Problem Set I, Petar Maymounkov
6.841J – Advanced Complexity with Prof. Madhu Sudan

1. Hierarchy review:

(a) $\text{TIME}\,(n^2) \not\subset \text{TIME}\,(\omega(n^2 \log n))$:

Let $X = x_1, x_2, \ldots$ be a lexicographic enumeration of all binary strings, and let $M_1, M_2, \ldots$ be an enumeration of all Turing Machines. For a fixed encoding choice, we can find $M_1, M_1, M_2, M_1, M_2, M_3, \ldots$ as a subsequence in X in a linear-time verifiable manner. For example, in order to ensure multiple copies M_i in the sequence, one can designate each string of the (DFA) form $\mathsf{Encode}(M_i) \cdot 1 \cdot 0^*$ to represent M_i.

We are going to build a machine M^* which runs in $O(n^2 \log n)$ time and differs from each M_i on at least one input. On input x, the machine M^* proceeds as follows:

(a) Check if x is a valid encoding of a Turing Machine. If not, output "0", otherwise let that machine be M_i

(b) Simulate M_i for exactly $|x|^2 \cdot \log^* |x|$ steps, by keeping an additional step counter whose size is $O(\log n)$ bits. Using a remark from class, we know that we can simulate a Turing Machine in real time. In addition, at each step we need to increment the counter, which takes $O(\log n)$ time steps per increment. Therefore, the overall simulation cost will be $O(n^2 \cdot \log n \cdot \log^* n)$.

If M_i halts during simulation, then complement its output, otherwise output "0".

For correctness, let M_i be any machine in $\text{TIME}\,(n^2)$ which eventually runs in time cn^2 for some constant $c > 0$. Since M_i appears infinitely often in the above sequence, eventually there will be a x_j which encodes M_i and $|x_j|^2 \cdot \log^* |x_j| \geq cn^2$. By construction M^* will differ from M_i on that input.

Finally, observe that $\log^*(\cdot)$ could have been replaced by any function that grows faster than $O(1)$.

(b) $\text{SPACE}\,(n^2) \not\subset \text{SPACE}\,(\omega(n^2))$:

The construction in this case is much the same except for the following differences. M^* simulates M_i on input x by giving it $|x|^2 \cdot \log^* |x|$ space and running it for $O(2^{|x|^2 \cdot \log^* |x|})$ time steps. If M_i does not terminate, or falls in a loop, or overflows the allocated space, then M^* simply outputs "0", otherwise it complements the output of M_i. Also note that the time step counter requires $O(|x|^2 \cdot \log^* |x|)$ bits.

Finally, by replacing $\log^*(\cdot)$ with any function that grows faster than $O(1)$ we get the required result.

2. Non-deterministic space-bounded computation:

(a) This first part is straightforward. The program goes like this. Set the "current" vertex v_c to equal s. Repeat at most $|V(G)|$ times: Guess an adjacent vertex v_a and set $v_c := v_a$. If v_c equals t then halt "YES". After $|V(G)|$ unsuccessful guesses, halt "NO".

(b) (This part is adapted from Sipser's textbook.) Begin by building an **NL** subroutine for computing the number C of vertices reachable from s. Let C_i denote the number of vertices reachable from s in at most i steps. $C_0 = 1$ and assume inductively that we have an **NL** subroutine that computes C_i. Compute C_{i+1} as follows:

- Set $C_{i+1} := C_i$
- For each $v \in V(G)$ do:
  - Set $C'_i := 0$
  - For each $v \neq u \in V(G)$ do:
    - Guess whether u reachable from s in i steps
    - If guessed yes, check that guess is correct. If guess is incorrect reject
    - Set $C'_i := C'_i + 1$
    - If $(u, v) \in E(G)$ set $C_{i+1} := C_{i+1} + 1$
  - If $C'_i \neq C_i$ reject
- Accept and output C_{i+1}

To solve co-PATH, we proceed as follows:

- Find C, using the above **NL** routine
- Set $R := 0$
- For each $v \in V(G)$, do:
  - Guess whether v is connected to s
  - If guessed yes: check that this is indeed true and increment R, otherwise reject
  - If $v = t$ then reject
- If $R \neq C$ reject
- Accept

3. Space-efficient boolean matrix multiplication and consequences:

(a) We compute each C_{ij} in turn (maintaining a counter to tell us where we are at). For each C_{ij} we check whether both a_{ik} and b_{kj} equal 1, for all $1 \leq k \leq n$ (by maintaining a counter for k). We will need another counter that helps us locate a_{ik} (or b_{kj}) given i, j and k by walking to them.

(b) Consider the recursive routine $\mathsf{Compute}(A_{ij}^k)$:

- If $k = 1$, return A_{ij}
- For every $1 \leq l \leq n$ do:

  If $\mathsf{Compute}(A_{il}^{\lceil k/2 \rceil}) = \mathsf{Compute}(A_{lj}^{\lfloor k/2 \rfloor}) = 1$, then return 1
- Return 0

The above function has bounded recursion depth $O(\log k)$. Furthermore the body of the function can be executed using a constant number of pointer/counters. Thus the total space requirement is $O(\log k \log n)$.

