{B}}$ be a branching program of depth d (recall that this step does not rely on the width of the program). We have,*

$$\Pr_{A,B\sim U^{n\times n}}\left(\widetilde{\mathcal{B}} \text{ correctly outputs at least } m \textit{ distinct tuples on } (A,B) \mid (A,B) \textit{ is hard}\right) \leqslant 2^{-\Omega(m)}.$$

The heart of the proof is the following claim.

Claim 12. *Fix a computation path (i.e., a transcript of input queries/answers and outputs) of $\widetilde{\mathcal{B}}$ in which $\widetilde{\mathcal{B}}$ correctly outputs m distinct tuples. The number of hard inputs (A, B) that is consistent with this transcript and $\widetilde{\mathcal{B}}$ is correct on is at most $2^{2n^2 - q_A - q_B - \Omega(m)}$, where q_A and q_B are the number of distinct entries of A and B that $\widetilde{\mathcal{B}}$ reads in this computation path, respectively.*

Proof. Call a row of A **heavy** if $\mathcal{B}$ reads at least t entries of that row, and **light** otherwise. Similarly, call a column of B **heavy** if $\mathcal{B}$ reads at least t entries of that column and **light** otherwise.

As $\mathcal{B}$ can read at most d entries in A and B in its computation, there are at most d/t heavy rows and at most d/t heavy columns. This means at most $(d/t)^2$ tuples output by $\widetilde{\mathcal{B}}$ can be in both a heavy row and a heavy column. Using the assumption $(d/t)^2 \leqslant m/2$, at least half the tuples must be in either light rows or light columns. Then one of the following must be true:

- $m/4$ tuples are in light columns of B, or
- $m/4$ tuples are in light rows of A.

We will assume the former and use that A is good; the latter case is symmetric and uses that $B^\top$ is good.

From here on, we fix a completion of the unknown entries of A. The remaining randomness is in the unknown entries of B. Let us count how many completions of B are consistent with $\widetilde{\mathcal{B}}$'s output.

For each light column $j \in [n]$ with at least one output tuple,

- $I_j \subseteq [n]$ be the rows i such that tuple $(i, j, (AB)_{ij})$ is output,
- $I'_j \subseteq [n]$ be I_j if $|I_j| \leqslant t$, and be an arbitrary subset of I_j of size t otherwise, and
- $J_j \subseteq [n]$ be the rows of B that $\mathcal{B}$ does *not* read in column j,

