Computational Complexity - Homework I (Solutions)

K. Subramani
LCSEE,
West Virginia University,
Morgantown, WV
{ksmani@csee.wvu.edu}

1 Problems

1. Assume that you are given an instance of the Traveling Salesman Problem (TSP) with n cities and inter-city distances d_{ij}, $i, j = 1, 2, \ldots n$. Let S denote some subset of the cities, excluding city 1 and let $C[S, j]$ denote the shortest path that starts in city 1, visits all the cities in S and ends in city j.

   (a) Use Dynamic Programming to compute $C[S, j]$, i.e., in computing $C[S, j]$ for a given S, use the values computed for subsets of S.

   (b) Analyze the space and time requirements of your algorithm.

   (c) Modify this algorithm to devise a *polynomial time* algorithm for the problem of computing the shortest path from city 1 to city n; note that this shortest path need not visit all the other cities.

   Solution:

   (a) Focus on the vertex $i \in S$, which is connected to vertex j in the path determined by $C[S, j]$. Regardless of how i is chosen, we must have $C[S - \{i\}, i]$ as the shortest path that starts from vertex 1, visits all the vertices in $C[S - \{i\}]$ and ends at vertex i. Accordingly, the dynamic programming solution is:

   $$C[S, j] = \min_{i \in S}(C[S - \{i\}, i] + d_{ij})$$

   The entry of interest is $C[\{2, 3, \ldots, n-1\}, 1]$.

   (b) Given that the size of S varies from 0 to $(n-1)$ and for a specific size k, there are $\binom{n-1}{k}$ distinct subsets, the total number of subsets generated is $\sum_{k=0}^{n-1}\binom{n-1}{k}$. For a set S of size k, we need to compute $C[S, j]$ for all $(n-1-k)$ possibilities of j; each of these computations takes k steps. Thus, the total time required is:

   $$\sum_{k=0}^{n-1}\binom{n-1}{k} \cdot k \cdot (n-1-k) \in O(n^2 \cdot 2^n)$$

   Space requirements can be computed in similar fashion by noting that all combinations of S and j have to be stored resulting in space requirements of

   $$\sum_{k=0}^{n-1}\binom{n-1}{k} \cdot (n-1-k) \in O(n \cdot 2^n)$$

   (c) Let p_{ij}^k denote the shortest path from vertex i to vertex j, with intermediate vertices drawn only from the set $S_k = \{1, 2, \ldots, k\}$. It is straightforward to see that either p_{ij}^k is equal to p_{ij}^{k-1} (if vertex k does not lie on the shortest path from i to j with all the intermediate vertices in S_k) or $p_{ij}^k = p_{ik}^{k-1} + p_{kj}^{k-1}$ (if vertex k does lie on the shortest path from i to j with all the intermediate vertices in S_k). Accordingly,

$$\begin{aligned} p_{ij}^k &= \min(p_{ij}^{k-1}, p_{ik}^{k-1} + p_{kj}^{k-1}), \text{ if } k \geq 1 \\ &= d_{ij}, \text{ if } k = 0. \end{aligned}$$

It is not hard to see that if the entries are computed in bottom-up fashion, each p_{ij}^k entry can be computed in constant time. There are n^2 such entries that have to be computed for each $k = 0, 1, \ldots, n-1$; thus the running time of the corresponding algorithm is $O(n^3)$. Note that in the absence of negative cost cycles, the entry of interest is d_{1n}^{n-1}, since the shortest path from 1 to n cannot have vertex n as an intermediate vertex. The shortest path problem is not defined in the presence of negative cost cycles. For additional information, please see the description of the Floyd-Warshall algorithm in [CLRS01].

□

2. Argue that if a Turing Machine uses less than $c \log \log n$ space, for all $c > 0$, then it uses constant space.

Solution: The proof of the assertion in this problem requires the development of two concepts, viz., semi-configurations and crossing sequences.

Assume that our model of computation is a k-string Turing machine with input and output denoted by $M = (Q_M, \Sigma_M, \delta_M, s)$, where the symbols have their usual meaning.

Definition 1.1 (Semi-Configuration) *A semi-configuration of the Turing machine M, that runs in s(n) space, consists of:*

(i) Its current state ($|Q_M|$ possibilities),

(ii) The symbol currently being read on the input tape ($|\Sigma_M|$ possibilities),

(iii) The contents of the work tape ($|\Sigma_M|^{s(n)}$ possibilities), and

(iv) The position of the work head on the work tape ($s(n)$ possibilities).

Observe that M has at most $N = |Q_M| \cdot |\Sigma_M|^{s(n)+1} \cdot s(n) = 2^{O(s(n))}$ possible semi-configurations.

