Computational Complexity Theory

Lecture 19: Perfect matching in RNC; Class BPL; Randomized reductions

Department of Computer Science,
Indian Institute of Science

Recap: BPP is in PH

- We saw that $P \subseteq BPP \subseteq EXP$. But, is $BPP \subseteq NP$? Not known! (Yes, people still believe BPP = P.)

- Sipser showed $BPP \subseteq PH$, Gacs strengthened it to $BPP \subseteq \Sigma_2 \cap \Pi_2$, Lautemann gave a simpler proof.

- Theorem. *(Sipser-Gacs-Lautemann '83)* $BPP \subseteq \Sigma_2 \cap \Pi_2$.

Recap: Derandomization of BPP ?

- Can the Sipser-Gacs-Lautemann theorem be strengthened? How low in the PH does BPP lie ?

- Theorem. *(Nisan & Wigderson 1988,…, Umans 2003)* If there's a $L \in \mathrm{DTIME}(2^{O(n)})$ and a constant $\varepsilon > 0$ such that any circuit C_n that decides $L \cap \{0,1\}^n$ requires size $2^{\varepsilon n}$, then BPP = P .

- Lower bounds ➡ Derandomization !
- Caution: Shouldn't interpret this result as "randomness is useless".

Recap: Class RP

- Class RP is the <u>one-sided error</u> version of BPP.

- Definition. A language L is in RTIME(T(n)) if there's a PTM M that decides L in O(T(n)) time such that

$$x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3$$
$$x \notin L \Rightarrow \Pr[M(x) = 0] = 1.$$

- Definition. $RP = \bigcup_{c > 0} RTIME(n^c)$.

- Clearly, $RP \subseteq BPP$. Obs. $RP \subseteq NP$.

Recap: Class co-RP

- Definition. co-RP = $\{L : \overline{L} \in \mathrm{RP}\}$.

- Obs. A language L is in co-RP if there's a PTM M that decides L in poly-time such that

$$x \in L \Longrightarrow \Pr[M(x) = 1] = 1$$

$$x \notin L \Longrightarrow \Pr[M(x) = 0] \geq 2/3.$$

- Obs. co-RP $\subseteq$ BPP .

- Is RP$\cap$co-RP in P ? Not known!

Recap: Class ZPP

- **Definition.** A language L is in $\mathrm{ZTIME}(T(n))$ if there's a PTM M s.t. on every input x, $M(x) = L(x)$ whenever M halts, and M has expected running time $O(T(n))$.
- **Definition.** $\mathrm{ZPP} = \bigcup_{c>0} \mathrm{ZTIME}(n^c)$.
- Problems in ZPP are said to have poly-time *Las Vegas algorithms*, whereas those in BPP are said to have poly-time *Monte-Carlo algorithms*.

- **Theorem.** $\mathrm{ZPP} = \mathrm{RP} \cap \text{co-RP} \subseteq \mathrm{BPP}$. *(Assignment)*

- **Note.** If $\mathrm{P} = \mathrm{BPP}$ then $\mathrm{P} = \mathrm{ZPP} = \mathrm{BPP}$.

Perfect Matching in RNC

Randomness brings in simplicity

- The use of randomness helps in designing *simple* and efficient algorithms for many problems.
- We'll see one such algorithm in this lecture, namely an efficient randomized, <u>parallel</u> algorithm to check if a given bipartite graph has a perfect matching.

Class RNC

- The use of randomness helps in designing *simple* and efficient algorithms for many problems.
- We'll see one such algorithm in this lecture, namely an efficient randomized, parallel algorithm to check if a given bipartite graph has a perfect matching.
- Definition. A language L is in RNC^i if there's a *randomized* $O((\log n)^i)$-time parallel algorithm M that uses $n^{O(1)}$ parallel processors s.t. for every $x \in \{0,1\}^*$,

  $x \in L \Rightarrow \Pr[M(x) = 1] \geq 2/3$,

  $x \notin L \Rightarrow \Pr[M(x) = 0] = 1$.

  Here, n is the input length.

Class RNC

- The use of randomness helps in designing *simple* and efficient algorithms for many problems.
- We'll see one such algorithm in this lecture, namely an efficient randomized, parallel algorithm to check if a given bipartite graph has a perfect matching.

- Definition. $\mathrm{RNC} = \bigcup_{i > 0} \mathrm{RNC}^i$.