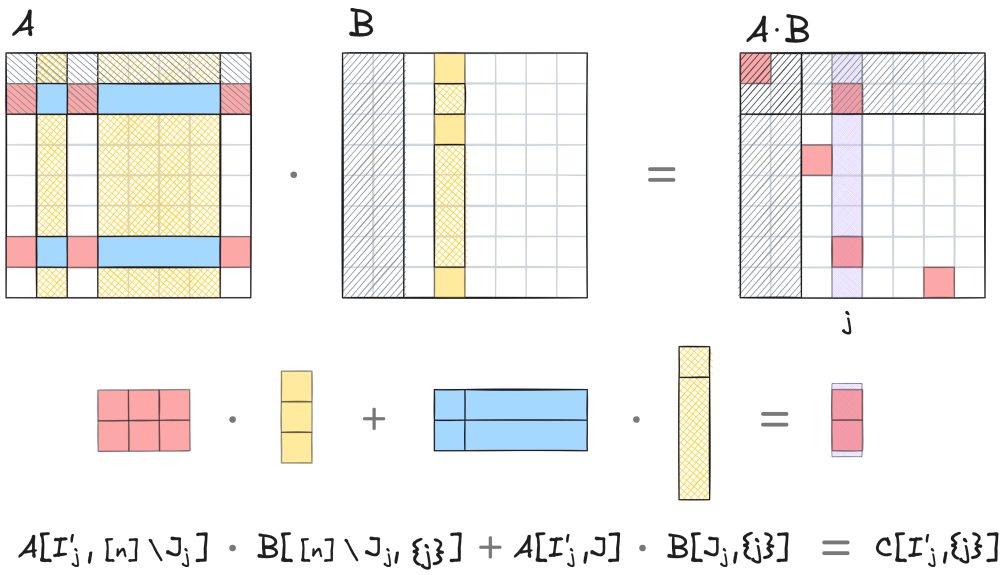

Figure 2: A visualization of the sets I'_j and J_j for a light column j. The elements in I'_j are the red-filled rows in A while the elements in J'_j are the set of yellow-shaded columns. The black-shaded rows and columns are heavy. The red-filled cells in $A \cdot B$ are the output entries of $\widetilde{\mathcal{B}}$, and the yellow-filled and yellow-shaded cells in B are the read and unread entries of B, respectively. The submatrix $A[I'_j, J_j]$ is highlighted in blue.

By definition, $|I'_j| \leqslant t$, and as j is a light column, $|J_j| \geqslant n - t$. Because A is good, the submatrix $A[I'_j, J_j]$ has full row rank. The equation corresponding to the output entry (i, j, c), $i \in I'_j$ is

$$c = \sum_{k=1}^{n} A_{ik} B_{kj} = \sum_{k \in J_j} A_{ik} B_{kj} + \sum_{k \notin J_j} A_{ik} B_{kj}.$$

By moving the deterministic second sum to the LHS, it can be rewritten as

$$A[I'_j, J_j] \cdot B[J_j, \{j\}] = c'$$

for some deterministic vector c'. This imposes $|I'_j| = \min(|I_j|, t)$ linearly independent constraints on the variables in the j-th column of B. Recalling that $|I_j| < n$, we have

$$\min(|I_j|, t) \geqslant \frac{t}{n} \cdot |I_j| = \delta \cdot |I_j|.$$

Between different light columns j, the constraints are on disjoint sets of variables, so the total number of independent constraints on B is

$$\sum_j |I'_j| \geqslant \sum_j \delta \cdot |I_j| \geqslant \delta \cdot \frac{m}{4},$$

as the sum of $|I_j|$ is at least $m/4$. Thus, the number of completions of B consistent with $\widetilde{\mathcal{B}}$'s output is at most $2^{n^2-q_B-\delta m/4}$, with q_B being the number of known entries of B. This applies for any completion of A, which are $2^{n^2-q_A}$ many, so the number of hard inputs (A,B) consistent with $\mathcal{B}$'s output is upper bounded by $2^{2n^2-q_A-q_B-\delta m/4}$. □

Proof of Lemma 11. Consider the uniform distribution $U^{n\times n}$ over matrices in $\mathbb{F}^{n\times n}$. Let Correct be the event that $\widetilde{\mathcal{B}}$ correctly outputs at least m distinct tuples on the input (A,B), and Hard be the event that the input (A,B) is hard.

Fixing a computation path τ, the total number of (A,B), not necessarily hard, consistent with this path is $2^{2n^2-q_A-q_B}$. Moreover, if (A,B) is a hard input, then, by Claim 12 only $2^{-\Omega(m)}$ fraction of these paths can lead to the event Correct; formally,

$$\Pr_{A,B\sim U^{n\times n}}(\mathsf{Correct}\wedge\mathsf{Hard}\mid\tau)=\frac{\text{number of inputs consistent with }\tau\text{ and satisfying }\mathsf{Correct}\wedge\mathsf{Hard}}{\text{number of inputs consistent with }\tau}\leqslant 2^{-\Omega(m)}.$$

Averaging over all choices of τ, we have

$$\Pr_{A,B\sim U^{n\times n}}(\mathsf{Correct}\wedge\mathsf{Hard})\leqslant 2^{-\Omega(m)}.$$

On the other hand, by Eq (1) we know that $(A,B)\sim U^{n\times n}$ is hard with probability $1-o(1)$, so

$$\Pr_{A,B\sim U^{n\times n}}(\mathsf{Correct}\mid\mathsf{Hard})=\frac{\Pr_{A,B\sim U^{n\times n}}(\mathsf{Correct}\wedge\mathsf{Hard})}{\Pr_{A,B\sim U^{n\times n}}(\mathsf{Hard})}\leqslant\frac{2^{-\Omega(m)}}{1-o(1)}=2^{-\Omega(m)}.$$

This concludes the proof. □

Step 4. This step is exactly as before. For any d, there are at most $2^S\cdot T$ sub-programs of depth d that can ever be executed in a branching program of space S and T.

