

Homework 9

Due: May 4, 2005

Vinod Vaikuntanathan

Readings: Sipser, Section 7.5. Also (optionally) see Garey and Johnson's book, "Computers and Intractability: a Guide to NP-Completeness".

Problem 1: (Sipser 7.20) Let G represent an undirected graph and let

$$\text{SPATH} = \{\langle G, a, b, k \rangle \mid G \text{ contains a simple path of length at most } k \text{ from } a \text{ to } b\}$$

and

$$\text{LPATH} = \{\langle G, a, b, k \rangle \mid G \text{ contains a simple path of length at least } k \text{ from } a \text{ to } b\}$$

1. Show that $\text{SPATH} \in \text{P}$.
2. Show that LPATH is NP-complete. You may assume the NP-completeness of UHAMPATH, the Hamiltonian path problem for undirected graphs.

Solution 1:

1. $\text{SPATH} \in \text{P}$. Breadth-first-search through the graph, starting at a until you reach node b , check that the shortest path from a to b is less than or equal to k . If so, accept. Else, reject. Using Dijkstra's single source algorithm, you can do this in $O(n^2)$, for a graph with n nodes.
2. LPATH is NP-complete. First, LPATH is in NP. Let the certificate be a sequence of edges from a to b of length at least k .

Next, we must show that LPATH is NP-hard; we do so by showing $\text{UHAMPATH} \leq_P \text{LPATH}$. We map $\langle G, s, t \rangle \rightarrow \langle G', a, b, k \rangle$. Both graphs are undirected. Here, we set $G' = G$, $a = s$, $b = t$, and we set k to be the number of nodes in G minus 1. A simple path, by definition, has no repeated nodes in it. Thus, the only way to have a path from a to b in G' of length at least $n - 1$ is to visit each node only once – i.e., a Hamiltonian path; and vice versa.

Problem 2: For a cnf-formula ϕ with m variables and c clauses, show that you can construct in polynomial time an NFA with $O(cm)$ states that accepts all non-satisfying assignments, represented as Boolean strings of length m . Conclude that the problem of minimizing NFAs cannot be done in polynomial time unless $P = NP$.

Solution 2: If the given assignment is non-satisfying, at least one of the clauses of ϕ is evaluated to true under this assignment. This already has the flavor of the NFA accepting rule: the input to an NFA is *accepted* (which happens when it is a non-satisfying assignment), then there is *at least one path* from the start state q_{init} to the accept state q_{acc} in the NFA. Let's see how to formalize this intuition.

We will construct an NFA N which accepts exactly the non-satisfying assignments of a given formula ϕ . If the input x to the NFA is shorter/longer than ℓ , (where ℓ is the number of variables in ϕ) then x is considered a non-satisfying assignment. (This is a convention that is necessary to prove the second part of the problem).

The intuition of the construction is to let the NFA have multiple parallel paths – one corresponding to every clause. The string corresponding to the assignment will run through multiple paths of

the NFA, in parallel. The string reaches the end of the path *iff* it *does not* satisfy the corresponding clause. Therefore, it follows that a string reaches the end of the path iff it is a non-satisfying assignment for the formula. The rest of the jugglery (as in the full proof below) is to ensure that the NFA accepts all the strings of length not equal to ℓ . (As noted, we will need this, in order to prove the second part of the problem.)

The full proof:

Suppose ϕ has ℓ variables x_1, x_2, \dots, x_ℓ and k clauses C_1, C_2, \dots, C_k . Then, we create an NFA $N = (Q, \Sigma, \delta, q_{init}, F)$, where

- $Q = \{q_{init}\} \cup \{q_{acc}\} \cup \{q_{ij} \mid 1 \leq i \leq k, 1 \leq j \leq \ell\} \cup \{L_j \mid 1 \leq j \leq \ell + 1\} \cup \{S_j \mid 1 \leq j \leq \ell\}$.
That is, there is a state for every clause and every variable. There is a start state and an accepting state. Then, there are ℓ states of the form S_j and $\ell + 1$ states of the form L_j . These are dummy states: we will see where we use these dummy states shortly.
- $\Sigma = \{0, 1\}$, because assignments (which are inputs to the NFA) are composed of true/false values.
- q_{init} is the start state.
- $F = \{q_{acc}\}$.
- δ is defined as follows:

Note: It might be helpful to refer to the figure on the next page, for what follows. Figure ?? is a diagrammatic representation of the NFA we construct for the formula $(x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \bar{x}_3)$. The assignment $(0, 0, 0)$ unsatisfies the first clause, and this is clearly accepted by the NFA. Whereas the assignment $(1, 1, 1)$ satisfies all the clauses and is not accepted by the NFA. The strings 1111 as well as 00 are not satisfying assignments to ϕ , by convention, and therefore, are accepted by the NFA. They are accepted by virtue of the “dummy paths” we construct using the nodes L_j and S_j .

