

CS 6505: Algorithms, Computability and Complexity

Final Exam

Due 11:59pm on December 11, 2007

This take-home exam contains 8 problems. You are required to solve 5 of the 8 problems. If you solve more than 5 problems, then the 5 highest scores of the 8 problems will be added as your final exam score, and the scores of the remaining problems will be counted as extra credit. You must type your solutions, and submit your solutions in PDF electronically by the above deadline to Yan Ding (ding@cc.gatech.edu), Justin Cranshaw (justin@cc.gatech.edu), and Apurva Mudgal (apurva@cc.gatech.edu). **No collaboration is allowed for the exam – you must work on the exam completely on your own.**

Good Luck!

Problem 1

For each of the following two languages, answer whether or not it is decidable, and prove your answer.

1. $L_1 = \{\langle M \rangle : M \text{ is a TM and } M \text{ takes more than 6505 steps on some input}\}$
2. $L_2 = \{\langle M \rangle : M \text{ is a TM and } M \text{ takes more than 6505 steps on some input in } L(M)\}$

Problem 2

The derivative of a polynomial $f(x) = \sum_{k=0}^n a_k x^k$ is

$$\frac{d}{dx}f(x) = \sum_{k=1}^n k a_k x^{k-1}.$$

For $0 \leq t \leq n - 1$, the t -th derivative of $f(x)$ is defined as

$$f^{(t)}(x) = \begin{cases} f(x) & \text{if } t = 0 \\ \frac{d}{dx}f^{(t-1)}(x) & \text{if } 1 \leq t \leq n - 1. \end{cases}$$

Devise an algorithm that given a polynomial $f(x)$ of degree n and a point c , evaluates all the derivatives $f^{(0)}(c), f^{(1)}(c), \dots, f^{(n-1)}(c)$ at point c in time $O(n \log n)$, assuming that in one step one can compute an addition or a multiplication of two complex numbers. (**Hint:** Reduce the problem to a single convolution of two properly chosen vectors, one involving the coefficients a_0, a_1, \dots, a_n of the polynomial f , and the other involving the powers $1, c, \dots, c^n$ of the point c .)

Problem 3

Give a polynomial-time reduction from **Vertex-Cover** to **SAT**. That is, give a polynomial-time computable function f such that for any graph G and integer k , $\Phi = f(G, k)$ is satisfiable if and only if G has a vertex cover of size k . The reduction shall use some ideas from the proof of the Cook-Levin Theorem, but shall be much simpler. You may find the following guideline helpful:

- For each $1 \leq i \leq k$ and $1 \leq v \leq n$, introduce a variable $x_{i,v}$. The intention is that $x_{i,v}$ is to be set to 1 if and only if v is the i -th vertex of a (supposed) vertex cover of size k .
- The following are necessary and sufficient for G to have a vertex cover of size k :
 - There is a set of k vertices, whose i -th vertex, for each $1 \leq i \leq k$, is a distinct vertex in $\{1, \dots, n\}$; that is, the i -th vertex is a vertex in the graph G , and for $i \neq j$, no vertex can be both the i -th and the j -th vertices of the set.
 - For each edge $\{u, v\}$ in G , u or v is one of the k vertices in the above set.

Try to construct a short Boolean formula that expresses the above conditions.

Problem 4

Let X , Y and Z be three disjoint sets. Let $E \subseteq X \times Y \times Z$, that is, E is a subset of triples of the form (x, y, z) where $x \in X$, $y \in Y$ and $z \in Z$. A subset $M \subseteq E$ of triples is said to be a *3-D matching*¹ if for every two distinct triples $(x, y, z), (x', y', z') \in M$, it holds that $x \neq x'$, $y \neq y'$ and $z \neq z'$. The Maximum 3-D Matching problem is the following optimization problem: Given disjoint sets X , Y and Z , and a subset $E \subseteq X \times Y \times Z$ of triples, find a *maximum* 3-D matching, i.e. a 3-D matching with the largest size. Maximum 3-D Matching is a well-known NP-Hard problem.² In this problem, you will give a polynomial-time algorithm that *3-approximates* Maximum 3-D Matching, using ideas from the approximation algorithm for Vertex-Cover.

