CS170 Fall 2013 Solutions to Homework 12

Zackery Field, section Di, 103, `cs170-fe`

December 6, 2013

1 [10 pts.] Making DAGs is hard

Consider the search problem Max-Acyclic-Induced-Subgraph:

INPUT: A *directed* graph $G = (V, E)$, and a positive integer k.

OUTPUT: A subset $S \subseteq V$ of size k such that the graph G_S obtained from G by keeping exactly those edges whose endpoints are in S is a DAG.

Show that Max-Acyclic-Induced-Subgraph is NP-complete.

- Show that the Max-Acyclic-Induced-Subgraph (MAIS) problem is in NP.

  Given an output subset $S \subseteq V$ we can run a DFS on G_S and determine if there are any cycles. If there are no cycles, then a DAG of proper size (k), was constructed. If there is a cycle then G_S is not a DAG. The graph G_S can be constructed from G in polynomial time by iterating through the vertices in S and including only edges whose endpoints u and v are both in S. The cycle check on G_S can be done by DFS, which runs in linear time. Therefore, a solution to a MAIS problem can be checked in polynomial time, which implies that MAIS is in NP.

- Reduction: **Independent Set** $\rightarrow$ **MAIS**, to show NP-completeness.

  Let $T = (G, k)$ be some instance of an independent set problem, where $G = (V, E)$ is the undirected input graph, and k is the size of the desired independent subset of vertices V. If there is no independent subset of size k, then the algorithm should output that none exists. If a subset does exist, then it should output that subset. We first define a function $f(T)$ that takes as input an undirected graph G and desired set-size k, and turns G into a directed graph G'. This directed graph, and its associated k value, form the inputs of a MAIS problem.

  Construction of $G = (V, E) \rightarrow G' = (V, E')$:

  For each of the undirected edges $e = \{u - v\}, \forall e \in E$, replace that edge with two directed edges $\{u \rightarrow v\}$, and $\{v \rightarrow u\}$.

Continued on Page 6

2 [10 pts.] Reductions redux

Show that you can not hope to do much better than insertion or deletion worst case complexity $\Omega(n \log n)$, where n is the number of elements in queue. Prove that in the "comparison model" , there cannot exist a priority queue implementation in which both Insert and Delete-Min operations have worst case complexity $O(1)$.

Lemma: Comparison Sorting n elements takes $O(n \log n)$ time.

Assume, for the sake of contradiction, that we have some priority queue whose insert and delete-min updates both take $O(1)$ time. We have some set of n items in our priority queue, and each of these items (i) can be compared to any other item (j) to reveal a relationship, either $i < j$ or $i > j$ in $O(1)$ time. Without loss of generality, we can implement the priority queue with a sorted array whose delete-min update takes $O(1)$. This running time is perfectly valid since we can lookup the min entry in $O(1)$ time. As the assumption states, the insert update must also take $O(1)$ time. In terms of the sorted array, this means that there is some update procedure by which you can insert an element into a sorted array in $O(1)$ time. If this is the case, then creating a sorted array for n unsorted items would only take $O(n)$ time, a contradiction. Therefore, there cannot be an implementation of a priority queue that has $O(1)$ insert and delete-min updates in the Comparison-Model.

3 [10 pts.] Finding Zero(s)

Consider the problem Integer-Zeros.

INPUT: A multivariate polynomial $P(x_1, x_2, x_3, \ldots, x_n)$ with integer coefficients, specified as a sum of monomials.

OUTPUT: Integers $a_1, a_2, \ldots, a_n$ such that $P(a_1, a_2, a_3, \ldots, a_n) = 0$.

Show that 3-SAT reduces in polynomial time to Integer-Zeros.

