Computability and Computational Complexity, A.Y. 2019–2020

Solution outlines to the written exam

Mauro Brunato

Wednesday, January 15, 2020

Exercise 1

Consider the UNIVERSITY HIRING decision problem:

> A university needs to hire the teaching staff for a new degree, for which a set T of topics must be taught. The executive board received n applications from prospective teachers, and every applicant $i \in \{1, \dots, n\}$ has knowledge of a subset $S_i \subseteq T$ of the required topics. The budget allows for hiring at most $k \leq n$ teachers. Is there a choice of k applicants so that all teaching topics are covered?

An instance of the problem consists of parameters $n, k, T, S_1, \dots, S_n$.

1.1) Prove that UNIVERSITY HIRING $\in$ **NP**.

1.2) Prove by reduction that UNIVERSITY HIRING is complete in the class **NP**. The reduction can refer to any language discussed during the course.

1.3) Prove that if k is kept constant (e.g., $k = 10$), then the problem's asymptotic complexity is polynomial wrt input size.

Solution 1

1.1) Proving that an instance has positive answer only requires to provide the k indexes $(i_1, \dots, i_k)$ of the hired professors. The certificate is clearly polynomial wrt the input size (actually, much smaller). To verify that the certificate is valid, we just need to check (i) that the certificate contains (no more than) k numbers; (i) that all such numbers are different and between 1 and n; (iii) that every element in T apears in at least one of the topic subsets $S_{i_1}, \dots, S_{i_n}$.

1.2) UNIVERSITY HIRING is just a rephrasing of SUBBSET COVER. Formally, given an instance of SUBSET COVER, we map the union of all sets to T, and each of the sets to a different S_i.

1.3) If k is constant, rather than being a parameter in the instance, then the naïf algorithm that consists of generating and checking all possible certificates must iterate through

$$\binom{n}{k} = O(n^k)$$

different certificates, each of which can be checked in polynomial time.

Observations

- Beware of the direction of the reduction! You have to start from a *generic* instance of a known **NP**-complete problem and map it to a UNIVERSITY HIRING instance, not vice versa. Otherwise, you are only proving that your particular idea is not feasible, but you are not ruling out all possible methods. E.g., by reducing UNIVERSITY HIRING to SAT you just prove that using SAT is a bad idea, but you are not excluding the possibility that there are other, better reductions to problems in **P**!

- It was also possible to start from other known **NP**-complete problems. For example, starting from VERTEX COVER and then mimicking the reduction proposed in the course to SUBSET COVER.

$\boxed{\textbf{Exercise 2}}$

2.1) When is a language recursive? When is it recursively enumerable?

2.2) Prove that the following property $\mathcal{P}$ of Turing machines $\mathcal{M}$ is not recursive:

$$\mathcal{P} = \{\mathcal{M} : \mathcal{M}(\varepsilon) \text{ halts after an even number of steps}\}$$

where ε is the empty input string.

2.3) Prove that $\mathcal{P}$ is recursively enumerable.

Hint — *Point 2.3 can be proved by explicitly outlining an enumeration algorithm.*

$\boxed{\textbf{Solution 2}}$

2.1) See the definitions in the literature

2.2) The property is not semantic, therefore Rice's Theorem cannot be invoked.
Since we are dealing with machines running on an empty input, let us reduce the halting problem on empty input (that we called HALT_ε) to property $\mathcal{P}$, and suppose that. Clearly, $\mathcal{P}(\mathcal{M}) \Rightarrow \text{HALT}_\varepsilon(\mathcal{M})$. On the other hand, if $\mathcal{P}(\mathcal{M})$ is false, $\mathcal{M}$ might still be halting in an odd number of steps. In this case, let us add one dummy step to machine $\mathcal{M}$ before the halting state. Formally:

$$\text{HALT}_\varepsilon(\mathcal{M}) \quad \Leftrightarrow \quad \mathcal{P}(\mathcal{M}) \vee \mathcal{P}(\mathcal{M} + \text{dummy step before halting}).$$

Otherwise, consider any machine $\mathcal{M}$: if it halts, then the machine $\mathcal{M} + \mathcal{M}$ (consisting of two executions of $\mathcal{M}$) halts in an even number of steps:

$$\text{HALT}_\varepsilon(\mathcal{M}) \quad \Leftrightarrow \quad \mathcal{P}(\mathcal{M} + \mathcal{M}).$$

2.3) Consider a UTM that simulates a machine $\mathcal{M}$ and counts the number of steps. If the simulation halts, then $\mathcal{U}$ accepts $\mathcal{M}$ iff the number of steps was even. This machine satisfies the definition of recursive enumerability.
For an explicit enumeration algorithm, let us just employ the usual diagonalization method, but before writing out a halting machine we additionally check whether it halted in an even number of steps.

Observations

- Saying that a language is recursive if "we" can decide it is not the best way to formulate a definition, since "we" (or "you," or "I") is not a well defined computational model.
- Again, the direction of the reduction is fundamental. Reducing $\mathcal{P}$ to HALT proves nothing, we need to reduce HALT to $\mathcal{P}$, i.e., assume that $\mathcal{P}$ is decidable and try to solve HALT with it.
- For the reason above, just saying "in order to decide $\mathcal{P}(\mathcal{M})$ we would need to know if $\mathcal{M}$ halts, but this is not possible" doesn't prove anything, because you are only ruling out methods that use HALT to decide $\mathcal{P}$; however, that line of reasoning is not excluding other possible ways of deciding $\mathcal{P}$ that don't use HALT at all.

Exercise 3

Consider the following language:

$$S = \left\{ \left(x \in \{0,1\}^*, k \in \mathbb{N}\right) \,:\, x \text{ contains a subsequence of } k \text{ adjacent 0's} \right\}.$$

For example, $(101000101, 3) \in S$ because the binary string contains 3 consecutive zeroes, while $(101000101, 4) \notin S$ because the binary string does not contain 4 consecutive zeroes.

3.1) Prove that $S \in$ **P**.

3.2) Prove that $S \in$ **L**.

Hint — *Again, both points can be proved by describing an algorithm and showing that it has the required property.*

Solution 3

Consider an algorithm that scans the input sequence x and uses a counter to keep track of the number of consecutive zeroes it finds (resetting it everytime it finds a one), halting as soon as it is sure of the answer. For instance:

```
function subsequence ( $\boldsymbol{x}$, $k$)
    $l \leftarrow 0$
    for each $x_i \in \boldsymbol{x}$
        if $x_i = 0$
            $l \leftarrow l + 1$
            if $l = k$
                accept and halt
        else
            $l \leftarrow 0$
    reject and halt
```

3.1) The algorithm clearly halts after a linear scan of the input, plus counter increments and comparisons, all of which can be carried out in polynomial time on a Turing machine. Therefore, $S \in$ **P**.

3.2) The algorithm only requires a constant number of counters (the position in the sequence, the counter l), each being of logarithmic size wrt the length of the input sequence x. Therefore, $S \in$ **L**.

Observations

- Although the algorithm only mentions one additional variable l to be used as a counter, the actual implementation might require more than one. For instance, if the algorithm were to be implemented on a TM, at least another counter to keep track of the current position in the input string might be necessary. What's important, is that a constant number of variables is used, and that each is logarithmic wrt input size.

- Counters cannot have *constant* size, otherwise they would not work for larger inputs.