Definition 1.2 *A semi-configuration if a Turing Machine M is* right-moving, *if M moves its input head to the right on this semi-configuration, and* left-moving, *otherwise.*

Definition 1.3 (Crossing Sequence) *Let x be an input accepted by M, and let i be a position in x. The crossing sequence at position i, $C^i(x)$, is defined to be the ordered sequence of semi-configurations of M whenever the input head is on the ith position of the input tape.*

Note that $|C^i(x)| \leq N$, since M would not halt on x otherwise.

Theorem 1.1 *If M runs in $s(n) = o(\log\log(n))$ space, then it has at most $o(n)$ crossing sequences.*

Proof: As each crossing sequence has length at most N, there are at most

$$\sum_{j=0}^{N} N^j = \frac{N^{N+1} - 1}{N - 1} = O(N^N) = O\left(\left(2^{O(s(n))}\right)^{2^{O(s(n))}}\right) = O\left(2^{2^{O(s(n))}}\right) = o(n)$$

crossing sequences in all, since $s(n) = o(\log \log n)$. □

We have therefore established that M has at most $o(n)$ crossing sequences. Thus, there exists an integer n_0, such that for all $n > n_0$, there are less than $\frac{n}{3}$ possible semi-configurations on inputs of size n (by the definition of $o()$!).

Let $n_1 \geq n_0$ be such that $s(n) < s(n_1)$ for all $n < n_1$. If such an n_1 does not exist, then the function s is bounded above by a constant and we are done! To see this, assume that no such n_1 exists and $s(n)$ is not bounded by a constant. Then for every constant c, there exists an integer $n_c > 0$, such that n_c is the smallest positive integer for which $s(n_c) > c$. Thus, for any $0 < n < n_c$, we have, $s(n) \leq c < s(n_c)$. It follows that n_c satisfies the qualities of the n_1 we are looking for, i.e., if no such n_1 exists, then $s(n)$ is bound by a constant.

We will now show that $s(n_1)$ is an upper bound for $s(n)$, regardless of the value of n, i.e., regardless of the length of the input string.

Assume the contrary. Let x be a string of minimum length such that:

(a) M accepts x,

(b) $|x| \geq n_1$, and

(c) $M(x)$ uses more than $s(n_1)$ space.

Let $|x| = n > n_0$. As $|x| > n_0$, for any $0 < i \leq |x|$, the number of possible crossing sequences on x at position i is less than $n/3$. Thus, by the pigeon hole principle, there exist positions $0 < i < j < k \leq n$ such that $C^i(x) = C^j(x) = C^k(x)$.

Let $x = \alpha a \beta a \gamma a \delta$ with the a's at positions i, j, and k, where α, β and γ are strings. (Why must each position have the same character a?) Also let $C^i(x) = C^j(x) = C_1^i \ldots C_l^i$, with C_1^i though C_l^i representing the semi-configurations in $C^i(x)$. We will now traverse though the computation of M on input $x' = \alpha a \gamma a \delta$.

The executions of $M(x)$ and $M(x')$ are identical until they come to the first right-moving semi-configuration in the sequence $C^i(x)$, say $C^i_{r_1}$. The execution of $M(x')$ after this point is now identical to the execution of $M(x)$ beginning at $C^j_{r_1}$ until $M(x')$ comes to the next left-moving semi-configuration following $C^j_{r_1}$, say $C^j_{r_2}$. The execution of $M(x')$ after this point is now identical to the execution of $M(x)$ beginning at $C^i_{r_2}$, and so on. Matters continue in this vein, until $M(x')$ reaches semi-configuration $C^i_l = C^j_l$ (and this has to happen, since $r_1 < r_2 < \ldots$). From this point onwards, the execution of $M(x')$ will be identical to the execution of $M(x)$ and hence M accepts x'.

A similar argument shows that M accepts the string x'', where we define $x'' = \alpha a \beta a \delta$. Note that both x' and x'' are strictly shorter than x and accepted by M.

Let s_w denote the maximum number of work cells (peak workspace) used by M on input the string w. If s_x work cells are used by $M(x)$ when its input head is within the substring αa or the substring δ, then $s_{x'}, s_{x''} \geq s_x$. (Why?) If s_x work cells are used by $M(x)$ when its input head is within the substring γa then $s_{x'} \geq s_x$. (Why?) Similarly, if s_x work cells are used by $M(x)$ when its input head is within the substring βa then $s_{x''} \geq s_x$. If less than s_x cells are used by $M(x)$ in all three cases, then $M(x)$ uses less than s_x space!

From the above discussion, it follows that either $s_{x'} \geq s_x \geq s(n_1)$ or $s_{x''} \geq s_x \geq s(n_1)$. Without loss of generality, assume the former. In this case, we must have $|x'| \geq n_1$. (Why?) In other words, x' satisfies the following three properties:

(a) M accepts x',

(b) $|x'| \geq n_1$, and

(c) $M(x)$ uses more than $s(n_1)$ space.

