CS388T: ANSWER TO FINAL EXAM

VISHVAS VASUKI

Remark 0.0.1. Due by 1400 on May 8 in TAY 3.146. Clue: "Keep Savitch's theorem in mind."

1. Question

Problem 15.5.3. Show that if $NC_{i+1} = NC_i$, then $NC = NC_i$. That is, if two consecutive levels of the NC hierarchy coincide, the whole hierarchy collapses to that level. (Compare with Theorem 17.9; if fact, the proofs of the two results are not unrelated.)

1.1. Solution.

Notation 1.1.1. Let n be the size of the input. Below, when we refer to circuits, we refer to constant fan-in, single fan-out circuits whose basis are the AND, OR and NOT gates.

Lemma 1.1.2. *If $NC_j = NC_i$, for some $j > i$, then $NC_{j+1} = NC_{i+1}$.*

Proof. We already know that $NC_{i+1} \subseteq NC_{j+1}$. Below we show that $NC_{i+1} \supseteq NC_{j+1}$.

Consider a tree-like circuit C of depth $O((\log n)^{j+1})$. C can be viewed as a collection of subcircuits of depth $O((\log n)^j)$, C1, C2 etc.., whose outputs are combined by a subcircuit D of depth $O(\log n)$ to give the final output.

If $NC_j = NC_i$, then any tree-like polynomial sized circuit of depth
$O((\log n)^j)$ can be replaced by an equivalent tree-like polynomial sized circuit of depth
$O((\log n)^i)$. Both circuits compute the same boolean function. So, C1, C2 etc.. can be replaced by equivalent circuits of depth $O((\log n)^i)$, and their outputs can be combined by the subcircuit D. Thus, we have constructed a method for replacing any polynomial sized circuit of depth $O((\log n)^{j+1})$ with an equivalent circuit of depth $O((\log n)^{i+1})$.

Thus, considering the definition of NC_k, we arrive at the result. □

Theorem 1.1.3. *If $NC_{i+1} = NC_i$, then $NC = NC_i$.*

Proof. We prove this by induction. Using the lemma, we see that if $NC_{i+1} = NC_i$, then $NC_{i+2} = NC_{i+1} = NC_i$. Using the lemma again, we find that $NC_{i+3} = NC_{i+1} = NC_i$. By induction, considering the lemma, we see that if $NC_{i+1} = NC_i$, then for all $j > i$, $NC_j = NC_{i+1} = NC_i$.

Besides this, by the definition of NC_i, we know that, for all $j \leq i$, $NC_j \subseteq NC_i$.

As $NC = \bigcup_j NC_j$, we have the result. □

2. Question

Describe an RNC algorithm for deciding if a bipartite graph has a perfect matching. Explain why our algorithm is RNC.

2.1. **Solution.**

Notation 2.1.1. Let n be the size of the input. Below, when we refer to circuits, we refer to constant fan-in, single fan-out circuits whose basis are the AND, OR and NOT gates.

Let PERFECT-MATCH denote the perfect matching decision problem. Let m be the number of edges in the bipartite graph.

A rough sketch of the algorithm is described in [1] in pages 381 and 244. It was also described in class.

Use a square matrix, called the Tutte matrix A, to represent the bipartite graph. This is a symbolic matrix, where $A_{i,j}$ is 0 if there is no corresponding edge in the bipartite graph, and is some symbol x_i otherwise.

The determinant of A is a polynomial with m variables, which is identically equal to 0 if and only if a perfect matching does not exist.

Our algorithm is as follows:

Algorithm 2.1.2. ALG-PERFECT-MATCH

- **Input:** A Tutte matrix, A, corresponding to the bipartite graph, m random integers between 0 and 2m. A decision about whether the bipartite graph has a perfect matching. Let A' represent the matrix obtained by substituting the symbols with the m random integers.
- **Output:** Solution to an instance of PERFECT-MATCH.
- **Algorithm:**
- Compute the determinant of A' (det A').
- If det $A' = 0$, reply that there is probably no perfect matching. Otherwise, reply that there is definitely a perfect matching.

Theorem 2.1.3. *$PERFECT - MATCH \in RNC$.*

Proof. ALG-PERFECT-MATCH has one-sided error - It has false negatives. As observed in section 11.1 in [1], due to Schwartz-Zippel lemma, we know that the false negative probability of ALG-PERFECT-MATCH is no more than 1/2.

As observed in class and in section 15.1 of [1], using a truncated matrix power series for a symbolic matrix derived from A', det A' can be found in polylogarithmic parallel time and polynomial work in the size of the input. Hence, this part of the algorithm is in NC, and there is a uniform family of polynomial sized circuits of polylogarithmic depth to compute det A'. The other steps of ALG-PERFECT-MATCH require constant time.

So, one can construct a uniform family of circuits (of polylogarithmic depth) to embody ALG-PERFECT-MATCH. Each circuit accepts as input a Tutte matrix and m random integers, and solves PERFECT-MATCH correctly with an arbitrarily low (false-negative) error probability. These characteristics fit the definition of the class RNC, as stated in [1]. □

3. Question

Prove that $RP \subseteq P/Poly$.

3.1. **Solution.** In [1] (Theorem 11.6 page 269), and also in class, it was proved that $BPP \subseteq P/Poly$. This proof, while not quoting that theorem directly, uses similar arguments.

