

Homework Assignment 5

November 18, 2004

Due Date: December 7

Core Problems

1. (15 pts) We consider the MAX-3-SAT optimization problem in which the input is a 3-CNF-SAT formula $\phi = C_1 \wedge \dots \wedge C_r$ over Boolean variables x_1, \dots, x_n . (As a reminder, a *literal* is either a variable or its negation. For a literal ℓ , $\bar{\ell}$ is the negation of the literal. So if $\ell = x_i$ then $\bar{\ell} = \bar{x}_i$ and if $\ell = \bar{x}_i$, then $\bar{\ell} = x_i$.)

The goal of MAX-3-SAT is to find an assignment to the boolean variables that satisfies as many of the clauses in ϕ as possible. Here you will use the accounting method to prove that the following greedy algorithm for solving this problem has a $4/3$ approximation ratio. Unlike the problem we discussed in class, this algorithm does not use any randomization.

Max-3-Sat-Approx(ϕ)

$T = \emptyset$ (the literals that are assigned to be true)

$L = \{x_1, \bar{x}_1, \dots, x_n, \bar{x}_n\}$

$S = \{C_1, \dots, C_r\}$ (clauses not yet satisfied)

While some clause in S contains a literal in L

 Let y be the literal in L contained in the most clauses of S

$T = T \cup \{y\}$ (so if $y = \bar{x}_i$ this sets x_i to false)

$L = L - \{y, \bar{y}\}$

 Let X be the set of clauses in S that contain y

$S = S - X$

Return any assignment A in which all literals in T are true

Let me get you started. The accounting scheme I suggest charges a penalty of \$1 whenever a literal ℓ in some clause in S is set to false (i.e. $\bar{\ell}$ is placed into T). Let p_i be the penalty accrued during the i th iteration of the while loop and let p_{tot} be the total penalty accrued. Let s_i be the number of clauses removed from S (because they were satisfied) during the i th round of the for loop. Let s_{tot} be the total number of clauses removed from S . Let S_{fin} be the value of S when the approximation algorithm exits the while loop. Let C_{approx} be the number of clauses satisfied by this approximation algorithm and let C_* be the number of clauses satisfied by an optimal solution. Think about the relationship between the following C_* and r ; C_{approx} and s_{tot} ; r , s_{tot} and $|S_{\text{fin}}|$; p_{tot} and $|S_{\text{fin}}|$; s_i and p_i . You take it from here....

2. (15 pts) Below is a fully-polynomial time approximation scheme (FPTAS) for 0-1 knapsack and guide you through the proof for its correctness. Let $I = \langle v_1, \dots, v_n, w_1, \dots, w_n, W \rangle$ be the knapsack input where there are n items where item i has value v_i and weight w_i . Finally, the knapsack can hold weight W . Also you are given that for all i , $w_i \leq W$. Let v be the maximum value of an item.

The basis of this FPTAS is the fact that there is a pseudo-polynomial dynamic programming algorithm, call it DPA, to optimally solve the knapsack problem that has time complexity $O(n^2 v \log W)$. Observe this is not a polynomial time algorithm because it has an exponential dependence on $\log v$ which is the number of bits needed to encode a value of the most valuable item. (If you are interested in designing this dynamic programming algorithm, I can give some guidance. It is similar in flavor to problem 2 from your first exam).

The FPTAS works as follows where $\epsilon > 0$ is any real where the requirement is that the FPTAS output a solution that is a $(1 + \epsilon)$ approximation.

```

Knapsack-FPTAS( $I, \epsilon$ )
  Let  $k = \frac{v}{(1 + \frac{1}{\epsilon})n}$ 
  For  $i = 1$  to  $n$ 
     $v'_i = \lfloor v_i/k \rfloor$ 
  Knapsack input  $I' = \langle v'_1, \dots, v'_n, w_1, \dots, w_n, W \rangle$ 
  Return DPA( $I'$ )

```

In other words, the precision of the values are reduced by a factor of k in I' by replacing v_i by v'_i . Then the dynamic programming algorithm finds an optimal solution (i.e. the set of items to take) to I' and uses this as the approximate solution for I .

