

CSE 200
Computability and Complexity
Homework 3
Polynomial-time Hierarchy
Space Complexity
Probabilistic algorithms
Average-case complexity and cryptography Due
June 4

May 29, 2012

Give proofs for each problem. Proofs can be high-level, but be precise. You may use without giving a proof any result proved in class or in the textbook. In particular, to prove NP-completeness, it suffices to give a reduction from any of the NP-complete problems from the text or from class. However, you must show your reduction is valid, by showing the equivalence of the constructed instance and the original.

From last homework Prove that NP is closed under polynomial-time Turing reducibility (i.e., that $P^NP \subseteq NP$ if and only if $NP = co-NP$).

Polynomial-time hierarchy Assume that $NP \subseteq TIME(n^{O(\log n)})$. Prove that for every i there is a k so that $\Sigma_i^P \subseteq TIME(n^{O((\log n)^k)})$

Randomized algorithms Prove that, if $NP \subseteq BPP$, then $NP = RP$.

Space complexity Remember that k -SAT is the satisfiability problem restricted to CNF formulas with clause sizes at most k . Prove that 2-SAT is co- NL -complete (under deterministic logspace mapping reductions). (Hint: The previous version said NL complete. While

$NL = co - NL$, so this is true, it is easier to map the complement of $2SAT$ to the connectivity problem and vice versa. At least half of the problem is to prove that $2-SAT$ is in $co - NL$. Also, you may use any facts about NL mentioned in class without proof.)

Foundations of cryptography Remember that a *one-way function* is a function $F \in FP$ so that for any polynomial time probabilistic algorithm A and any polynomial p , for sufficiently large n , the probability over input x of length n and A 's random coins that A inverts $F(x)$, i.e., that $F(A(F(x))) = F(x)$, is at most $1/p(n)$. Let $R(x, y)$ be a relation in P , with $|y| = poly(|x|)$. A *solved instance sampler* S for R is a probabilistic time procedure that given 1^n , $S(1^n)$ generates a distribution on pairs (x_n, y_n) so that $|x_n| = n$ and $R(x_n, y_n)$. The sampler *produces hard instances* if for any probabilistic polynomial-time algorithm A and any polynomial p , for sufficiently large n , the probability that $R(x_n, A(x_n))$ is less than $1/p(n)$. Prove that there is a one-way function F if and only if there is a relation $R \in P$ and a solved instance sampler S for R producing hard instances.

Sudoku experiment We started looking at reducing sudoku problems to SAT last assignment.

The *sudoku* problem of size n is as follows. The input is an $n^2 \times n^2$ matrix M whose entries are either “blank” or an integer between 1 and n^2 . A solution fills in the blank spaces with integers between 1 and n^2 . The following constraints must be met: Each integer from 1 to n^2 appears exactly once in each row, in each column, and in each $n \times n$ sub-matrix of the form $M[jn + 1 \dots (j + 1)n][in + 1 \dots (i + 1)n]$ for each $0 \leq i, j \leq n - 1$. The problem is to find any solution meeting the constraints, or return “no solution possible” if there is no such solution.

Last assignment, you gave at least two different ways to reduce the Sudoku problem to $CNF - SAT$.

Try solving sudoku problems by combining the above reductions with a complete SAT solver, such as Zchaff (download page,

[//www.princeton.edu/~chaff/zchaff/index2.html](http://www.princeton.edu/~chaff/zchaff/index2.html)

Use as test inputs puzzles of sizes 9 by 9, 16 by 16, and 25 by 25 (and larger, if possible) generated by picking the entries of the first row and column as random permutations, conditioned on all entries in

the first block being distinct. (Alternative: Fill in the first block with random distinct entries. Extend the first row and column at random with distinct entries not already used.)

Be sure to credit the SAT solver you use.

The experiment should return the following information: For each size, how much time did the SAT solver take using different reductions? Which reduction is best? Did the difficulty ratings reflect in the time taken by these solvers?

(Note: Be careful not to use up too much computer time. Don't leave programs running unsupervised too long. Depending on your algorithm and reduction, you may find even very small sizes take huge amounts of time.)