Assuming that CNF is provided, we are given a 3-SAT problem statement S of the form: $S = \{(x_1 \vee x_2 \vee \bar{x_3}) \wedge (x_1 \vee \bar{x_i} \vee \bar{x_n}) \wedge \cdots \wedge (x_n \vee x_2 \vee x_{i+1})\}$

Let $f_{IZ}(S)$ be the function that formulates an Integer-Zeros problem from a 3-SAT problem. Given a 3-clause $(x_1 \vee x_2 \vee x_3)$, this function will construct a polynomial $P_i(x_1x_2x_3) = x_1x_2x_3$. Since all of the relations within a 3-clause are OR and none of the clauses are negated, we can simply multiply the 3-clause into one monomial. This way, whenever one of the x_i clauses is zero, the entire polynomial will be zero, which is equivalent to the or relation where 0 ≡TRUE. If there are one or more single clause $(\bar{x})$ in the 3-clause that is/are negated, then this can be represented by inserting $(1-x)$ into the monomial. Here is one such example: $(x_1 \vee \bar{x_2} \vee \bar{x_3})$, $P_i(x_1x_2x_3) = x_1(1-x_2)(1-x_3)$. The only way to make a $1-x$ negation clause true (zero) is to set $x = 1$, thus 1 ≡FALSE. The AND relations between each of the 3-clauses is represented by the summation of all of the individual 3-clause monomials; $(x_1x_2x_3) + \cdots + (x_1(1-x_2)(1-x_3)) = 0$. This summation is equivalent to the boolean AND relation, because if any single monomial is nonzero, then the entire sum will be nonzero.

Now that we have defined a polynomial time algorithm for constructing an Integer-Zeros problem, we must determine if this problem is equivalent. Each monomial is constructed s.t. any one zero value for some x_i, or some unit value for some $(1 - x_i)$ will be sufficient to make the entire term zero. This is consistent with the 3-clauses, because in CNF the relation between each of the clauses in a 3-clause is OR. The AND between each 3-clause is represented by the summation of the individual monomials, as stated above. This summation construction is sufficient to ensure that all monomials are satisfied in order for the entire polynomial to be satisfied.

The Integer-Zeros problem will either output a set of inputs $\{a_1, a_2, \cdots, a_n\}$ which map to $P(a_1, a_2, \cdots, a_n) = 0$ for some $P(x_1, x_2, \cdots, x_n)$, or it will output that no such inputs exist. This set of inputs, by construction of P, will either be zero, or one, corresponding to TRUE and FALSE, respectively. A function $h(O)$ that maps this returned set of inputs back to a solution to the original 3-SAT will simply look at the values of the individual clauses x_i, then assign values based on the relation: x_i =TRUE if $a_i = 0$, and x_i =FALSE if $a_i = 1$.

Note: Since the polynomial P is meant to be represented as the sum of polynomials in order to ensure that each unit clause x_i is given a value in the set of output inputs (rather than just the compounded 3-clauses) insert for each x_i one monomial: $x_i(1-x_i)$. This will also ensure that each value in the output set of inputs is either zero or one.

4 [10 pts] Approximately independent

Consider the following algorithm for finding an independent set in an undirected graph $G = (V, E)$.

Algorithm 1 Finding a large independent set

1: **Input** A graph $G = (V, E)$
2: **Output** A subset $I \subseteq V$
3: **while** $V \neq \emptyset$ **do**
4: Let v be the vertex with the *smallest* degree in $G' = (V, E)$
5: $I \leftarrow I \cup \{v\}$
6: Let S be the set of neighbors of v in G'
7: $V \leftarrow V - (S \cup \{v\})$
8: $E \leftarrow E - \{e | e$ is incident on a vertex in S
9: **end while**

(a) Show that the output of algorithm 1 is an independent set.

Let $G = (V, E)$ be some problem instance of the algorithm, and let $I \subseteq V$ be the solution that is output.

Assume, for the sake of contradiction, that I is not an independent set. By the definition of independence, for some two vertices $\{u, v\} \in I$, u and v share an edge; u and v are neighbors.

For some iteration of the algorithm v (or u, arbitrary choice) is the vertex of smallest degree in G' (line 4). v is then added to I in line 5. When the set S of neighbors of v is constructed in line 6, $u \in S$, by definition. Then, (line 7) all edges $\in S$ and v are removed from the list of iterable vertices V. This step ensures that u will never be chosen at a later iteration. In the last line (line 8) all edges incident on a vertex in S, including the edge $u - v$, are removed from the set of edges E. This is a contradiction, since the assumption at the start was that u and v are neighbors, but the edge $u - v$ that defines them as such is removed, during line 8. Therefore, I is an independent set.

