CS 467/567, Assignment 2

Answers

This assignment is individual. Submit your solutions by email as a single document typeset to PDF. I recommend that you solve the problems by yourself with no external references, but if references are used then they must be provided and also cited in the text.

Problem 1: Approximate Maximum Clique

Let $G = (V, E)$ be an undirected graph. For $k \geq 1$ define $G^{(k)} = (V^{(k)}, E^{(k)})$ such that $V^{(K)} = \{(v_1, v_2, \ldots, v_k) : v_i \in V, 1 \leq i \leq k\}$ and $((v_1, v_2, \ldots, v_k), (w_1, w_2, \ldots, w_k)) \in E^{(k)}$ iff either $(v_i, w_i) \in E$ or $v_i = w_i$ for all $1 \leq i \leq k$.

1. Prove $|C^{(k)}| = |C|^k$, where $C^{(k)}$ and C are the maximum cliques of $G^{(k)}$ and G, respectively.

Answer: The following general property will support the proof:

Lemma 1 *Define the product of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ as $G = (V_1 \times V_2, E)$ such that $((v_1, v_2), (w_1, w_2)) \in E$ iff $(v_1, w_1) \in E_1$ or $v_1 = w_1$, and $(v_2, w_2) \in E_2$ or $v_2 = w_2$. Let C_1, C_2, and C be the maximum cliques for G_1, G_2, and G, respectively. Then $C_1 \times C_2 = C$ (and thus $|C_1| \times |C_2| = |C|$).*

Proof. It is immediate that $C_1 \times C_2$ is a clique of G (and so $C_1 \times C_2 \subseteq C$). Indeed, let $v_1 \in C_1$ and $v_2 \in C_2$. We have $(v_1, w_1) \in E_1$ for all $w_1 \in C_1$ and $(v_2, w_2) \in E_2$ for all $w_2 \in C_2$ (by the definition of a clique) and so $((v_1, v_2), (w_2, w_2)) \in E$ for all $(w_1, w_2) \in C_1 \times C_2$ (by the definition of G); $C_1 \times C_2$ is therefore a clique.

It is also easy to see that $C \subseteq C_1 \times C_2$. Indeed, suppose that there exist a vertex $(v_1, v_2) \in C$ such that $v_1 \notin C_1$ (the case $v_2 \notin C_2$ is similar). Since $v_1 \notin C_1$ there exists a vertex $w_1 \in C_1$ such that $(v_1, w_1) \notin E_1$. If this is so then $((v_1, v_2), (w_1, u))$ cannot be an edge in G no matter what is the value of u (by the definition of G) and so $(v_1, v_2) \notin C$, a contradiction. ■

We now proceed to the main proof by induction over k. For $k = 1$ we have $G^{(1)} = G$ and so $C^{(1)} = C$. It is therefore immediate that $|C^{(1)}| = |C|^1$. Note now that the Cartesian product is associative within a natural bijection and so $G^{(k)} = G^{(k-1)} \times G$ (with the product of two graphs as defined in Lemma 1). Therefore $|C^{(k)}| = |C^{(k-1)}| \times |C|$ by Lemma 1. By inductive assumption we also have that $|C^{(k-1)}| = |C|^{k-1}$, so $|C^{(k)}| = |C|^{k-1} \times |C| = |C|^k$, as desired.

---

2. Argue that the existence of an approximation algorithm for finding the maximum clique with a constant approximation ratio implies the existence of a polynomial-time approximation scheme for finding the maximum clique.

Answer: Let A be an polynomial ρ-approximation algorithm for clique. We then use A as follows: for a fixed k and an input graph G we construct $G^{(k)}$, we launch A on $G^{(k)}$ obtaining $C^{(k)}$, and then we obtain C from $C^{(k)}$. Let $C^{(k)*}$ and C^* be the maximal clique for $G^{(k)}$ and G, respectively. We have:

$$\begin{aligned} \frac{|C^{(k)*}|}{|C^k|} &\leq \rho \qquad \text{by the definition of } A\text{, therefore:} \\ \frac{|C^*|^k}{|C|^k} &\leq \rho \qquad \text{by Question 1, therefore:} \\ \frac{|C^*|}{|C|} &\leq \rho^{1/k} \end{aligned}$$

The end result is a $\rho^{1/k}$-approximation scheme (which has an approximation ratio arbitrarily close to 1 for an arbitrarily large k).

Let now n be the number of vertices in G (and so the number of edges is less than n^2). A trivial algorithm for computing $G^{(k)}$ takes $O(n^{O(k)})$ time, which is polynomial in n (but not in k). A takes polynomial time by definition. Extracting C out of $C^{(k)}$ is also immediate: we just scan all the vertices $(v_1, v_2, \ldots, v_k) \in C^*$ and add all the v_i to the *set* C. This takes linear time in $|C^{(k)}|$ and so once more $O(n^{O(k)})$ time. In all, the running time is polynomial in n and exponential in k.

Overall we have an approximation scheme that is polynomial (but not fully polynomial).

Note that the answer to this question does not say anything about how easy to approximate Clique is. It is already well known (since at least 1978) that a fully polynomial-time approximation scheme cannot not exist, but how about a polynomial-time approximation scheme? The question at hand merely says that there cannot be a middle ground: the problem either has a polynomial-time approximation scheme or does not have any polynomial-time approximation algorithm with a constant approximation ratio. To date the best known approximation algorithm for Clique has an approximation ratio of $O(n(\log\log n)^2/\log^3 n)$, which is not constant and so does not imply a polynomial-time approximation scheme. There appear to be a relatively recent (2000s) result showing that the approximation ration cannot be better than $O(n^{1-\varepsilon})$ for any $\varepsilon > 0$, but I did not have the time to check it out.

Problem 2: Linear Inequality Feasibility

I claimed in class that the original linear programming problem (an optimization problem) turns out to be no harder than the apparently simpler (decision) problem of determining whether a

certain simplex is empty (that is, just finding a point in the simplex, not necessarily the optimal one). This question will prove my claim.

Given a set of m linear inequalities over n variables, the linear inequality feasibility problem asks whether there exists an assignment of the variables that satisfies all the linear inequalities simultaneously.

1. Prove that we can use an algorithm for linear programming to solve linear inequality feasibility problems. The number of variables and constraints used in the linear programming problem must be polynomial in n and m, and the slowdown must be polynomial.

ANSWER: The linear programming algorithm will return the optimal point in the simplex. In the linear inequality feasibility problem we are happy with *any* point in the simplex; as long as the simplex is not empty we have a positive answer. Running a liner programming algorithm with the objective function 0 (or indeed any other constant) will do exactly that. The input for the linear programming algorithm features n variables and m constraints.

---

2. Prove that we can use an algorithm for the linear inequality feasibility problem to solve linear programming problems. The number of variables and linear inequalities must be polynomial in the number of variables and constraints of the linear program, and the slowdown must be (you guessed it) polynomial.

ANSWER: Assume that the linear inequality feasibility algorithm will produce a point in addition to the yes answer. This is a reasonable assumption given that such a point needs to be computed to justify a positive answer.

Now we run the linear inequality feasibility algorithm on each possible pair of constraints and record all the points thus found. This results in an m^2 (polynomial) slowdown and no more than m^2 points. We then compute the objective function for all the points and return the one with a maximum such an objective function. Each such a computation takes $O(n)$ time. That the correct answer is returned is given by the fact that the optimum is always a vertex of the simplex. The overall slowdown is quadratic in m and linear in n.
