Computability and Complexity

Assignment 1

Ari Feiglin

Exercise 1.1:

Prove or disprove the following: for every search problem R, if $S_R \in \mathbf{P}$ then $R \in \mathbf{PF}$.

This is false, let us define the following search problem:

$$R = \left\{ \left((M, \omega), 1 \right) \;\middle|\; M \text{ is a Turing machine which accepts the input } \omega \right\} \cup \left\{ \left((M, \omega), 0 \right) \;\middle|\; M \text{ is a Turing machine which does not accept the input } \omega \right\}$$

Now, there exists a reduction from A_{TM} ($A_{\text{TM}} = \{(M, \omega) \mid M \text{ is a Turing machine which accepts } \omega\}$) to R: if A solves R then given an input (M, ω) we return $A(M, \omega)$. This returns 1 if M accepts ω and 0 otherwise, so it would decide A_{TM}. Thus R is not decidable, as that would contradict the undecidability of A_{TM}. And so in particular $R \notin \mathbf{PF}$.

But

$$S_R = \{(M, \omega) \mid M \text{ is a Turing machine}\}$$

which is in $\mathbf{P}$, as all we need to do is determine if given an input (M, ω), that M is a Turing machine. So $S_R \in \mathbf{P}$ but $R \notin \mathbf{PF}$.

Exercise 1.2:

We define the class $\mathbf{NP'}$ as follows: given a decision problem S, we say that $S \in \mathbf{NP'}$ if there exists a polynomial-time algorithm V such that for every x:

(1) If $x \in S$ then there exists a polynomial p and an input y whose length is at most $p(|x|)$ such that $V(x, y) = 1$.

(2) If $x \notin S$ then for every y, $V(x, y) = 0$.

Prove, disprove, or show an equivalence to an open problem that $\mathbf{NP} = \mathbf{NP'}$.

Notice that the requirement that if $x \in S$ then there exists a polynomial p and y such that $|y| \leq p(|x|)$ and $V(x, y) = 1$ can be simplified to there simply existing a y such that $V(x, y) = 1$. That is, $S \in \mathbf{NP'}$ if and only if there exists a polynomial-time algorithm V such that

(1) If $x \in S$ then there exists a y such that $V(x, y) = 1$.

(2) If $x \notin S$ then for every y, $V(x, y) = 0$.

Ie. $S \in \mathbf{NP'}$ if and only if S has a polynomial-time verifier. Obviously if $S \in \mathbf{NP'}$ then this is true, and if this is true then for every $x \in S$ we can simply define the polynomial $p = |y|$ (where y is the witness for x), then this satisfies the condition for $S \in \mathbf{NP'}$. This is because the polynomial is dependent on x, and not the other way around (ie. for $\mathbf{NP}$ there exists a polynomial which must satisfy a condition for every $x \in S$, and for $\mathbf{NP'}$ for every $x \in S$ there must exist a polynomial which satisfies some coniditon).

Now, notice that $A_{\text{TM}} \in \mathbf{NP'}$ as we can define the polynomial-time verifier V where $V\big((M, \omega), P\big) = 1$ if and only if P is a valid path of ω through M's states from the start state to its accept state. The verifier works by first verifying the validity of its input (that M is a Turing machine, and P is a path of states starting from M's starting state), and then simulate M running on ω and check that each step is precisely the next state in P. Both of these steps take polynomial time (checking that M is a Turing machine and P is a path of states takes polynomial time, and then the second step just requires iterating over P, which takes linear time). And V is a verifier for A_{TM} since $(M, \omega) \in A_{\text{TM}}$ if and only if there exists some path of states on the input ω which starts at the start state and ends at the accept state, ie if and only if there exists a path P such that $V\big((M, \omega), P\big) = 1$.

