CS 467/567, Assignment 2

Wentao Lu (002276355), Yi Ren (002212345)

1 Problem 1: Approximate Maximum Clique

1. Let $V' = \{(v_1, v_2, \ldots, v_k) | v_i \in C, 1 \leq i \leq k\} \subseteq V^{(k)}$, where C is the maximum clique of G, so every pair of vertices (v_i, v_j) is adjacent, clearly V' is a clique of $G^{(k)}$. Since each element v_i in the k-tuple $(v_1, v_2, \ldots, v_k)$ comes from C, the size of V' is $|C|^k$. Therefore, the size of the maximum clique of $G^{(k)}$ is at least $|C|^k$. We will prove by induction that this is indeed the size of the maximum clique.

   In the base case, when $k = 1$, $G^{(1)} = G$, we have $|C^{(1)}| = |C|^1$. Now suppose that $|C^{(k)}| = |C|^k$ is true for $1 \leq i \leq k$. Let $C^{(k+1)} = \{(p_i, v_1, v_2, \ldots, v_k)\}$ where p_i is some vertex in G.

   Since the adjacency condition of $(v_1, v_2, \ldots, v_k)$ and $(w_1, w_2, \ldots, w_k)$ only requires the adjacency of v_i and w_i, the choice of vertices at index i is independent of vertices at any other index. Hence, if we remove the prefix vertex p_i from every tuple in $C^{(k+1)}$, the remaining k-tuples must form a maximum clique of $G^{(k)}$, otherwise we can always do better. By induction hypothesis, it has size $|C|^k$.

   In order to make $C^{(k+1)}$ a clique of $G^{(k+1)}$, we must choose a set of prefixes $\{p_i\} \subseteq V$ such that every pair of vertices (p_i, p_j) is connected by an edge. Moreover, to make it a maximum clique, we also want to maximize the number of prefixes we choose. In other words, we are trying to find a maximum set of mutually adjacent vertices from G and prepend them to every k-tuple in $C^{(k)}$, so we should choose the set of prefix vertices $\{p_i\} = C$. Therefore, the size of $C^{(k+1)}$ is $|C|^k \times |C| = |C|^{k+1}$.

2. Given a c-approximation algorithm for finding the maximum clique, we can apply it on $G^{(k)}$ and obtain a clique of size n, while the true maximum clique of $G^{(k)}$ has size $|C|^k$.

   By definition of an approximation ratio, $\frac{|C|^k}{n} \leq c$, so we have $\frac{|C|}{n^{1/k}} \leq c^{1/k}$. For any $\varepsilon > 0$, we can choose a positive integer $k > \log_{1+\varepsilon} c$, then $\frac{|C|}{n} \leq \frac{|C|}{n^{1/k}} \leq c^{1/k} < 1 + \varepsilon$.

   This approximation algorithm is polynomial in the size of input, so we have a polynomial-time approximation scheme for finding the maximum clique.

2 Problem 2: Linear Inequality Feasibility

1. Given a set of linear inequalities, we can simply formulate the equality feasibility problem as a linear program. First, we can set the objective function to be a constant such as 0 because there's nothing to maximize or minimize in this case. Next, we need to transform those linear inequalities into their slack form, and then apply the simplex algorithm. If the linear program returns a feasible solution, it implies that all the linear inequalities can be satisfied at the same time. Otherwise, those linear inequalities are not feasible. The variables and

constraints in the linear program are exactly the same as in the equality feasibility problem so they are polynomial in n and m.

2. Given a linear program in standard form, we want to maximize the objective function $\Sigma_{j=1}^{n} c_j x_j$ with n variables, subject to m constraints $\Sigma_{j=1}^{n} a_{ij} x_j \leq b_i, 1 \leq i \leq m$. Suppose we have an algorithm for the linear inequality feasibility problem, then we can apply it on the m constraints of the linear program. If the algorithm fails to find a solution for the linear inequality feasibility problem, it means that those inequalities can not be satisfied, so the linear program should return infeasible. If the algorithm does find an assignment of the n variables that satisfies all the linear inequalities, then the linear program is also feasible.

   In order to solve the linear program and maximize the objective function, in each step we first calculate the value of our objective function using the assignment values returned by that algorithm, then we update the linear program by introducing a new constraint. For example, if the algorithm finds an assignment $\overline{x}$ which gives an objective function value of $\Sigma_{j=1}^{n} c_j x_j = k$, we will add another constraint $\Sigma_{j=1}^{n} c_j x_j > k$ to the original linear program. As long as this updated linear program is feasible, it must be equivalent to the original linear program because both the objective function and all other constraints are not affected. If we iteratively run this step, we are getting closer to the optimal solution. We will repeat this step until the algorithm detects that the updated constraints are not feasible anymore, in which case we stop and return the last calculated objective function value as the optimal solution.

   Whenever the algorithm returns a feasible assignment so that all constraints are satisfied, it represents a point that's somewhere in the simplex. The basic idea is to update that point until it reaches a local maximum, which is also a global maximum because the simplex is convex. Since we are only adding 1 more constraint in each step, we can optimistically assume that the number of linear inequalities is polynomial in the number of constraints. If that's not true, we can modify the constraint to be $\Sigma_{j=1}^{n} c_j x_j > k + \alpha$, where α is a reasonable step size to choose in different trials. Unless the original linear program has an unbounded solution, we will eventually converge to an optimal solution.