KØBENHAVNS
UNIVERSITET

Computability and Complexity: Problem Set 1

Due: Friday February 23 at 23:59 AoE.

Submission: Please submit your solutions via *Absalon* as a PDF file. State your name and e-mail address close to the top of the first page. Solutions should be written in LaTeX or some other math-aware typesetting system with reasonable margins on all sides (at least 2.5 cm). Please try to be precise and to the point in your solutions and refrain from vague statements. Make sure to explain your reasoning. *Write so that a fellow student of yours can read, understand, and verify your solutions.* In addition to what is stated below, the general rules for problem sets stated on the course webpage always apply.

Collaboration: Discussions of ideas in groups of two to three people are allowed—and indeed, encouraged—but you should always write up your solutions completely on your own, from start to finish, and you should understand all aspects of them fully. It is not allowed to compose draft solutions together and then continue editing individually, or to share any text, formulas, or pseudocode. Also, no such material may be downloaded from or generated via the internet to be used in draft or final solutions. Submitted solutions will be checked for plagiarism. You should also clearly acknowledge any collaboration. State close to the top of the first page of your problem set solutions if you have been collaborating with someone and if so with whom. *Note that collaboration is on a per problem set basis, so you should not discuss different problems on the same problem set with different people.*

Reference material: Some of the problems are "classic" and hence it might be possible to find solutions on the Internet, in textbooks or in research papers. It is not allowed to use such material in any way unless explicitly stated otherwise. Anything said during the lectures or in the lecture notes, or any material found in Arora-Barak, should be fair game, though, unless you are specifically asked to show something that we claimed without proof in class. All definitions should be as given in class or in Arora-Barak and cannot be substituted by versions from other sources. It is hard to pin down 100% watertight, formal rules on what all of this means—when in doubt, ask the main instructor.

Grading: A total score of 80 points will be enough for grade 02, 110 points for grade 4, 140 points for grade 7, 170 points for grade 10, and 200 points for grade 12 on this problem set. Any revised versions of the problem set with clarifications and/or corrections will be posted on the course webpage `jakobnordstrom.se/teaching/CoCo24/`.

Questions: Please do not hesitate to ask the instructors or TA if any problem statement is unclear, but please make sure to send private messages when using Absalon—sometimes specific enough questions could give away the solution to your fellow students, and we want all of you to benefit from working on, and learning from, the problems. Good luck!

1 (10 p) The Arora-Barak textbook defines NP to be the set of languages L with the following property: There is a polynomial-time (deterministic) Turing machine M and a polynomial p such that $x \in L$ holds if and only if there is a witness y of length *exactly* $p(|x|)$ for which $M(x, y) = 1$. An attentive listener will have noticed that in the lectures we were in fact a bit more relaxed, and stipulated that the witness y should be of length *at most* $p(|x|)$, but might be shorter for some x.

Show that these two different definitions of NP are equivalent. That is, prove formally that the two definitions yield exactly the same set of languages in NP. (This is not hard, but please be careful so that you do not run into problems with any annoying details.)

2 (20 p) Let L be the language

$$L = \{n \in \mathbb{N}^+ \mid n = pq \text{ for distinct prime numbers } p \text{ and } q\} .$$

Prove that L is in NP and also in coNP.

3 (20 p) Let OneNegSat be the language of satisfiable CNF formulas in which each clause has at most one negated literal. Prove that OneNegSat is in P.

4 (40 p) In this course we mostly focus on decision problems, although in real life one would be interested in solving search problems yielding actual solutions. We argued in class that the restriction to decision problems is essentially without loss of generality, since efficient algorithms for decision problems can usually be used to solve also search problems efficiently. In this problem we wish to investigate two concrete examples of this.

4a (20 p) Jakob had actually prepared pseudocode for an algorithm `SEARCH-PATH(G, s, t)`, showing how the search version of (s, t)-Path, where the task is to find a path from s to t in G if there is one, can be solved by using an algorithm `DECIDE-PATH(G, s t)` for the decision version of the problem. In the end he did not have time to present this in class, but the algorithm he had in mind was as follows (where `path` is an ordered set, `nbqueue` is a queue of vertices, and `N(v)` denotes the out-neighbours of a vertex `v`):

```
SEARCH-PATH(G, s, t)
    currv := s
    path  := {}
    stuck := FALSE
    while (currv != t and not(stuck))
        stuck   := TRUE
        nbqueue := {}
        for v in N(currv)                   // make queue of neighbours
            if (not(v in path))             //   not already in path
                enqueue(nbqueue, v)
        while (not(empty(nbqueue)) and stuck) // try to extend path with
            nextv := dequeue(nbqueue)       //   some such neighbour
            if (DECIDE-PATH(G, nextv, t))
                stuck := FALSE
                currv := nextv
                append(path, nextv)         // add new vertex to path
    if (stuck)
        return {}
    else
        return path
```

Can you help Jakob by filling in the analysis of why this algorithm is correct, i.e., showing that it will return a path from s to t if there is one and otherwise the empty set? (Or, in case you think there are issues with the algorithm, can you point them out as clearly and convincingly as possible?)