- RNC stands for Randomized NC. We can alternatively define RNC using (uniform) circuits.

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- The input $G = (L \dot\cup R, E)$ is given as a $n \times n$ *biadjacency matrix* $A = (a_{ij})_{i,j \in n}$, where $n = |L| = |R|$.

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- The input $G = (L \cup R, E)$ is given as a $n \times n$ *biadjacency matrix* $A = (a_{ij})_{i,j\in n}$, where $n = |L| = |R|$.

$a_{ij} = 1$ if there's an edge from the i-th vertex in L to the j-th vertex in R, otherwise $a_{ij} = 0$.

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- **Theorem.** *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- The input $G = (L \dot\cup R, E)$ is given as a $n \times n$ *biadjacency matrix* $A = (a_{ij})_{i,j \in n}$, where $n = |L| = |R|$.
- **Algorithm.**

1. Construct $B = (b_{ij})_{i,j \in n}$ as follows: If $a_{ij}=0$, then $b_{ij}=0$. Else, pick b_{ij} independently and uniformly <u>at random</u> from $[2n]$.
2. Compute $\det(B)$.
3. If $\det(B) \neq 0$ output "yes", else output "no".

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- The input $G = (L \dot{\cup} R, E)$ is given as a $n \times n$ *biadjacency matrix* $A = (a_{ij})_{i,j \in n}$, where $n = |L| = |R|$.
- Algorithm. (RNC^2 algorithm)

1. Construct $B = (b_{ij})_{i,j \in n}$ as follows: If $a_{ij}=0$, then $b_{ij}=0$. Else, pick b_{ij} independently and uniformly <u>at random</u> from $[2n]$. (This can be done using n^2 processors.)
2. Compute $\det(B)$. (determinant is in NC^2, *Csanky '76*)
3. If $\det(B) \neq 0$ output "yes", else output "no".

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching ∈ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j \in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $\det(X) = \sum_{\sigma \in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i \in [n]} x_{i\,\sigma(i)}$.

- S_n is the set of all permutations on [n].

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j\in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $\det(X) = \sum_{\sigma\in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i\in[n]} x_{i\,\sigma(i)}$.

- Obs. $\det(X) \neq 0$ $\Longleftrightarrow$ G has a perfect matching.

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j\in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $\det(X) = \sum_{\sigma\in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i\in[n]} x_{i\,\sigma(i)}$.

- Obs. $\det(X) \neq 0$ $\Longleftrightarrow$ G has a perfect matching.

Polynomial in the x_{ij} variables.

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j\in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $\det(X) = \sum_{\sigma\in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i\in[n]} x_{i\,\sigma(i)}$.

- Obs. $\det(X) \neq 0 \iff G$ has a perfect matching.
- In the algorithm, we set $x_{ij} = b_{ij}$, where b_{ij} is picked randomly from $[2n]$ if $x_{ij} \neq 0$, otherwise $b_{ij} = 0$.

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j \in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $\det(X) = \sum_{\sigma \in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i \in [n]} x_{i\,\sigma(i)}$.

- Obs. $\det(X) \neq 0 \Longleftrightarrow G$ has a perfect matching.
- If $\det(X) = 0$ then $\det(B) = 0$. (So, the algorithm has one-sided error.)

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j\in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $$\det(X) = \sum_{\sigma\in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i\in[n]} x_{i\,\sigma(i)} .$$

- Obs. $\det(X) \neq 0 \iff G$ has a perfect matching.
- If $\det(X) \neq 0$, what is the probability that $\det(B) \neq 0$?

The answer is given by the **Schwartz-Zippel lemma**

Schwartz-Zippel lemma

- Lemma. *(Schwartz 1980, Zippel 1979)* Let $f(x_1, \ldots, x_n) \neq 0$ be a multivariate polynomial of (total) degree at most d over a field F. Let $S \subseteq F$ be finite, and $(a_1, \ldots, a_n) \in S^n$ such that each a_i is chosen independently and uniformly at random from S. Then,

$$\Pr_{(a_1, \ldots, a_n) \in_r S^n} [f(a_1, \ldots, a_n) = 0] \leq d/|S|.$$

- *Proof idea.* Roots are far fewer than non-roots. Use induction on the number of variables.