- (The start state connects by an ϵ -arrow to a set of nodes, one corresponding to every clause, and the two nodes corresponding to the dummy paths.)

$$\delta(q_{init}, \epsilon) = \{q_{11}, q_{21}, \dots, q_{k1}, L_1, S_1\}.$$

- (This specifies the connections among the nodes L_j . Intuitively, the dummy nodes are there, because you want to accept strings longer than ℓ . Remember: By convention, strings longer than ℓ do not constitute satisfying assignments to ϕ .)

$$\text{For all } 1 \leq j \leq \ell, \delta(L_j, 0) = L_{j+1} \text{ and } \delta(L_j, 1) = L_{j+1}.$$

$$\text{Also, } \delta(L_{\ell+1}, 0) = \{L_{\ell+1}, q_{acc}\} \text{ and } \delta(L_{\ell+1}, 1) = \{L_{\ell+1}, q_{acc}\}.$$

Note: convince yourself that this constitutes an accepting path for any string longer than ℓ and for no shorter string.

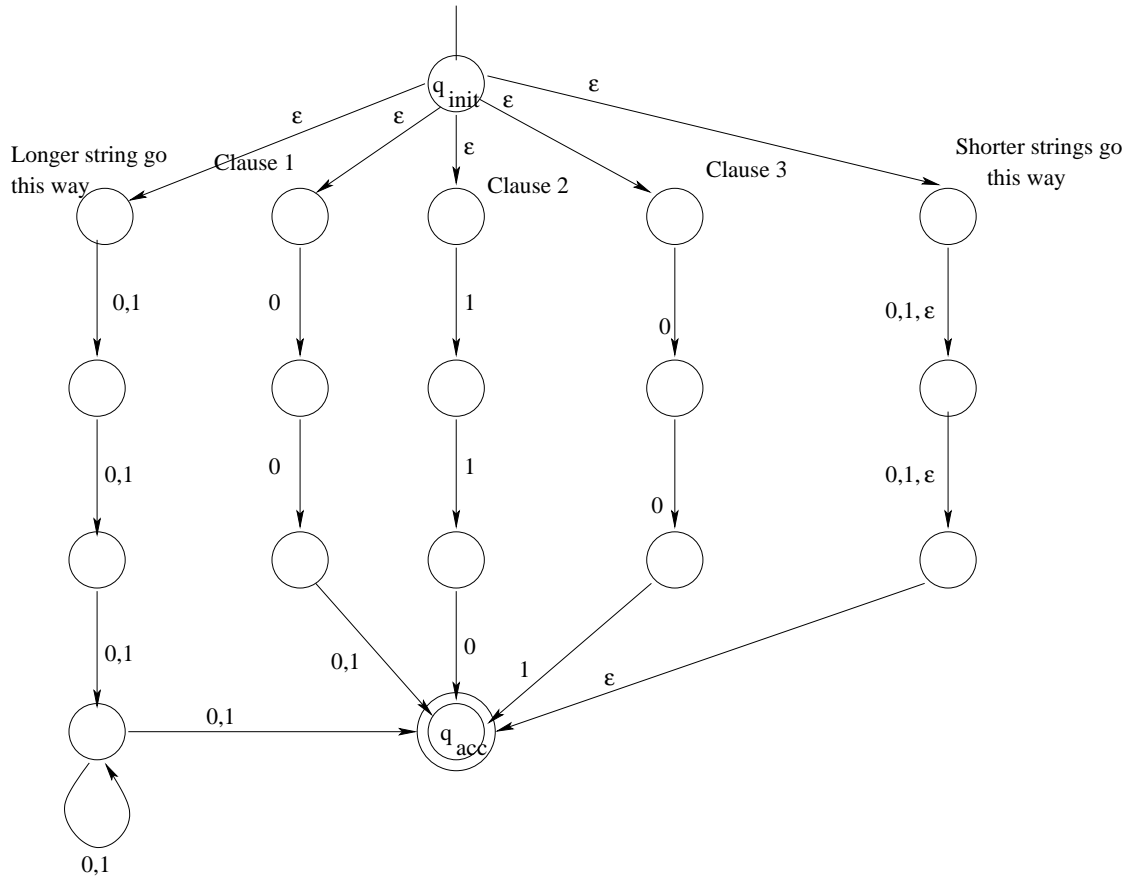
- (This specifies connections among the nodes S_j . These nodes are there because we want to accept strings that are strictly shorter than ℓ . Note, again, that such strings are non-satisfying assignments of ϕ .)