4b (20 p) Another problem discussed in class was FACTORING, i.e., given an integer N, the task of returning the prime factors of N sorted in increasing order (with repetitions if a prime factor occurs to a high power, say). Note that for number-theoretic algorithms like this we measure efficiency in the size of the *representation* of the number N, which is $\lceil \log(N+1) \rceil$ bits, and an efficient algorithm should thus scale polynomially in $\log N$.

To make this into a decision problem FACTOR, let us say that $\langle N, k \rangle \in$ FACTOR if N has a factor f such that $1 < f \leq k$. Show that if we have an algorithm for deciding FACTOR in time polynomial in $\log N$, then we can use this to build an algorithm for the search problem FACTORING that also runs in time polynomial in $\log N$.

5 (20 p) In our proof of the Cook-Levin Theorem, we used the fact that any Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be represented as a CNF formula of size at most $n \cdot 2^n$.

5a (10 p) Use the construction in our proof (or proof sketch) of this lemma to write down a CNF representation of

$$NEQ(x_1, x_2, x_3) = \begin{cases} 0 & \text{if all inputs } x_1,\ x_2, \text{ and } x_3 \text{ are equal,} \\ 1 & \text{otherwise.} \end{cases}$$

5b (10 p) Do the same for

$$MAJ(x_1, x_2, x_3) = \begin{cases} 0 & \text{if at least two inputs are 0,} \\ 1 & \text{if at least two inputs are 1.} \end{cases}$$

Can you find a smaller CNF formula for MAJ than the one that the construction in the proof gives you?

6 (40 p) For a CNF formula F, let $\widetilde{F}$ denote the "canonical 3-CNF version" of F constructed as follows:

- Every clause $C \in F$ with at most 3 literals appears also in $\widetilde{F}$.
- For every clause $C \in F$ with more than 3 literals, say, $C = a_1 \lor a_2 \lor \cdots \lor a_k$, we add to $\widetilde{F}$ the set of clauses

  $$\{y_0,\ \overline{y}_0 \lor a_1 \lor y_1,\ \overline{y}_1 \lor a_2 \lor y_2,\ \ldots,\ \overline{y}_{k-1} \lor a_k \lor y_k,\ \overline{y}_k\}\,,$$

  where $y_0, \ldots, y_k$ are new variables that appear only in this subset of clauses in $\widetilde{F}$.

6a (10 p) Prove that $\widetilde{F}$ is unsatisfiable if and only if F is unsatisfiable. (Please make sure to prove this claim in both directions, and to be careful with what you are assuming and what you are proving.)

6b (10 p) A CNF formula F is said to be *minimally unsatisfiable* if F is unsatisfiable but any formula $F' = F \setminus \{C\}$ obtained by removing an arbitrary clause C from F is always satisfiable. Prove that $\widetilde{F}$ is minimally unsatisfiable if and only if F is minimally unsatisfiable.

6c (20 p) Consider the language

$$\textsc{MinUnsat} = \{F \mid F \text{ is a minimally unsatisfiable CNF formula}\} .$$

What can you say about the computational complexity of deciding this language?

For this subproblem, and for this subproblem only, please look at textbooks, search in the research literature, or roam the internet to find an answer. As your solution to this subproblem, provide a brief but detailed discussion of your findings regarding MinUnsat together with solid references where one can look up any definitions and/or proofs (i.e., not a webpage but rather a research paper or possibly textbook). Note that you should still follow the problem set rules in that you are not allowed to collaborate or interact with anyone other than your partner(s) on this problem set.

7 (50 p) Given a (multi)set $A = \{a_1, a_2, \ldots, a_m\}$ of integer terms and a target sum T, does there exist a subset $S \subseteq [m]$ such that $\sum_{i \in S} a_i = T$? We have learned in class that this problem, known as SubsetSum, is NP-complete. In this problem, we want to look more closely at the reduction establishing NP-hardness and study what happens when we tinker with this reduction.

7a (15 p) Recall the reduction we saw from 3-Sat to SubsetSum constructed as follows: We are given a 3-CNF formula F with m clauses $C_1, \ldots, C_m$ over n variables $x_1, \ldots, x_n$. We build from this F a SubsetSum instance with $2(n+m)$ integer terms and target sum as follows, where all numbers below have $n+m$ decimal digits each:

- For each variable x_i, construct numbers A_i^T and A_i^F such that:
  - the ith digit of A_i^T and A_i^F is equal to 1;
  - for $n+1 \leq j \leq n+m$, the jth digit of A_i^T is equal to 1 if the clause C_{j-n} contains the literal x_i;
  - for $n+1 \leq j \leq n+m$, the jth digit of A_i^F is equal to 1 if C_{j-n} contains $\overline{x}_i$, and
  - all other digits of A_i^T and A_i^F are 0.
- For each clause C_j , construct numbers B_j^1 and B_j^2 such that
  - the $(n+j)$th digit of B_j^1 is equal to 1;
  - the $(n+j)$th digit of B_j^2 is equal to 2; and
  - all other digits of B_j^1 and B_j^2 are 0.
- The target sum T has
  - jth digit equal to 1 for $1 \leq j \leq n$ and
  - jth digit equal to 4 for $n+1 \leq j \leq n+m$.