Continued on Page 7

5 [10 pts.] Coin Tosses vs Local Search

We are given a graph $G = (V, E)$ and our goal is to produce a partition $(A, V - A)$ of V such that the number of edges crossing the cut $(A, V - A)$ is as large as possible. Let C be the *maximum* size of such a cut. We consider two approaches for solving this problem:

Devil-May-Care For each vertex $v \in V$, we toss a fair coin. If the coin comes up heads, we put v in A, else we don't put it in A.

Prudence-Is-Us Given a subset $S \subseteq V$ and a vertex $v \in V$, let $N(v, S)$ denote the number of neighbors of v in S (not including v, in case $v \in S$). We then use algorithm 2.

Algorithm 2 Finding the large cuts

Input A *graph* $G = (V, E)$
Output A *subset* $A \subseteq V$
$A \leftarrow \emptyset$; flag $\leftarrow$ **true**
repeat
  flag $\leftarrow$ **false**
  if $\exists v \in V - A$ *such that* $N(v, A) < N(v, V - A)$ **then**
    $A \leftarrow A \cup \{v\}$
    flag $\leftarrow$ **true**
  else if $\exists v \in A$ *such that* $N(v, A) > N(v, V - A)$ **then**
    $A \leftarrow A - v$
    flag $\leftarrow$ **true**
  end if
until flag $\neq$ **true**;

(a) Let $(A, V - A)$ be the partition produced by the first approach. Show that the expected number of edges crossing the cut is at least $C/2$.

(b) Show that there are at most $|E| + 1$ iterations of the loop at line 4 in algorithm 2.

(c) Show that if A is the set output by algorithm 2, then the number of edges crossing $(A, V - A)$ is at least $C/2$.

Extra space for Problem 1

Continued from Page 1

The MAIS problem (G', k) will output either a subset of vertices S that constitute a DAG, or it will output that no subset of size k exists. This solution subset requires no post-processing function $h(S)$. Any output of the MAIS problem ,with the pre-processing function $f(T)$, will output an independent set.

Proof:

Note that one cannot choose two adjacent vertices, $\{u, v\}$, to be placed in the DAG subset S of any MAIS problem. This is because the particular construction chosen for the directed graph input forms a cycle between any two adjacent vertices.

Remember that a DAG is defined as having no cycles, so for this problem their has to be some equivalence between the cycles in graph G' and the non-independence of the vertices in these cycles. In other words, a subset of vertices of G' (and their component edges) is acyclic if and only if the subset is also an independent set on G.

The trivial portion of this equivalence is; Any independent set is acyclic. In the graph G', if no two vertices are pairwise adjacent in the subset (independent), then there is no cycle that can be created, because none of the vertices are connected. (As the problem states, an edge is only considered if *both* of its endpoints are in the subset)

The reverse is also true; Any acyclic subset of vertices of G' is independent on G. Let R be some acyclic subset of vertices. Assume for the sake of contradiction that R is not independent. Then there exists two vertices u, and v that are adjacent. But by construction, these two vertices create a cycle because both $\{u \to v\}$, and $\{v \to u\}$ exist, a contradiction. Therefore, any acyclic subset of G' is also an independent set on G. This shows that the output to the MAIS problem on $\{G', k\}$ will output an independent subset of size k, if and only if one exists.

Extra space for Problem 4

Continued from Page 4

(b) Show that if the maximum degree of a vertex in G is Δ, and the size of the maximum independent set is α, then we have $|I| \leq \frac{\alpha}{\Delta+1}$.

Start by rearranging the inequality to get: $\Delta + 1 \leq \frac{\alpha}{|I|}$ For a lower bound on Δ, if we take a single vertex graph that has no self loops, then $\Delta = 0$. This shows that $\alpha = |I|$ is possible, if and only if $\Delta = 0$. An upper bound on Δ would be some completey connected graph, where if n is the number of vertices then $\Delta = n - 1$. In this case, we can intuit that $\alpha = 1$, and the inequality becomes $(n-1) + 1 \leq \frac{1}{|I|} \Rightarrow |I|n \leq 1 \Rightarrow |I| = 0$. But this is a contradiction since we know that on the first iteration of the algorithm at least one vertex is added to $|I|$. Therefore, the inequality does not hold.

Extra space for Problem 5

Continued from Page 5