But $A_{\text{TM}} \notin \mathbf{NP}$ as if $V(x, y)$ is a polynomial proof system for V then we would be able to decide A_{TM}: Suppose that $A_{\text{TM}} \in \mathbf{NP}$ then there exists $V(x, y)$, a polynomial-time verifier for A_{TM}, and p a polynomial such that if $x \in A_{\text{TM}}$ then there exists a y where $|y| \leq p(|x|)$ and $V(x, y) = 1$. Then we can define the following algorithm:

1. **function** A_{TM}-DECIDER(M, ω)
2. **for** $\big(y \in \{0,1\}^*$ **and** $|y| \leq p\big(|(M,\omega)|\big)\big)$
3. **if** $\big(V\big((M,\omega), y\big) = 1\big)$ **return** 1
4. **end for**
5. **return** 0
6. **end function**

Then A_{TM}-DECIDER decides A_{TM}, as if it accepts (M,ω) then for some y, $V\big((M,\omega), y\big) = 1$ and since V is a verifier for A_{TM}, $(M,\omega) \in A_{\text{TM}}$. And if it doesn't accept (M,ω) then for every y whose length is less than $p\big(|(M,\omega)|\big)$, $V\big((M,\omega), y\big) = 0$. Since $V(x,y)$ is a polynomial proof system for A_{TM} whose polynomial bound on x is p, this means that $(M,\omega) \notin A_{\text{TM}}$. Thus A_{TM}-DECIDER indeed decides A_{TM}, which contradicts it being undecidable.

Thus $A_{\text{TM}} \notin \mathbf{NP}$ but $A_{\text{TM}} \in \mathbf{NP}'$, so they are not equal.

Exercise 1.3:

We define the following decision problem

$$\text{TSP} = \left\{ (X,d,D) \;\middle|\; \begin{array}{l} X \text{ is a set of targets and } d\colon X \times X \to \mathbb{N} \text{ is a distance (weight) function, and } D \text{ is} \\ \text{a natural number such that there exists a cyclic path which visits each target whose} \\ \text{total length is at most } D. \end{array} \right\}$$

We further define the search problem R_{TSP} where given (X,d) (as defined in TSP), returns a shortest possible cyclic path which visits every target in X (this is called the **travelling salesman problem**).

Define a Cook reduction from R_{TSP} to TSP.

Let us observe that in a minimal hamiltonian path (ie. a solution to R_{TSP}), an edge cannot be reused. That is, if we traverse from x to y then we cannot traverse later on from x to y again (we can traverse from y to x, though). After all, if we did then we'd have a cycle of the form $x \to y \to \cdots \to x \to y \to \cdots \to x$, let us denote the first path $y \to x$ by P_1 and then second by P_2 so this cycle has the form $x \to y \to P_1 \to x \to y \to P_2 \to x$, but we can then create a new cycle $x \to y \to P_1 \to x \to P_2^r \to y \to x$, this reduces the weight of the cycle by $d(x,y)$, and so it is shorter.

Thus, we have a maximum bound on the length of a minimal hamiltonian path,

$$M = \sum_{x \neq y \in X} d(x,y)$$

Now, we can perform a binary search between 0 and M to determine the length of the minimal hamiltonian path: we take a lower bound $\ell = 0$ and an upper bound $h = M$ and check if $\big(X, d, \frac{h+\ell}{2}\big) \in \text{TSP}$ using an oracle for TSP. If so, we set $\ell = \frac{h+\ell}{2}$ and recurse, otherwise we set $h = \frac{h+\ell}{2} - 1$ and recurse. We do this until $h \leq \ell$ and then we'd know that ℓ is the the length of the minimal hamiltonian path.

Suppose that L is the length that we computed for the minimal hamiltonian path. Now, suppose we have an edge (u,v), we can set $d(u,v) = L + 1$. If there still exists a hamiltonian cycle of length L, then there exists a hamiltonian cycle which does not use (u,v), so it is redundant and we can ignore it. If we continue this until we have iterated over every edge, we will get the minimal hamiltonian cycle. This is because the remaining edges (the edges whose distance was not set to $L+1$) must contain a hamiltonian cycle, as we only removed redundant edges, and there cannot be a remaining redundant edge as we removed all of them.

All that remains is to convert the remaining edges into the hamiltonian path. Since the set of remaining edges constitutes a hamiltonian path, all we must do is start at some node v, and follow the remaining edges (and remove them as we go) appending them to the path, until we've visited every node and returned to v.

So the steps of the algorithm are as follows:

(1) Compute L, the length of the minimum hamiltonian path via a binary search from 0 to M (the sum of all the distances).

(2) Iterate over the edges (u,v) and check if there still exists a hamiltonian cycle of length L even if we set $d(u,v) = L+1$. If so, then (u,v) is redundant and we can we can ignore it.