Finally, we can prove our matrix multiplication lower bound.

Proof of Theorem 8. Let $\mathcal{B}$ be a BP that solves matrix multiplication with space S and time T. We break $\mathcal{B}$ down into b sub-computations of depth d for

$$b:=\frac{2T^2}{\delta^2n^4}\qquad\text{and}\qquad d:=\frac{T}{b}=\frac{\delta^2n^4}{2T}.$$

By Step 2, we have that on *any* input, $\mathcal{B}$ should have a depth-d sub-program that outputs $m=n^2/b$ tuples. Let $m=n^2/b$ and note that we have

$$\frac{m}{2}=\frac{n^2}{2b}=\frac{\delta^2\cdot n^6}{4T^2}\qquad\text{and}\qquad\frac{d}{t}=\frac{T}{\delta n\cdot b}=\frac{\delta\cdot n^3}{2T},$$

which means $(m/2)\geqslant(d/t)^2$. Thus, we can apply Lemma 11 to have, for any fixed depth-d sub-BP $\widetilde{\mathcal{B}}$,

$$\Pr_{A,B\sim U^{n\times n}}\left(\widetilde{\mathcal{B}}\text{ outputs at least }n^2/b\text{ distinct tuples on }(A,B)\mid(A,B)\text{ is hard}\right)\leqslant 2^{-\Omega(n^2/b)}.$$

Using Step 4 and a union bound, we further have

$$\Pr_{A,B\sim U^{n\times n}}\left(\mathcal{B}\text{ outputs }\geqslant n^2/b\text{ distinct tuples on }(A,B)\text{ in one sub-program }\mid(A,B)\text{ is hard}\right)\leqslant T\cdot 2^S\cdot 2^{-\Omega(n^2/b)}.$$

But since by Eq (1), the probability of (A,B) being hard is $1-o(1)$ (even just non-zero is enough for this part) and $\mathcal{B}$ is supposed to be correct on all inputs (and by Step 2 should be able to output n^2/b tuples in one-subprogram), we must have

$$1\leqslant 2^S\cdot T\cdot 2^{-\Omega(n^2/b)}=2^S\cdot T\cdot 2^{-\Omega(n^6/T^2)}.$$

Using the assumptions that $S = \Omega(\log T)$ and $T = \Omega(n^2)$, we get the desired time-space tradeoff of

$$S \cdot T^2 = \Omega(n^6),$$

concluding the proof. □

This concludes our derivation of the lower bound for matrix multiplication in [Abr86]. We have seen how to adapt Beame's framework to reprove this result (and also obtain a stronger tradeoff than $S \cdot T = \Omega(n^3)$ that directly follows from the framework); the main technical step here was Step 3, where we had to be more careful in proving that we cannot output many entries of AB with too little information on A, B.

Remark. For simplicity of exposition and to showcase the main ideas, we focused on proving the lower bounds for deterministic algorithms that always output the correct answer. However, almost exactly the same proofs also apply to randomized algorithms/branching programs that need to be correct with some constant probability and S and T being their expected space and time complexity.

References

[Abr86] Karl Abrahamson. Time-space tradeoffs for branching programs contrasted with those for straight-line programs. In *27th Annual Symposium on Foundations of Computer Science*, pages 402–409, 1986. 1, 5, 6, 10

[ADV+25] Josh Alman, Ran Duan, Virginia Vassilevska Williams, Yinzhan Xu, Zixuan Xu, and Renfei Zhou. More asymmetry yields faster matrix multiplication. In Yossi Azar and Debmalya Panigrahi, editors, *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, pages 2005–2039. SIAM, 2025. 6

[Bea91] Paul Beame. A general sequence time-space tradeoff for finding unique elements. *SIAM J. Comput.*, 20(2):270–277, 1991. 1, 2, 5