Homework 4

April 16, 2010

Notes

• No submission is required.
1. The assignment problem is usually stated this way: There are \( n \) people to be assigned to \( n \) jobs. The cost of assigning the \( i \)th person to the \( j \)th jobs is \( cost(i,j) \). You are to develop a branch and bound algorithm that assigns every job to a person and at the same time minimizes the total cost of the assignment.

Solution:
In our recursive (exhaustive) algorithm we pick a person (who wasn’t assigned before) and assign him a job (which wasn’t assigned before). We need to define how the following two steps are done (i) How to select the next node and (ii) how to prune a subtree. Let \( f(x) \) be the cost for all assignment already done. \( g(x) \) be the least cost possible for rest of the assignments. We obtain an estimate for this by calculating the least cost job possible for each person.

\[
 g(i) = \min_{1 \leq j \leq n} cost(i,j)
\]

\[
 g(x) = \sum_{\forall \text{remaining persons } i} g(i)
\]

We maintain a value \( lower \) which is the current best solution. For step (i) we select a node whose \( g(x) \) value is minimum. For step (ii) we prune the subtree if \( g(x) \) value is greater than \( lower \).

2. This problem is called the postage stamp problem. Envision a country that issues \( n \) different denominations of stamps but allows no more than \( m \) stamps on a single letter. For given values of \( m \) and \( n \), write values, from one on up, and all possible sets of denominations that realize that range. For example, for \( n = 4 \) and \( m = 5 \), the stamps with values (1, 4, 12, 21) allow the postage values 1 through 71.
Are there any other sets of four denominations that have the same range. Use a backtracking algorithm.

Solution:
This is similar to n-queens problem. First assume that you have a black box which calculates the range when given \( n \), \( m \) and set of stamp values. Now iterate over all possible combinations of \( n \)-tuples. Let \( \{a_1, a_2, ..., a_4\} \), be the \( n \)-tuple. Start with \( \{1, 2, ..., 4\} \). Vary \( a_i \) from 1 till 71. Do this for each \( a_i \) from \( a_4 \) to \( a_2 \).

3. ATLEAST-TWO-SAT is defined as the problem of checking whether a given propositional formula in 3-CNF form has at least two different satisfying assignments. Prove that ATLEAST-TWO-SAT is NP-complete.

Solution:
First we prove that is in NP. For this consider that we have a propositional formula \( p \) in 3-CNF form (in some \( n \) variables and some \( k \) clauses) and we have access to two assignments \( a_1(p) \) and \( a_2(p) \). Note that each assignment is in the form of \( n \) bits corresponding to true/false values of each of the \( n \) variables. It is easy to check in time \( O(k) \) (polynomial in the input size) whether \( a_1(p) \) and \( a_2(p) \) are satisfying assignments for the formula \( p \). Hence this problem is in NP.

Solution:
To prove it is NP-hard, do a reduction from 3-SAT to ATLEAST-TWO-SAT which takes polynomial time. This will prove that ATLEAST-TWO-SAT is \textit{at least as hard as} 3-SAT. As 3-SAT is NP-hard, we thus prove that ATLEAST-TWO-SAT is also NP-hard. Note that the reduction must take polynomial time. The actual reduction is as follows: Consider a propositional formula \( p \) in \( n \) variables \( x_1, x_2, x_3, ..., x_n \) which is an input to 3-SAT. Add in a new clause \( x_{n+1} \lor \overline{x_{n+1}} \) so that the formula becomes \( \bar{p} = p \land (x_{n+1} \lor \overline{x_{n+1}}) \). \( \bar{p} \) is an input to ATLEAST-TWO-SAT. \( \bar{p} \) will have two or more satisfying assignments \textit{if and only if} \( p \) has at least one satisfying assignment, i.e. \textit{if and only if} \( p \) is satisfiable. Therefore, if we could solve ATLEAST-TWO-SAT in polynomial time, clearly we could have solved 3-SAT in polynomial time. This proves that ATLEAST-TWO-SAT is also NP-complete.
4. Consider two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. Suppose you are asked to check whether there is any subgraph of $G_2$ that is isomorphic to $G_1$. This is called the subgraph isomorphism problem. Prove that this problem is NP-complete. The MAXIMUM COMMON SUBGRAPH (MCG) problem takes as input graphs $G_1$ and $G_2$ as earlier and an integer $z$ as well. It asks you to find subsets $W_1 \subseteq V_1$ and $W_2 \subseteq V_2$ whose deletion leaves behind at least $z$ vertices in each graph and makes the two graphs identical to one another. Prove that MCG is NP-complete as well.

