CS170 Discussion Section 13 : 11/20

Dominating Sets

In an undirected graph $G = (V, E)$, we say $D \subseteq V$ is a *dominating set* if every $v \in V$ is either in D or adjacent to at least one member of D. In the DOMINATING SET problem, the input is a graph and a budget b, and the aim is to find a dominating set in the graph of size at most b, if one exists. Show that DOMINATING SET is NP-complete.

Solution We reduce VERTEX COVER to DOMINATING SET. For simplicity, assume that the graph is connected and has no isolated vertices.

Given a graph $G = (V, E)$ and a number k as an instance of VERTEX COVER, we convert it to an instance of DOMINATING SET as follows. For each edge $e = (u, v)$ in the graph G, we add a vertex a_{uv} and the edges (u, a_{uv}) and (v, a_{uv}). Thus we create a "triangle" on each edge of G. Call this new graph $G' = (V', E')$.

We now claim that a G' has a dominating set of size at most k if and only if G has a vertex cover of size at most k. It is easy to see that vertex cover for G is also a dominating set for G' and hence one direction is trivial.

For the other direction, consider a dominating set $D \subseteq V'$ for G'. For each triangle (u, v, a_{uv}) (corresponding to edge (u, v)), at least one of the three vertices must be in D, since the only neighbors of a_{uv} are u and v. Since we can exchange a_{uv} with u or v, still maintaining a dominating set, we can assume that none of the added vertices (a_{uv})s is in D. Since D must then contain at least one endpoint for every edge, it is also a vertex cover.

Bounded-Degree Independent Set Approximation

Given an undirected graph G in which each node has degree at most d, show how to find an independent set whose size is at least $1/(d + 1)$ times that of the largest independent set.

Solution Initially, let G be the original graph and $I = \emptyset$. Repeat the process below until $G = \emptyset$:

1. Pick any node v and let $I = I \cup \{v\}$.
2. Delete v and all its neighbors from the graph.
3. Let G be the new graph.

Notice that I is an independent set by construction. At each step, I grows by one vertex and we delete at most $d+1$ vertices from the graph (since v has at most d neighbors). Hence there are at least $|V|/(d + 1)$ iterations. Let K be the size of the maximum independent set. Then the previous argument implies that

$$|I| \geq \frac{|V|}{d+1} \geq \frac{K}{d+1}$$

3-SAT

Consider the optimization version of 3-SAT where the objective is to find a variable assignment that satisfies as many clauses as possible. For simplicity, assume that each clause contains exactly three different literals (e.g. $(x_1 \vee \overline{x_2})$ is not allowed).

1. Consider the 3-SAT instance $\overline{x_1} \vee x_2 \vee x_3$. Suppose we set, for $i = 1, 2, 3$, x_i to be 0 or 1 with probability 1/2 independently. What is the probability the instance is satisfied?
2. Give a randomized algorithm with a 7/8-approximation in expectation, i.e. when the input contains n clauses, the expected number of satisfied clauses is $7n/8$.
3. Give a deterministic 8/7-approximation algorithm; the number of satisfied clauses should be at least $7n/8$.

Solution:

1. $\overline{x_1} \vee x_2 \vee x_3$ enumerates to false iff $x_1 = 1$, $x_2 = 0$ and $x_3 = 0$. Since the assignment for x_1, x_2, x_3 is chosen independently, $P[x_1 = 1, x_2 = 0, x_3 = 0] = (1/2)^3 = 1/8$. In other words, $P[\overline{x_1} \vee x_2 \vee x_3 = 1] = 1 - 1/8 = 7/8$.
2. Let $C_1, ..., C_n$ be the clauses. Suppose we set each x_i randomly and independently. Then by the same argument in (1), $P[C_j = 1] = 7/8$. Now by linearity of expectation, the expected number of satisfied C_j is

$$E[\sum_{j=1}^{n} C_j] = \sum_{j=1}^{n} E[C_j] = \sum_{j=1}^{n} P[C_j = 1] = 7n/8.$$

3. We derandomize the algorithm in (2). The idea is to try $x_1 = 0$ and $x_1 = 1$ one at a time, and recurse on the one that gives a higher expected number of satisfied clauses.

   Suppose we set $x_1 = 0$. We calculate

$$N_0 := E[\sum_{j=1}^{n} C_j | x_1 = 0] = \sum_{j=1}^{n} P[C_j = 1 | x_1 = 0]$$

   by computing $P[C_j = 1 | x_1 = 0]$. For instance, if C_j contains the literal $\overline{x_1}$, then $P[C_j = 1 | x_1 = 0] = 1$.

Similarly, we compute $N_1 := E[\sum_{j=1}^{n} C_j | x_1 = 1]$. Note that

$$(N_0 + N_1)/2 = E[\sum_{j=1}^{n} C_j] \geq 7n/8.$$

This implies $N_i \geq 7n/8$ for the bigger N_i. We set $x_1 = i$ for the larger N_i.

The remaining variables are set in an analogous way and one at a time. Each time a new variable is chosen and assigned the value 0 or 1 that gives a bigger expected number of satisfied clauses.

When this procedure terminates, we have set all variables. Moreover, such an assignment satisfies at least $7n/8$ clauses as the expected number of satisfied clauses is at least $7n/8$ at all time (because we always go for the setting with the bigger expected value).

Maximum Acyclic Subgraph

Given a directed graph, return the largest subset of edges that corresponds to a DAG. Find a 2-approximation algorithm, i.e., propose an algorithm that keeps at least half of the number of edges that would have been kept in the optimal solution.

Solution: Take an arbitrary ordering N of the vertices V, and split the edge set E into two sets: $N(u) > N(v)$ and $N(v) > N(u)$, $\forall (u, v) \in E$. The first set keeps all of the edges that preserve the topological ordering N, and the second set keeps all of the edges that preserve N in reverse; return the larger set as the solution.

Both sets of edges independently form valid DAGs, and at least one of them must have at least $\frac{|E|}{2}$ edges. Furthermore, we know that an optimal solution can retain at most $|E|$ edges, so we have a $\frac{1}{2}$ approximation solution.