- (a) (1 pt) Argue that the algorithm yields a legal solution.
 - (b) (9 pts) Prove that the solution returns is a $(1 + \epsilon)$ - approximation. *Hint: Think about how much replacing v_i by v'_i could reduce the value of the solution obtained. Also you will probably want to make use of the fact that $C_* \geq v$. Convince yourself that this is true. Remember that we assume that for all i , $w_i \leq W$.*
 - (c) (5 pts) Prove that the algorithm runs in time polynomial in the number of bits to encode the knapsack input and $1/\epsilon$. You should state the time complexity of Knapsack-FPTAS as a function of n, v, W , and ϵ and briefly argue it is a fully-polynomial time approximation scheme.
3. (10 pts) Give the best lower bound you can for the problem of finding the median among the $2n + 1$ elements in the two sorted arrays $A[1] < A[2] < \dots < A[n] < A[n + 1]$ and $B[1] < B[2] < \dots < B[n]$ using a comparison-based algorithm that can only learn about the array elements by posing questions of the form is $A[i] < B[j]$ for $1 \leq i \leq n + 1$ and $1 \leq j \leq n$.
 4. (15 pts) Consider the problem of determining whether an undirected graph with n vertices has a vertex of degree 2 or more. Assume that the algorithm is restricted to ask questions of the form “Is there an edge between i and j ?”. Give the best adversary argument you can on the number of such questions required to determine if there is a vertex in the graph with degree 2 or more.
 5. (8 points) Consider the following on-line problem. You have to buy an item and know that you are going to be offered a sequence of prices $\langle p_1, p_2, p_3, \dots, p_m \rangle$ all in the range, $\ell \leq p_i \leq h$.

When presented with the option of buying an item at price p_i , you already know p_1, \dots, p_{i-1} and have passed on these offers, but you know nothing about p_i, \dots, p_m except that they are all in the interval $[\ell, h]$. So when presented with p_i you can either buy the item for that price, or pass in which case you have committed to buying that item at price among p_{i+1}, \dots, p_m . (So if you wait until p_m , then you are obligated to buy the item at that price.) Your goal is to buy the item for the cheapest price. Here is a deterministic on-line algorithm A for this problem. Accept the first p_i such that $p_i \leq \sqrt{\ell h}$. (If all $p_i > \sqrt{\ell h}$ then A accepts p_m .)

Prove that this algorithm is \sqrt{r} competitive where $r = h/\ell$ is the ratio of the highest to lowest possible price.

6. (12 pts) Consider the “searching for a hole in the fence” problem that was considered in class. You are initially standing at the origin of the real line. The hole is at some unknown (positive or negative) integer coordinate h . You can move left or right at a cost equal to the distance moved, and the game continues until you reach h . The goal is to minimize the distance traveled. In class we gave a deterministic on-line algorithm with a competitive ratio of 9 (which is optimal for deterministic strategies). In this problem you are to describe and analyze a randomized algorithm whose competitive ratio is at most 7. If you prove something better than 7-competitive then you will receive more than 12 pts. So challenge yourself to prove the best competitive ratio that you can.

Hint: To get a competitive ratio of 7, a single flip of a fair coin is all that you need.

Advanced Problems (Required only for CSE 541T students)

7. (15 pts) Consider the problem of determining whether an undirected graph G with 2^k vertices is connected (i.e. whether there is an undirected path between each pair of vertices). Assume that the algorithm is restricted to ask questions of the form “Is there an edge between i and j ?”. Give the best adversary argument you can on the number of such questions required to determine if G is connected. You should go well beyond the solution for Practice Problem 2.

Hint: Extend the adversary argument given in the practice problem solutions for Problem 2. Think recursively.

Challenge Problem (Optional for all students)

8. Using an adversary argument to show that *any* comparison-based algorithm to find the second smallest of n elements must make at least $n + \lceil \lg n \rceil - 2$ comparisons. (Although I’m not asking you to find it, there is a comparison-based algorithm that makes at most $n + \lceil \lg n \rceil - 2$ comparisons.)

We’ll just record a * (done correctly) or \checkmark (very close) as well as giving you comments on what you submit.