Theorem 3.1.1. *$RP \subseteq P/Poly$.*

Proof. Consider any language L in RP. L has a polynomial time randomized algorithm RP-ALG-L', which is always correct for any input $x \notin L$ and is correct at least half the time for $x \in L$. RP-ALG-L' can be repeated a couple of times independently, to reduce the error probability to 1/4, thereby obtaining another polynomial time algorithm RP-ALG-L. RP-ALG-L takes time at most p(n) for an input of size n, where p(n) is a polynomial in n.

By repeating RP-ALG-L $m = 12(n+1)$ times with independent random choices, and by returning the answer found by RP-ALG-L in a majority of these trials, we have another polynomial time randomized algorithm, RP-ALG-L-REP. The expected number of false negative errors RP-ALG-L makes in m independent runs is at most m/4. RP-ALG-L-REP makes an error only if more than m/2 runs of RP-ALG-L produce erroneous output. By applying a Chernoff bound, for a given input x of size n, we see that RP-ALG-L-REP has a reduced error probability of $Pr(Err(x)) \leq 2^{-(n+1)}$.

RP-ALG-L-REP can be viewed as a polynomial time algorithm, which takes at most p(n)m time, and which makes at most p(n)m random choices during its execution. Without loss of generality, we can assume each random choice to be a binary choice.

Without loss of generality, assume that the input alphabet is binary. Then, the number of inputs of size n is at most 2^n. Applying the union bound, we see that $Pr(\bigcup_x Err(x)) \leq 2^n 2^{-(n+1)} = 2^{-1} < 1$. In other words, there exists atleast one sequence of random choices the randomized algorithm RP-ALG-L-REP can make, which will result in the algorithm producing the correct answer for all inputs of size n.

So, for a given input size n, we can derandomize RP-ALG-L-REP and produce a polynomial time deterministic algorithm $ALG-L-REP_n$ which will produce the correct answer. Note that this deterministic algorithm is specific to a particular input size, n.

Given $ALG-L-REP_n$, we can produce a polynomial sized circuit C_n which will decide the "slice" of L corresponding to inputs of size n. Hence, L has a polynomial sized family of circuits.

As this is true of all L in RP, $RP \subseteq P/Poly$. □

4. Question

Prove that uniform NC is strictly contained in PSPACE. (That is, there are languages in PSPACE that cannot be computed in uniform NC.)

4.1. **Solution.**

Definition 4.1.1. Let $LSPACE = \bigcup_k SPACE((\log n)^k)$.

Lemma 4.1.2. *$LSPACE \subset PSPACE$*

Proof. We use the same technique used in [1] (page 145) to prove that $P \subset EXP$.

Consider the space required by any language in LSPACE. Let $c > 1$ be a constant. For every possible value of k, $O((\log n)^k) = o(n^c)$. Hence, $LSPACE \subseteq SPACE(n^c)$.

Using the space hierarchy theorem from [1] (Theorem 7.2, page 145), we know that $SPACE(n^c) \subset SPACE(n^{c+1}) \subseteq PSPACE$.

Hence, $LSPACE \subset PSPACE$. □

Theorem 4.1.3. *$NC \subseteq LSPACE \subset PSPACE$.*

Proof. The second inclusion is from the lemma proved earlier. The first inclusion is due to the reasoning used in Borodin's theorem, which was discussed in class, which implies that $NC_i \subseteq SPACE((\log n)^i)$. □

5. Question

Recall the CIRCUIT VALUE problem: The language CIRCUIT VALUE contains those pairs (C,x) where C is a Boolean circuit that evaluates to 1 on input x.

Now consider the language FORMULA VALUE, that contains those pairs (F,x), where F is a formula (a circuit with fan out 1) that evaluates to 1 on input x.

Prove that FORMULA VALUE is in the complexity class L.

5.1. Solution.

Notation 5.1.1. Let n be the size of the input. Below, when we refer to circuits, we refer to constant fan-in, single fan-out circuits whose basis are the AND, OR and NOT gates. We use the terms 'formula' and 'tree-like circuit' interchangably.

Consider the following algorithm.

Algorithm 5.1.2. ALG-FORMULA-VALUE:

- **Input**: (F,x), where F is a formula (a circuit with fan out 1) that evaluates to 1 on input x.

  F is assumed to be formed like a well bracketed, proper boolean expression. No operator precedence is assumed other than the precendence of all 'bracket' symbols compared to other operators. Example: $((a \wedge b) \vee (\sim c)) \wedge (d \wedge (e \wedge f))$.
- **Output**: Result of FORMULA VALUE problem.
- Find out the number of levels in in the tree-like circuit. Record this number, DEEPEST-LEVEL in the work tape.

  This can be done by maintaining a counter in the worktape to keep track of the level as F is parsed from left to right. The maximum level possible is n-1, so this can be done in space log n.
- While $DEEPEST - LEVEL > 0$, do the following:

  Evaluate all sub-circuits at rooted at DEEPEST-LEVEL, using information in x and SUBCIRCUIT-VALUES.

  Replace the contents of SUBCIRCUIT-VALUES with these values.

  The number of sub-circuits rooted at any given level is at most log n. So this can be done in log n space.

  Decrement DEEPEST-LEVEL.

ALG-FORMULA-VALUE correctly solves the FORMULA VALUE problem. As noted in explanations added to the various steps of ALG-FORMULA-VALUE, ALG-FORMULA-VALUE requires only log n space.

So, we have proved by construction of ALG-FORMULA-VALUE, that FORMULA VALUE is in the complexity class L.

References

[1] Christos H. Papadimitriou. *Computational Complexity*. Addison Wesley, November 1993.