This construction is in the heart of Savitch's Theorem stating NSPACE $(f(n)) \subseteq$ DSPACE $(f(n)^2)$. In particular, given a machine in NSPACE $(f(n))$ the configuration space (states + tape) is a directed graph G_M on $2^{O(f(n))}$ vertices. Savitch's Theorem boils down to checking connectivity between the starting configuration an the $2^{f(n)}$ halting configurations in at most $2^{O(f(n))}$ steps. This problem is equivalent to computing $A_{ij}^{2^{f(n)}}$, where A is the adjacency matrix of G_M and (i,j) encode halting configurations. Therefore the space required by the deterministic simulation of M is $O(\log 2^{O(f(n))} \log 2^{f(n)}) = O(f(n)^2)$.

4. Ladner's general theorem:

Let $f_1, f_2, \ldots, f_i, \ldots$ be an enumeration of all binary functions $f_i : \{0,1\}^* \to \{0,1\}^*$ such that $f_i(x)$ can be computed in at most $|x|^i$ time steps.

For a polynomially-computable $h(n) : \mathbb{N} \to \mathbb{N}$ (to be specified later), define L_A and L_B as follows:

$$\begin{aligned} L_A = \big\{x \mid\ & (x = 0 \cdot w \ \wedge\ w \in L_1) \ \vee \\ & (x = 1 \cdot w \ \wedge\ h(|x|) \text{ even } \wedge\ x \in L_2)\big\} \\ L_B = \big\{x \mid\ & (x = 0 \cdot w \ \wedge\ w \in L_1) \ \vee \\ & (x = 1 \cdot w \ \wedge\ h(|x|) \text{ odd } \wedge\ x \in L_2)\big\} \end{aligned}$$

By construction, it is clear that $L_1 \leq_{\mathbf{P}} L_A, L_B \leq_{\mathbf{P}} L_2$. It remains to show that $L_A \not\leq_{\mathbf{P}} L_B$ and $L_B \not\leq_{\mathbf{P}} L_A$. These two conditions will be implied by the definition of $h(n)$.

We define $h(n)$ recursively as follows. The base case is $h(1) = 2$. In the inductive step:

- If $h(n) = 2i$, then
  - If there exists x with $|x| \leq Q(n)$ and $L_A(x) \neq L_B(f_i(x))$, then $h(n+1) = 2i+1$, otherwise $h(n+1) = h(n)$.
- If $h(n) = 2i + 1$, then
  - If there exists x with $|x| \leq Q(n)$ and $L_B(x) \neq L_A(f_i(x))$, then $h(n+1) = 2(i+1)$, otherwise $h(n+1) = h(n)$.

Note that $h(n)$ goes to infinity, because otherwise it would follow that $L_2 \leq_{\mathbf{P}} L_1$ for contradiction.

The value $Q(n)$ is chosen so that we can deterministically compute in $O(\text{poly}(n))$ time steps the following:

- A description of f_i for any $i \leq Q(n)$
- The values $L_1(x)$ and $L_2(x)$ for all x with $|x| \leq Q(n)$

By construction, it is now clear that $h(n)$ is polynomial-time computable, and $L_A \not\leq_{\mathbf{P}} L_B$ and $L_B \not\leq_{\mathbf{P}} L_A$.

5. Approximation and inapproximability:

(a) It is straightforward that ASYMMETRIC k-CENTER is in **NP**, since a solution can be easily verified using the All Pairs Shortest Paths.

The decision version of ASYMMETRIC k-CENTER would be the language containing all (G, k, d) for which G is a weighted (un)directed graph for which there exists $S \subseteq V(G)$ with $|S| \leq k$ and $\max_{x \in V(G)} \min_{y \in S} \ d_G(x, y) \leq d$. We reduce VERTEX COVER cover to this problem.

Let G be undirected. Consider G^* which is a copy of G with an additional vertex $v_{(u,w)}$ for each edge $(u, w) \in E(G)$, and additional edges $(u, v_{(u,w)})$ and $(v_{(u,w)}, w)$. It is easily seen that G has a k-VERTEX COVER iff G^* has a k-CENTER. Note that this reduction reduces VERTEX COVER to DOMINATING SET, which is the same as SYMMETRIC k-CENTER with distance 2.

(b) We show that ASYMMETRIC k-CENTER cannot be $(2 - \epsilon)$-approximated for any $\epsilon > 0$. We demonstrate how to solve DOMINATING SET using a $(2 - \epsilon)$-approximation oracle for SYMMETRIC k-CENTER.

For an unweighted, undirected graph G, let G^* be a weighted copy of G with some additional edges. In particular, for every $(u, w) \notin E(G)$, we have $(u, w) \in E(G^*)$ and $w(u, w) = 2$.

Now just note (using a basic integrality argument) that every k-center S of G^* with $\text{Obj}(S) < 2$ (found by the approximation oracle) is in fact one with $\text{Obj}(S) = 1$ and hence it is a dominating set.

(c) Let Π be a promise problem defined as follows. Π_{YES} contains all pairs (k, G) for which graph G has a k-center S with $\text{Obj}(S) \leq 1$. Π_{NO} contains all pairs (k, G) for which graph G has no k-center S with $\text{Obj}(S) \leq c$.

Let $A(k, G) \in \mathbb{R}$ be the output of the c-approximation oracle for SYMMETRIC k-CENTER on input (k, G). Then the predicate "$A(k, G) \leq c$" decides Π.