(3) The remaining edges define a minimal hamiltonian path, and we convert this to a path by following the remaining edges.

Suppose $\text{ORACLE}_{\text{TSP}}$ is an oracle for TSP, then the following is an algorithm for computing the minimum hamiltonian path:

1. **function** TSP-OPT(X, d)
2. $M \leftarrow \sum_{x \neq y \in X} d(x, y)$
3. $L \leftarrow$ **binary-search**$\big(0, M, i \mapsto \text{ORACLE}_{\text{TSP}}(X, d, i)\big)$
   ▷ **binary-search**(ℓ, h, f) *returns the minimum* $\ell \leq i \leq h$ *such that* $f(i) = 1$.
4. **for** $((u, v) \in X)$
5. $d \leftarrow d(u, v)$
6. $d(u, v) \leftarrow L + 1$
7. **if** $(\neg\, \text{ORACLE}_{\text{TSP}}(X, d, L))$ $d(u, v) \leftarrow d$
8. **end for**
9. $u \leftarrow$ *start* $\in X$
10. $P \leftarrow \varnothing$
11. **while** $(u \neq$ *start* **and visited**$(X) \neq 1)$
12. $v \in \{v \in N(u) \mid d(u, v) < L + 1\}$
    ▷ *Find* u*'s neighbor* v *such that* (u, v) *is not redundant.*
13. $P \leftarrow (P \to v)$
    ▷ *Append* v *to the path* P
14. $d(u, v) \leftarrow L + 1$
15. **visited**$(u) \leftarrow 1$
16. $u \leftarrow v$
17. **end while**
18. **return** P
19. **end function**

Lines 2 and 3 correspond to the first part of the algorithm, 4 to 8 correspond to the second part, and 9 to 17 correspond to the final part (converting the remaining edges to the hamiltonian path). Part 1 takes $\log(M)$ time, and since M's length is less than the length of d (or at least less than $|X| \cdot |d|$), this is polynomial with respect to the input. Part 2 takes $|X|^2$ time. And part 3 takes $|X|^2$ time (since we can bound the length of the minimum hamiltonian path by $|X|^2$). Thus this algorithm takes polynomial time, and is therefore a valid Cook reduction, as required.

Exercise 1.4:

Prove or disprove the following:

(1) For every decision problem $S \in \mathbf{P}$, there exists a decision problem S' such that there exists no Cook reduction from S to S'.

(2) For every decision problem $S \in \mathbf{P}$, there exists a decision problem S' such that there exists no Karp reduction from S to S'.

(1) This is false, suppose S' is a decision problem and $\text{ORACLE}_{S'}$ is an oracle for S'. And since $S \in \mathbf{P}$, there exists a polynomial-time algorithm A which solves S. Then we can define the following Cook reduction:

1. **function** S, S'-REDUCT(x)
2. $\text{ORACLE}_{S'}(x)$
3. **return** $A(x)$
4. **end function**

(Line 2 is really only to "use" $\text{ORACLE}_{S'}$). This is a reduction from S to S' (it obviously solves S, as A does). And so for every decision problem S', there exists a reduction from S to it.

(2) Let $S' = \varnothing$ and $S \neq \varnothing$, then for every function f, if $x \in S$ then $f(x) \notin S'$ so there cannot exists a Karp reduction from S to S'. Similarly if $S' = \{0,1\}^*$ and $S \neq \{0,1\}^*$, then for every function f, there exists an $x \notin S$ but $f(x) \in S'$, so there also cannot exists a Karp reduction here. And so this is true.

But if we ignore the edge cases, ie. if $S' \neq \varnothing, \{0,1\}^*$ then this is false. Let $y_1 \in S'$ and $y_2 \notin S'$, and let A solve S, then we can define the following function

1. **function** $f(x)$
2. **if** $(A(x) = 1)$ **return** y_1
3. **else return** y_2
4. **end function**

So if $x \in S$ then $A(x) = 1$ and so $f(x) = y_1 \in S'$. And if $x \notin S$ then $A(x) = 0$ and so $f(x) = y_2 \notin S'$. And since f runs in polynomial time, as A does, f is a Karp reduction from S to S'.

So if we take the statement as-is, then it is true. But if we require that S' be non-trivial (empty or $\{0,1\}^*$) then it is false.