$$\text{For all } 1 \leq j < \ell, \delta(S_j, 0) = \{S_{j+1}\} \text{ and } \delta(S_j, 1) = \{S_{j+1}\} \text{ and } \delta(S_j, \epsilon) = \{S_{j+1}\}.$$

$$\text{Also, } \delta(S_\ell, \epsilon) = \{q_{acc}\}.$$

- (This specifies the connections among the clause nodes. These connections ensure that a string of length ℓ is accepted if and only if it is a non-satisfying assignment for ϕ .)

$$\text{For } j < k,$$



- * If clause C_i contains variable \bar{x}_j , then $\delta(q_{i,j}, 1) = \{q_{i,j+1}\}$
- * If clause C_i contains variable x_j , then $\delta(q_{i,j}, 0) = \{q_{i,j+1}\}$
- * If clause C_i contains neither variable x_j nor \bar{x}_j , then $\delta(q_{i,j}, 0) = \{q_{i,j+1}\}$ and $\delta(q_{i,j}, 1) = \{q_{i,j+1}\}$.

– (This specifies the connections from the nodes at the end of each clause path to the accepting state. They are of the same flavour as the previous δ -rules.)

For $j = k$,

- * If clause C_i contains variable \bar{x}_k , then $\delta(q_{i,k}, 1) = \{q_{acc}\}$
- * If clause C_i contains variable x_k , then $\delta(q_{i,k}, 0) = \{q_{acc}\}$
- * If clause C_i contains neither variable x_k nor \bar{x}_k , then $\delta(q_{i,k}, 0) = \{q_{acc}\}$ and $\delta(q_{i,k}, 1) = \{q_{acc}\}$.

If the input string is shorter than ℓ , then going through the path consisting of the S -nodes leads to the accept state. If the input string is longer than ℓ , then there is a path that passes through only the L -nodes that leads to the accept state. If A is a non-satisfying assignment of length ℓ for ϕ , then A unsatisfies at least one clause C_j . To unsatisfy C_j , it has to unsatisfy *all* the variables in C_j . By construction, if this happens, then there is a path (corresponding to clause C_j) that leads from q_{init} to q_{acc} .

On the other hand, if A is a satisfying assignment, then the NFA cannot accept by virtue of reaching q_{acc} through paths that pass through L -nodes or S -nodes, since only strings of length

different from ℓ can go to q_{acc} through such nodes. (Note that there is no self-loop in q_{acc} , so if you reach q_{acc} and there are more input bits to read, then the NFA rejects). When the string is of length exactly ℓ and is an accepting assignment, it is easy to check that there is no accepting path from q_{init} to q_{acc} (by virtue of our construction). Thus, N accepts exactly the non-satisfying assignments.

Now, we reduce the problem of solving SAT to minimizing NFAs. Thus, it follows that, if there is an algorithm that minimizes NFAs, then we can solve SAT, and hence $P = NP$. What does the reduction look like? Given the previous part of our solution, we do the following:

- From the given formula ϕ , construct an NFA N which accepts exactly the non-satisfying assignments of ϕ .
- Now, run the algorithm that minimizes N . Let the result of the minimization be N' .
- Check if N' is the trivial one-state NFA that accepts all the strings. If that is the case, then we know that the formula ϕ has no satisfying assignments. Else, we know that ϕ has at least one satisfying assignment, since the NFA rejects at least one string, and the rejected string corresponds to a satisfying assignment.

Problem 3: An edge-cover in a graph $G(V, E)$ is a set of edges $E' \subseteq E$ of G such that each vertex in G is the end-point of at least one of the edges in E' . As a language,

EDGE-COVER = $\{(G, k) \mid G \text{ is an undirected graph that has an edge-cover with at most } k \text{ edges}\}$.

Show that EDGE-COVER $\in P$. (Recall the problem VERTEX-COVER that we proved NP-complete in the class.)

(Hint: You can assume, without proof, that the problem of finding a maximum matching in a graph is in P . In a graph $G = (V, E)$, a matching is a set $E' \subseteq E$ of edges, such that no two edges in E' are adjacent to the same vertex. A maximum matching in G is such a set E' of the largest cardinality. Use a maximum matching to construct a minimum edge-cover, and prove that your construction works, and runs in polynomial time.)

Solution 3: Following the hint, assume that there is a polynomial time algorithm to find a maximum matching M in G . (Recall that M is a largest set of edges such that no two edges in M are adjacent to the same vertex). Construct an edge-cover C as follows: Initially, $C \leftarrow M$. For every vertex v such that v is not the end-point of any edge in M , find an edge e such that e is adjacent to v , and add e to C . i.e, $C \leftarrow C \cup \{e\}$. C is clearly an edge-cover, and its size is $|C| = |M| + (|V| - 2|M|) = |V| - |M|$.

