CS170 Discussion Section 4/27

Problems

Exact 4SAT *In* EXACT 4SAT*, the input is a set of clauses, each of which is a disjunction of exactly four literals, and such that each variable occurs at most once in each clause. The goal is to find a satisfying assignment, if one exists. Prove that* EXACT 4SAT *is* **NP***-complete.*

EXACT 4SAT is a search problem since we can verify any given solution in polynomial time. To show that it is **NP**-complete, we will reduce from 3SAT. We can assume that the 3SAT instance has no clauses with one variable, since those variables can be directly assigned. Let $C_2 = (l_1 \vee l_2)$ and $C_3 = (l_3 \vee l_4 \vee l_5)$ be two clauses have 2 and 3 literals respectively. We can write them as the following equivalent groups of clauses with exactly 4 literals - we need to add one new variable for C_3 and two new ones for C_2. $C_3' = (x \vee l_3 \vee l_4 \vee l_5) \wedge (\overline{x} \vee l_3 \vee l_4 \vee l_5)$ and $C_2' = (y \vee z \vee l_1 \vee l_2) \wedge (\overline{y} \vee z \vee l_1 \vee l_2) \wedge (y \vee \overline{z} \vee l_1 \vee l_2) \wedge (\overline{y} \vee \overline{z} \vee l_1 \vee l_2)$. Any assignment satisfying C_3' must assign either true or false to x, making one of the clauses true- in which case the other clause becomes exactly C_3. A similar argument holds for C_2'.

Node-Disjoint Paths *In the* NODE-DISJOINT PATHS *problem, the input is an undirected graph in which some vertices have been specially marked: a certain number of sources $s_1, ...s_k$ and an equal number of destinations $t_1, ..., t_k$. The goal is to find k node-disjoint paths (paths which have no nodes in common) where the ith path goes from s_i to t_i. Show that this problem is* **NP***-complete.*

For a given 3SAT instance with n variables and m clauses, we create a graph with $m + n$ source sink pairs - one for each variable and one for each clause. For each clause c of the form $(l_1 \vee l_2 \vee l_3)$, we create 6 new vertices $l_1, l_2, l_3, \overline{l_1}, \overline{l_2}, \overline{l_3}$.
We add the edges $(s_c, l_i), (l_i, t_c)$ for $i = 1, 2, 3$ so that the only paths connecting this source-sink pair are the ones that pass through at least one of the literals (not its complement) in the clause. Think of tihs as making the literal true. Also, for all variables x, we connect the occurrences of x in all the clauses in a path and connected its endpoints to s_x and t_x. We also do this for all the occurences of $\overline{x}$. A path from s_x to t_x must contain either all the occurrences of x or all the occurrences of $\overline{x}$.
Given node disjoint paths, we now construct a satisfying assignment. If for variable x, the solution has the path containing all occurrences of x, we assign x = false (and x = true otherwise). Thus, for a path from s_x to t_x we set all the literals in the path to false. Since each clause c must have a path from s_c to t_c containing a literal appearing in the clause, which has not been set to false, the clause must be satisfied. By the same argument, we can also construct a set of paths from a satisfying assignment.

Multiway Cut *In the* MULTIWAY CUT *problem, the input is an undirected graph $G = (V, E)$ and a set of terminal nodes $s_1, s_2, ..., s_k \in V$. The goal is to find the minimum set of edges in E whose removal leaves all terminals in different components.*

(a) *Show that this problem can be solved exactly in polynomial time when $k = 2$.*
This is the same problem as finding a (s_1, s_2) min cut, which can be done by a max flow computation in polynomial time.

(b) *Give an approximation algorithm with ratio at most 2 for the case $k = 3$.*
Find a (s_1, s_2) min cut $E_1 \subseteq E$ using max flow. Suppose s_1 and s_3 fall on the same side of the cut (the other case is symmetric). Compute a (s_1, s_3) min cut $E_2 \subseteq E$ and output $E_1 \cup E_2$. To see this is a 2-approximation, consider the optimal multiway cut E^*: because E^* is both a (s_1, s_2) cut and a (s_1, s_3) cut, we have $|E_1| \leq E^*$ and $|E_2| \leq E^*$. Hence, $|E_1 \cup E_2| \leq |E_1| + |E_2| \leq 2|E^*|$ as required.

Branch and Bound *Devise a branch-and-bound algorithm for the* SET COVER *problem. This entails deciding:*

(i) *What is a subproblem?*
A subproblem is a new instance of SET COVER in which the set of elements is a subset of the original set of elements, and the collection of sets is a subset of the original collection of sets.

(ii) *How do you* `choose` *a subproblem to expand?*
`choose` can be implemented by picking the subproblem with the least number of elements.

(iii) *How do you* `expand` *a subproblem?*
`expand` a subproblem by considering all subproblems obtained by letting one set S into the cover. Then, each subproblem will consist of all the elements not in S and all the original sets except S.

(iv) *What is an appropriate* `lowerbound`*?*
Many `lowerbound` techniques exist. A simple way to bound the cost of a subproblem is to consider the ratio between the number of elements and the cardinality of the largest set. More accurate ways would be to use the greedy algorithm described in Chapter 5. If that algorithm produces a solution of cost ALG on the subproblem, then, by the analysis in the book, the optimal cost for the subproblem must be at lest $\frac{ALG}{\log n}$, where n is the number of elements in the subproblem.

Local Search *Design a local search algorithm for the* SET COVER *problem. Be sure to decide what the* `neighborhood` *of a solution should be, and briefly discuss your decision.*

Starting from a collection of sets S, we can say the neighbors of S are found by adding and removing a small number of sets to the collection: for example, adding up to two sets and removing up to three sets. As long as the limits are constants, the size of the neighborhood will be polynomial in size, so it is possible to efficiently check the neighborhood of a solution.