Since we discussed this only briefly in class, write down a detailed proof establishing that the above is a correct reduction from 3-Sat to SubsetSum that proves the NP-hardness of the latter problem. That is, argue that the reduction (i) is polynomial-time computable, (ii) maps yes-instances to yes-instances, and (iii) maps no-instances to no-instances.

7b (15 p) Given a 3-CNF formula F with m clauses over n variables, run the same reduction as in problem 7a except that the numbers B_j^1 and B_j^2 are omitted and the target sum T has all digits equal to 1. Formulas that map into satisfiable instances of SubsetSum under this modified reduction have a very specific form of satisfying assignments. Describe what such assignments look like.

7c (20 p) Consider the language HackedSat consisting of 3-CNF formulas that map to satisifiable SubsetSum instances under the reduction in problem 7b. What is the complexity of deciding this language? Is it in NP? In P? Or NP-complete? For full credit, provide either a polynomial-time algorithm or a reduction from some problem proven NP-complete in chapter 2 in Arora-Barak or during the lectures.

8 (50 p) Recall that as discussed in class, we can agree on some fixed, standardized encoding of Turing machines in the binary alphabet $\{0, 1\}$. This allows us to view each Turing machine as an integer, namely the number whose binary expansion is the encoding of the Turing machine in question. We can also also agree that integers that do not correspond to Turing machines under this translation are interpreted as the Turing machine that immediate halts regardless of input. Given this convention, any number x encodes a Turing machine M_x, and we can define a function $g : \mathbb{N} \to \mathbb{N}$ by

$$g(x) = \begin{cases} s & \text{if } M_x \text{ takes } s < \infty \text{ steps before halting given the empty string as input;} \\ 0 & \text{if } M_x \text{ does not halt given the empty string as input.} \end{cases}$$

Note that given that we have fixed the encoding of Turing machines into binary strings, this is certainly a well-defined mathematical function that maps any non-negative integer x into some non-negative integer $y = g(x)$.

Even though the function $g(x)$ exists, *computing* it is another matter. In this problem, we want to show that $g(x)$ is not computable in a very strong sense. Namely, your task is to prove that $g(x)$ grows faster than any computable function. That is, show that there cannot exist any monotonically increasing function $h : \mathbb{N} \to \mathbb{N}$ and any Turing machine M^h such that $g(x) = \mathrm{O}(h(x))$ and M^h computes $h(x)$ when given x as input.

9 (60 p) Your task in this problem is to produce a complete, self-contained proof of (the vanilla version of) Ladner's theorem that was discussed briefly during one of the lectures. The goal is (at least) twofold:

- To have you work out the proof in detail and make sure you understand it.
- To train your skills in mathematical writing.

When you write the proof, you can freely consult the lecture notes as well as the relevant material in Arora-Barak, but you need to fill in all missing details. Also, the resulting write-up should stand on its own without referring to the lecture notes, Arora-Barak, or any other source.

Your write-up should be accessible to a student who has studied and fully understood the material during the first two weeks of lectures of this course but does not necessarily know more than that. (However, you do not need to explain again the material in our first lectures, but can assume that they have been fully digested.)

You are free to structure your proof as you like, except that all of the ingredients listed below should be explicitly addressed somewhere in your proof. (You can take care of them in whatever order you find appropriate, however. Please do not refer to the labelled subproblems in your write-up, since it should be a stand-alone text, but make sure that it is easy to find where in your solution the different items are dealt with.)

9a Define

$$\textsc{Sat}_P = \left\{ \psi 0 1^{n^{P(n)}} \;\middle|\; \psi \in \textsc{CnfSat} \text{ and } n = |\psi| \right\}$$

as the language of satisfiable CNF formulas padded by a suitable number of ones at the end as determined by the function P, which we assume to be polynomial-time computable.

9b Prove that if $P(n) = \mathrm{O}(1)$, then $\textsc{Sat}_P$ is NP-complete.

9c Prove that if $P(n) = \Omega(n/\log n)$, then $\textsc{Sat}_P \in \mathsf{P}$.

9d Give a complete description of the algorithm computing $H(n)$ (as in the lecture notes) and prove that H is well-defined in that the algorithm terminates and computes some specific function.

9e Prove that not only does the algorithm terminate, but it can be made to run in time polynomial in n. (Note that there are a number of issues needing clarification here, such as, for instance, how to solve instances of $\textsc{CnfSat}$ efficiently enough.)

9f Prove that $\textsc{Sat}_H \in \mathsf{P}$ if and only if $H(n) = \mathrm{O}(1)$.

9g Prove that if $\textsc{Sat}_H \notin \mathsf{P}$, then $H(n) \to \infty$ as $n \to \infty$.

9h Assuming that $\mathsf{P} \neq \mathsf{NP}$, prove that $\textsc{Sat}_H$ does not lie in P but also cannot be NP-complete.