We claim that the C that results is a minimum edge-cover. Suppose there is an edge-cover C' smaller than C . Then, a matching M' can be formed from C' as follows: Find a maximal set of disjoint edges in C' . Output this set as M' . We claim that M' will be bigger than M . This clearly contradicts the assumption that M was a biggest matching in G . But, why would $|M'|$ be larger than $|M|$?

- C' covers all the vertices in G . That is, every vertex in G is the endpoint of one of the edges in C' . This is true, since C' is an edge-cover.
- The edges used to form M' cover at most $2|M'|$ vertices. There are $|V| - 2|M'|$ vertices, that have to be covered by the remaining $|C'| - |M'|$ edges in C' . (the ones left after forming M')
- None of the remaining edges (in $C' - M'$) can be used to cover two of the remaining vertices simultaneously. If this happens, we can add another edge to M' , violating the assumption that M' is maximal.

- Therefore, to cover $|V| - 2|M'|$ remaining vertices, we need at least $|V| - 2|M'|$ edges. We have $|C'| - |M'|$ of them left. Therefore, $|C'| - |M'| \geq |V| - 2|M'|$. $|M'| \geq |V| - |C'|$. Since $|C'| < |C|$ (by assumption), $|M'| > |M|$, violating the assumption that M is a maximum matching.

Problem 4: Suppose there exists a family of bijections $\{f_k\}_{k=1}^{\infty}$ such that f_k maps integers of length k onto integers of length k . We also know that

- For all k , f_k is computable in polynomial time (in k), and
- No f_k^{-1} is computable in polynomial time.

Prove that this would imply that the language

$$A = \{\langle x, y \rangle \mid f^{-1}(x) < y\}$$

is in $(NP \cap coNP) - P$.

Solution 4: We would need to prove that:

- $A \in NP$,
- $A \in coNP$, and
- If $A \in P$, then every f_k^{-1} is computable in polynomial time.

To show the first, we note that there is a short certificate we can give to prove that $f^{-1}(x) < y$, given an $\langle x, y \rangle$ pair. The certificate is $a = f^{-1}(x)$, which is unique because f is a permutation. How does one verify it? Compute $f(a)$, check that $f(a) = x$ and that $a < y$. The certificate is short – it is k bits long. The verification is efficient, *since f can be computed in polynomial time*.

Note: It is crucial that every f_k is computable in time $poly(k)$. Otherwise, we would not have an efficient verifier, and thus cannot prove that $A \in NP$.

To show that $A \in coNP$, we need to show that $\bar{A} \in NP$. What is \bar{A} ?

$$\bar{A} = \{\langle x, y \rangle \mid |x| \neq |y| \text{ or } f^{-1}(x) \geq y\}.$$

The witness that shows that $\langle x, y \rangle \in \bar{A}$ is again, $a = f^{-1}(x)$. The verifier does the following:

- If $|x| \neq |y|$, then accept.
- If $f(a) = x$ and $a \geq y$, then accept.
- Otherwise (if neither of these hold), reject.

It is easy to see that if $\langle x, y \rangle \in \bar{A}$, then this certificate actually causes the verifier to accept. On the other hand, if $\langle x, y \rangle \notin \bar{A}$, then there is no way to make the verifier accept. Thus, $\bar{A} \in NP$, and $A \in coNP$.

To show that A is not in P (assuming f_k^{-1} cannot be computed efficiently): Suppose $A \in P$, for the sake of contradiction. Then, we can binary search to invert an arbitrary element x as follows:

- (Binary Search) Set the variables $low = 1$, $high = 2^k - 1$
- Set $current = \lfloor (low + high) / 2 \rfloor$.
- For $i = 1$ to $k - 1$, do:
 - determine if $\langle x, current \rangle \in A$. (Note: This can be done in polytime by assumption.)

- If so, set $high := current$, else set $low := current$.
- If $low = high - 1$, output low as the inverse $f_k^{-1}(x)$. Else, go to Step 2.

Grading Guidelines:

Problem	Points/Guidelines
1	3 points – 1.5 point for every part.
2	2+1 = 3 points – We dont expect a completely formal proof. In particular, an answer that doesnt take care of the issue of handling longer/shorter strings should get 1.5 points (plus the 1 point, if the second part is right).
3	2 points – 1 point for the algorithm, 1 for the proof.
4	0.5 + 0.5 + 1 points – 0.5 points each to show that A is in NP, and coNP. 1 point to show that it is in P .