1. Define a subset $C \subseteq X \cup Y \cup Z$ to be a *3-D cover* if for every triple $(x, y, z) \in E$, it holds that $x \in C$, $y \in C$ or $z \in C$. What is the relation between the size of any 3-D matching and the size of any 3-D cover?
2. Define a 3-D matching to be *maximal* if it is not a proper subset of any 3-D matching. What is the relation between the size of any *minimum 3-D cover* and the size of any *maximal 3-D matching*?
3. Based on Parts (1) and (2), give a polynomial-time algorithm that *3-approximates* Maximum 3-D Matching, that is, given disjoint sets X , Y and Z , and a subset $E \subseteq X \times Y \times Z$ of triples, finds a 3-D matching whose size is at least $1/3$ of the maximum 3-D matching size.

¹Note that a 2-D matching is just a matching in a bipartite graph.

²This should be contrasted with the Max 2-D (i.e. Bipartite) Matching problem that can be solved efficiently.

Problem 5

The following is a well-known theorem in graph theory.

Theorem 1 *In every bipartite graph, the size of a maximum matching equals the size of a minimum vertex cover.*

1. Using the above theorem, give a polynomial-time algorithm that decides **Vertex-Cover** for *bipartite* graphs, that is, a polynomial-time algorithm that given a *bipartite* graph G and an integer k , decides whether G has a vertex cover of size k .
2. Using just Part (1), give a polynomial-time algorithm that given a *bipartite* graph G , finds a *minimum* vertex cover of G .

Problem 6

In this problem, you will give a proof of Theorem 1 stated in the previous problem using the Max-Flow Min-Cut Theorem, and also give a faster algorithm for finding a minimum vertex cover of a given bipartite graph.

Let $G = (X, Y, E)$ be a bipartite graph. Construct a flow network $G' = (V, E')$ as follows:

- $V = X \cup Y \cup \{s, t\}$, where s and t are the newly added source and sink respectively.
- $E' = \{(x, y) : \{x, y\} \in E, x \in X, y \in Y\} \cup \{(s, x) : x \in X\} \cup \{(y, t) : y \in Y\}$, and
 - for each $x \in X$ and $y \in Y$, $c(s, x) = 1$ and $c(y, t) = 1$;
 - for each $\{x, y\} \in E$ with $x \in X$ and $y \in Y$, $c(x, y) = \infty$.

Let M be a *maximum matching* in the original graph G , and let (S, T) be *minimum cut* in the network G' , where $s \in S$ and $t \in T$.

1. Show that $|M|$ equals the maximum flow value in G' , and thus the capacity of (S, T) .
2. From Part (1), conclude that for each edge $\{x, y\} \in E$, with $x \in X$ and $y \in Y$, it holds that $x, y \in S$, or $x, y \in T$, or $x \in T$ and $y \in S$.
3. Show that $C = (X \cap T) \cup (Y \cap S)$ is a *vertex cover* in G , and moreover $|C| = |M|$. Conclude that C is a *minimum* vertex cover in G .
4. Based on the above, give an efficient algorithm for finding a minimum vertex cover of a given bipartite graph. What's the running time of your algorithm?

Problem 7

Alice and Bob each have a database of N distinct records, each of which is k -bits long, where $k < N \ll 2^k$. Suppose that they want to test whether the two databases are identical, that is, whether they contain the same records.

1. Prove that any *deterministic* protocol for solving this problem *requires* $N(k - \log_2 N)$ bits of communication between Alice and Bob. You may find the following inequality useful: For any two natural numbers ℓ and m with $m \geq \ell$, $\binom{m}{\ell} \geq (\frac{m}{\ell})^\ell$. (**Hint:** How many databases with N distinct k -bit records are there?)
2. Give a simple, communication-efficient *randomized* protocol for solving this problem, and analyze its efficiency and probability of error. Your protocol should require that Alice and Bob exchange *only* $O(k)$ bits, and guarantee that the probability of error is at most $2^{-\Omega(k)}$. The computation time of Alice and Bob in your protocol should be about $\tilde{O}(kN)$.

Problem 8

Let $f : \{0, 1\}^* \rightarrow \{0, 1\}$ be a Boolean function. Suppose that there is a randomized polynomial-time algorithm A for computing f such that for each x with $f(x) = 1$, $A(x) = f(x) = 1$ always; and for each x with $f(x) = 0$, $\Pr[A(x) = 0] \geq 1/2$, where the probability is taken over the random coin tosses of A . Suppose that at the meantime, there is another randomized polynomial-time algorithm B for computing f such that for each x with $f(x) = 0$, $B(x) = f(x) = 0$ always; and for each x with $f(x) = 1$, $\Pr[B(x) = 1] \geq 1/2$. Construct a randomized algorithm C for computing f , using algorithms A and B as subroutines, such that

1. For each $x \in \{0, 1\}^*$, $C(x) = f(x)$ always.
2. Algorithm C runs in expected polynomial time.