(Homework / reading exercise)

Perfect matching in RNC

- Let PerfectMatching = {Bipartite graph G : G has a perfect matching}.
- Theorem. *(Lovasz 1979)* PerfectMatching $\in$ RNC^2.
- Correctness of the Algorithm.

1. Define $X = (x_{ij})_{i,j\in n}$ as follows: If $a_{ij}=0$, then $x_{ij}=0$. Else, x_{ij} is a formal variable.
2. $\det(X) = \sum_{\sigma\in S_n} (-1)^{\mathrm{sign}(\sigma)} \prod_{i\in[n]} x_{i\,\sigma(i)}$.

- Obs. $\det(X) \neq 0$ $\Longleftrightarrow$ G has a perfect matching.
- If $\det(X) \neq 0$, then $\Pr[\det(B) \neq 0] \geq ½$ as degree of $\det(X) = n$ (by the Schwartz-Zippel lemma).

Perfect matching in RNC

- Theorem. *(Mulmuley, Vazirani, Vazirani 1987)* Finding a maximum matching in a general graph is in RNC^2.

- Is finding maximum matching in NC ? Open!

Perfect matching in RNC

- Theorem. *(Mulmuley, Vazirani, Vazirani 1987)* Finding a maximum matching in a general graph is in RNC^2.

- Is finding maximum matching in NC ? Open!
- Theorem. *(Fenner, Gurjar, Thierauf 2016; Svensson, Tarnawski 2017)* Finding a maximum matching in a general graph is in quasi-NC.

In $O((\log n)^3)$ time using $\exp(O((\log n)^3))$ processors,

Randomized space bounded computation

Space bounded PTMs

- We say a PTM M <u>uses $S(n)$ space</u> if on a length-n input, M halts using at most $S(n)$ cells of it work-tape *regardless of its random choices.*

- Definition. A language L is in BPL if there's a PTM M such that M uses $O(\log n)$-space and for every $x \in \{0,1\}^*$, $\Pr[M(x) = L(x)] \geq 2/3$.

Space bounded PTMs

- We say a PTM M <u>uses $S(n)$ space</u> if on a length-n input, M halts using at most $S(n)$ cells of it work-tape *regardless of its random choices*.

- Definition. A language L is in BPL if there's a PTM M such that M uses $O(\log n)$-space and for every $x \in \{0,1\}^*$, $\Pr[M(x) = L(x)] \geq 2/3$.

- The success probability can be amplied as before as the BPP error reduction trick can be implemented using log-space. *(Homework)*

Space bounded PTMs

- We say a PTM M <u>uses $S(n)$ space</u> if on a length-n input, M halts using at most $S(n)$ cells of it work-tape *regardless of its random choices.*

- Definition. A language L is in RL if there's a PTM M s.t. M uses $O(\log n)$-space and for every $x \in \{0,1\}^*$,

$$x \in L \Longrightarrow \Pr[M(x) = 1] \geq 2/3$$
$$x \notin L \Longrightarrow \Pr[M(x) = 0] = 1.$$

- Clearly, RL $\subseteq$ NL $\subseteq$ P and BPL $\subseteq$ BPP.

Space bounded PTMs

- We say a PTM M uses S(n) space if on a length-n input, M halts using at most S(n) cells of it work-tape *regardless of its random choices*.

- Claim. BPL $\subseteq$ P .
- *Proof idea*. Think of the adjancency matrix A of the configuration graph of the O(log n)-space PTM. Compute the probability of acceptance by taking powers of A. *(Assignment problem)*

Space bounded PTMs

- We say a PTM M uses S(n) space if on a length-n input, M halts using at most S(n) cells of it work-tape *regardless of its random choices*.

- Claim. BPL $\subseteq$ P .
- *Proof idea*. Think of the adjancency matrix A of the configuration graph of the O(log n)-space PTM. Compute the probability of acceptance by taking powers of A. *(Assignment problem)*

- Is BPL = L ? Many believe that the answer is "Yes" !

Space bounded PTMs

- Theorem. *(Nisan '92, '94)* If $L \in BPL$ then there's a poly-time, $O((\log n)^2)$-space TM that decides L.
- Theorem. *(Saks, Zhou '99)* If $L \in BPL$ then there's a $n^{O(\sqrt{\log n})}$-time, $O((\log n)^{1.5})$-space TM that decides L.
- Theorem. *(Hoza '21)* If $L \in BPL$ then there's a $O((\log n)^{1.5}(\sqrt{\log\log n})^{-1})$-space TM that decides L.
- The last two results extend Nisan's techniques on *read-once branching programs*.