Further, since $|x'| < |x|$, we have a contradiction to our hypothesis that x is the *shortest* string satisfying these properties.

There exist a number of approaches for this problem; the method discussed above is detailed in [Kat] and [Gol08]. A similar argument is outlined in [Koz06]. For a completely different approach, see [Sze94]. □

3. Assume that you have a k-string NDTM (Non-deterministic Turing Machine) that accepts a language L in time $f(n)$. Show that L can accepted by a 2-string NDTM in time $O(f(n))$.

Solution: If $k \leq 2$, the problem is trivial, so assume $k > 2$. The key observation is that a single step of the k-string NDTM (T_1) can be simulated in k steps of the 2-string NDTM (T_2). Note that a single move of T_1 involves moves on all k strings. T_2 simply guesses k triplets, which correspond to the moves made on all k strings by T_1 on its first tape. It then uses the second tape to verify the guesses. Since T_1 accepts L in $f(n)$ steps, T_2 accepts L in $k \cdot f(n) = O(f(n))$ steps. $\Box$

4. Classify each of the following languages (with appropriate justification) as recursive, recursively enumerable (but not recursive), or not recursively enumerable.

   (a) $L = \{\langle M \rangle : M \text{ halts on the empty string}\}$.

   (b) $L = \{\langle M \rangle : M \text{ halts on at least one string}\}$.

   (c) $L = \{\langle M,\ M' \rangle : L(M) = L(M')\}$.

   Solution:

   (a) This language is not recursive, however it is recursively enumerable. To see this, assume the contrary and let T denote a Turing machine that decides L. Let N denote an arbitrary Turing machine and let x be an input to N. We construct the Turing machine N' defined as follows:

   $$N'(y) \quad = \quad N(x),\ \forall\, y$$

   Observe what happens when $\langle N' \rangle$ is provided as input to T. $T(N')$ will accept if and only if N halts on x. In other words, T can be used to solve the Halting Problem, i.e., L is not recursive. However L is recursively enumerable as the operation of $M(\epsilon)$ can be simulated.

   (b) This language is not recursive, however it is recursively enumerable. To see this, assume the contrary and let T denote a Turing machine that decides L. Let N denote an arbitrary Turing machine and let x be an input to N. We construct the Turing machine N' defined as follows:

   $$N'(y) \quad = \quad N(x),\ \forall\, y$$

   Observe what happens when $\langle N' \rangle$ is provided as input to T. $T(N')$ will accept if and only if N' halts on some input. Let z be such an input. By the definition of N' we have that $N'(z) = N(x)$ which means that $T(N')$ will accept if and only if N halts on x. In other words, T can be used to solve the Halting Problem, i.e., L is not recursive. However L is recursively enumerable as the operation of $M(x)$ can be simulated over all x though dovetailing.

   (c) This language is not recursively enumerable. To see this, assume the contrary and let T denote a Turing machine that accepts L. Let N denote a Turing machine that does not halt on any input. Let P denote an arbitrary Turing machine and let x be an input to P. We construct the Turing machine P' defined as follows:

   $$\begin{aligned} P'(y) \quad &= \quad P(x),\ \text{if } y = x \\ &= \quad \nearrow,\ \text{otherwise} \end{aligned}$$

   Observe what happens when $\langle P', N \rangle$ is provided as input to T. $T(P', N)$ will halt and accept if and only if P' *does not* halt on x. In other words, T can be used to solve the complement of the Halting Problem, i.e., L is not even recursively enumerable, much less recursive. $\Box$

5. Let S be an infinite set of boolean expressions, such that every finite subset of S is satisfiable. Argue that S itself must be satisfiable. i.e., the conjunction of all the expressions in S is satisfiable.

   Solution: We are being asked to prove the Compactness theorem for boolean logic. The arguments used to establish the Compactness theorem for first-order logic clearly suffice!

   Note that in case of boolean logic, a model is merely an assignment of $\{$**true, false**$\}$ to the variables of the expression. If S is unsatisfiable, then any assignment that satisfies S also satisfies $\phi \wedge \neg\phi$, where ϕ is an arbitrary boolean expression. But by the completeness theorem of first-order logic (and hence boolean logic) there must exist a proof of $\phi \wedge \neg\phi$ from S. By the definition of proofs, this proof must have finite length. This implies that a finite subset of S gives rise to a contradiction, which contradicts the hypothesis.

   A second approach is discussed on Page 85 of [Pap94]. $\Box$

References

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[Gol08] Oded Goldreich. *Computational Complexity*. Cambridge University Press, 2008.

[Kat] Jonathan Katz. Lecture Notes on Complexity.

[Koz06] Dexter Kozen. *Theory of Computation*. Springer, 2006.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

[Sze94] Andrzej Szepietowski. *Turing Machiens with Sublogarithmic Space*. Number 843 in Lecture Notes in Computer Science. Springer, 1994.