Solution:
To prove that this problem is in NP, we assume that we are given a subgraph of $G_2$ and a mapping $f$ from the vertices of this subgraph to those of $G_1$. In polynomial time, we can check whether this mapping $f$ is an isomorphism. This proves the problem is in NP. To prove that it is NP-hard, we do a reduction from the CLIQUE problem to this one. The CLIQUE problem asks you whether a given graph $G = (V, E)$ has a clique of some size $k$. Note that a clique of size $k$ is a complete subgraph of size $k$, which is commonly denoted as $K(k)$ in graph theory literature. So to check whether $G$ has a clique of size $k$, we give the following input to a SUBGRAPH-ISOMORPHISM routine: $G_2 = G$, $G_1 = K(k)$. This reduction is done in polynomial time. Thus $G$ has a clique of size $k$, IF AND ONLY IF there exists some graph of $G_2$ that is isomorphic to $G_1$, i.e. isomorphic to $K(k)$. If we have a polynomial time algorithm for SUBGRAPH-ISOMORPHISM, then we would have a polynomial time algorithm for CLIQUE. But as CLIQUE is known to be NP-hard, SUBGRAPH-ISOMORPHISM is also NP-hard (our reduction proves that SUBGRAPH-ISOMORPHISM is at least as hard, in a complexity theoretic sense, as CLIQUE. As SUBGRAPH-ISOMORPHISM is in NP (which we proved earlier), we conclude that SUBGRAPH-ISOMORPHISM is NP-complete.

MCG can be proved to be in NP in a manner similar to the previous problem. MCG is NP-hard because we can reduce in polynomial time from SUBGRAPH-ISOMORPHISM (which we just showed to be NP-hard) to MCG as follows. Let $G_1$ and $G_2$ be the inputs to SUBGRAPH-ISOMORPHISM. Then we provide $(G_1, G_2, |V_1|)$ as input to MCG. $G_2$ has a subgraph that is isomorphic to $G_1$ if and only if we can have subsets $W_1 \subseteq V_1$ and $W_2 \subseteq V_2$ whose deletion leaves behind at least $|V_1|$ vertices in each graph and makes the two graphs identical (isomorphic) to one another. In this case note that $W_1 = \phi$ whereas $|W_2| = |V_2| - |V_1|$. Clearly this is a polynomial time reduction. Hence if there were a polynomial time algorithm for MCG, we would have one for SUBGRAPH-ISOMORPHISM.

5. A library keeps track of the number of times a given customer has borrowed so and so book in a database. The records are maintained for each of its $n$ customers and for each of the $m$ books in the library. Given this database, your task is to find the largest possible subset of customers (denoted as $S$) such that no two customers in $S$ have ever borrowed the same book. Prove that this problem is NP-hard.

Solution:
We prove this problem is NP-hard by doing a reduction from graph independent set. Consider a graph $G$ and suppose we want to find the maximum independent set of $G$. To solve this, we create a customer database array $A$ of size $n \times m$ starting from the graph $G$ as follows. (1) Set all entries in the matrix $A$ to be zero. (2) For every vertex in $G$, we create a customer. (3) For any two vertices $v_i$ and $v_j$ in $G$ having an edge between them, we mark $A(i, k) = A(j, k) = 1$ where $1 \leq k \leq m$. In other words, we have forced these customers to borrow one and the same book. This conversion takes polynomial time.

Clearly for any subset $S$ of the customers such that no two customers in $S$ have borrowed even a single common book, we have an independent set of the corresponding vertices in the graph $G$. So if could find the largest of such sets $S$ in polynomial time, we would have a polynomial time algorithm for computing the maximum independent set of a graph $G$. Note that we have reduced from max independent set to this given problem, and thus we are starting from an arbitrary input graph $G$. This proves that our problem is at least as hard as max independent set. If there is no polynomial time algorithm to solve max independent set, then there will no polynomial time algorithm to solve our problem.
Thus our problem is NP-hard.