Space bounded PTMs

- Theorem. *(Nisan '92, '94)* If $L \in BPL$ then there's a <u>poly-time</u>, $O((\log n)^2)$-space TM that decides L.
- Theorem. *(Saks, Zhou '99)* If $L \in BPL$ then there's a $n^{O(\sqrt{\log n})}$-time, $O((\log n)^{1.5})$-space TM that decides L.
- Theorem. *(Hoza '21)* If $L \in BPL$ then there's a $O((\log n)^{1.5}(\sqrt{\log\log n})^{-1})$-space TM that decides L.
- *"Recent Progress on Derandomizing Space-Bounded Computation" survey by Hoza (2022).*

Randomized reductions

Randomized reduction

- Definition. We say a L_1 reduces to a L_2 in <u>*randomized polynomial-time*</u>, denoted $L_1 \leq_r L_2$, if there's a poly-time PTM M s.t. for every $x \in \{0,1\}^*$

$$\Pr[L_1(x) = L_2(M(x))] \geq 2/3. \longleftarrow \text{Success probability}$$

- For arbitrary L_1 and L_2, we <u>may not be able to boost</u> the success probability $2/3$, and so, the above kind of reductions **needn't be transitive**.

Randomized reduction

- Definition. We say a L_1 reduces to a L_2 in <u>*randomized polynomial-time*</u>, denoted $L_1 \leq_r L_2$, if there's a poly-time PTM M s.t. for every $x \in \{0,1\}^*$

$$\Pr[L_1(x) = L_2(M(x))] \geq 2/3.$$

- For arbitrary L_1 and L_2, we may not be able to boost the success probability $2/3$, and so, the above kind of reductions **needn't be transitive**. However,
- Obs. If $L_1 \leq_r L_2$ and $L_2 \in$ BPP, then $L_1 \in$ BPP.

(Easy homework)

Randomized reduction

- Definition. We say a L_1 reduces to a L_2 in *randomized polynomial-time*, denoted $L_1 \leq_r L_2$, if there's a poly-time PTM M s.t. for every $x \in \{0,1\}^*$

$$\Pr[L_1(x) = L_2(M(x))] \geq 2/3.$$

- Obs. If L_2 = SAT, then we can boost the success probability from $\frac{1}{2} + |x|^{-c}$ to $1 - \exp(-|x|^d)$.
- *Proof idea.* BPP error reduction trick + Cook-Levin.

(homework)

Randomized reduction

- Definition. We say a L_1 reduces to a L_2 in *randomized polynomial-time*, denoted $L_1 \leq_r L_2$, if there's a poly-time PTM M s.t. for every $x \in \{0,1\}^*$

$$\Pr[L_1(x) = L_2(M(x))] \geq 2/3.$$

- Obs. If $L_2 = SAT$, then we can boost the success probability from $\frac{1}{2} + |x|^{-c}$ to $1 - \exp(-|x|^d)$.

- Recall, $NP = \{L : L \leq_p SAT\}$. It makes sense to define a similar class using randomized poly-time reduction.

Class BP.NP

- **Definition.** We say a L_1 reduces to a L_2 in *randomized polynomial-time*, denoted $L_1 \leq_r L_2$, if there's a poly-time PTM M s.t. for every $x \in \{0,1\}^*$

$$\Pr[L_1(x) = L_2(M(x))] \geq 2/3.$$

- **Obs.** If L_2 = SAT, then we can boost the success probability from $\frac{1}{2} + |x|^{-c}$ to $1 - \exp(-|x|^d)$.
- **Definition.** $\mathbf{BP.NP} = \{L : L \leq_r \mathrm{SAT}\}$.
- Class BP.NP is also known as AM (*Arthur-Merlin protocol*) in the literature.

Class BP.NP

- Definition. BP.NP $= \{L : L \leq_r \text{SAT}\}$.
- Observe that NP $\subseteq$ BP.NP and BPP $\subseteq$ BP.NP. Is BP.NP $=$ NP ?

Class BP.NP

- Definition. BP.NP = $\{L : L \leq_r \text{SAT}\}$.
- Observe that NP $\subseteq$ BP.NP and BPP $\subseteq$ BP.NP. Is BP.NP = NP ? Many believe that the answer is "yes".

- Theorem. If certain reasonable circuit lower bounds hold, then BP.NP = NP.
- *Proof idea*. Similar to Nisan & Wigderson's conditional BPP = P result.

Class BP.NP

- Definition. $\mathrm{BP.NP} = \{L : L \leq_r \mathrm{SAT}\}$.
- Observe that NP $\subseteq$ BP.NP and BPP $\subseteq$ BP.NP. Is BP.NP = NP ? Many believe that the answer is "yes".

- We may further ask:

1. Is BP.NP in PH? Recall, BPP is in PH.

Class BP.NP

- Definition. BP.NP = $\{L : L \leq_r \text{SAT}\}$.
- Observe that NP $\subseteq$ BP.NP and BPP $\subseteq$ BP.NP. Is BP.NP = NP ? Many believe that the answer is "yes".

- We may further ask:

1. Is BP.NP in PH? Recall, BPP is in PH.

2. Is $\overline{\text{SAT}} \in$ BP.NP? Recall, if $\overline{\text{SAT}} \in$ BPP then PH collapses. (SAT $\in$ BP.NP as NP $\subseteq$ BP.NP .)

Class BP.NP

- Definition. BP.NP = {L : L $\leq_r$ SAT}.

- Theorem. BP.NP is in Σ_3. (In fact, BP.NP is in Π_2.)
- *Proof idea.* Similar to the Sipser-Gacs-Lautemann theorem. *(Assignment problem)*

Class BP.NP

- Definition. BP.NP = $\{L : L \leq_r \text{SAT}\}$.

- Theorem. BP.NP is in $\sum_3$. (In fact, BP.NP is in $\prod_2$.)
- *Proof idea.* Similar to the Sipser-Gacs-Lautemann theorem. *(Assignment problem)*

- Wondering if BP.NP $\subseteq \prod_2$ implies BP.NP $\subseteq \sum_2$? Is BP.NP = co-BP.NP ? (Recall, BPP = co-BPP).
- If BP.NP = co-BP.NP then co-NP $\subseteq$ BP.NP. The next theorem shows that this would lead to PH collapse.

Class BP.NP

- Definition. $\mathrm{BP.NP} = \{L : L \leq_r \mathrm{SAT}\}$.

- Theorem. If $\overline{\mathrm{SAT}} \in \mathrm{BP.NP}$ then $\mathrm{PH} = \Sigma_3$ (in fact, $\mathrm{PH} = \Sigma_2$).
- *Proof idea.* Similar to Adleman's theorem + Karp-Lipton theorem. *(Assignment problem)*

Class BP.NP

- Definition. BP.NP = $\{L : L \leq_r SAT\}$.

- Theorem. If $\overline{SAT} \in$ BP.NP then PH = $\sum_2$.

- We would use the above theorem to show that if GI is NP-complete then PH collapses.
- Thus, even without designing an efficient algorithm for GI, we know GI is unlikely to be NP-complete!

Class BP.NP

- Definition. $\mathrm{BP.NP} = \{L : L \leq_r \mathrm{SAT}\}$.
- Theorem. If $\overline{\mathrm{SAT}} \in \mathrm{BP.NP}$ then $\mathrm{PH} = \Sigma_2$.
- We would use the above theorem to show that if GI is NP-complete then PH collapses.
- Theorem. *(Goldwasser-Sipser '87, Boppana, Hastad, Zachos '87)* $\mathrm{GNI} \in \mathrm{BP.NP}$.
- *Proof.* We'll prove it.

Class BP.NP

- Definition. BP.NP = $\{L : L \leq_r \text{SAT}\}$.
- Theorem. If $\overline{\text{SAT}} \in$ BP.NP then PH = Σ_2.
- We would use the above theorem to show that if GI is NP-complete then PH collapses.
- Theorem. *(Goldwasser-Sipser '87, Boppana, Hastad, Zachos '87)* GNI $\in$ BP.NP.
- If GI is NP-complete then GNI is co-NP-complete. If so, then the above two theorems imply PH = Σ_2.

Graph Isomorphism in Quasi-P

- Theorem. *(Babai 2015)* There's a deterministic $\exp(O((\log n)^3))$ time algorithm to solve the graph isomorphism problem.