CS170–Fall 2022 — Homework 12 Solutions

Deming Chen `cdm@pku.edu.cn`

Last Modified: January 18, 2023

Collaborators: NONE

2 $\sqrt{n}$ coloring

(a) We prove this proposition by induction on the number of vertices.

When $|V| = 1$, there is no edges, so graph G is $(\Delta+1)$-colorable.

Suppose a graph which has $(n-1)$ vertices is $(\Delta+1)$-colorable, if it is of maximum degree Δ. Now consider graph G with n vertices. Say v is one of G's vertices. Removing v and edges meeting v we get G'. G' has $(n-1)$ vertices and is of maximum degree Δ, so G' is $(\Delta+1)$-colorable by induction hypothesis. Color G' with $(\Delta+1)$ colors so that for any edge $\{u, w\}$, u and w have different colors. Since v is connected to at most Δ vertices in G', we can assign a different color to v than any vertex adjacent to v. So, G is $(\Delta+1)$-colorable.

(b) In a valid 3-coloring of graph G, any neighbor of v has different color from v. So, they are assigned at most 2 colors. That is, that the graph induced on the neighborhood of v is 2-colorable.

(c) If G is a graph of maximum degree Δ, part (a) gives an algorithm to color its vertices using $(\Delta+1)$ colors. This algorithm takes time $\mathcal{O}(m+n)$.

Algorithm 1. $(\Delta+1)$**-Coloring**

```
procedure (Δ+1)-color(G)
Input:  a graph G of maximum degree Δ
Output:  a valid coloring of G's vertices using at most (Δ+1) colors

While some vertex v has not been colored:
    color v with different color from all v's neighbors
```

Based on Algorithm 1, we can build a polynomial time algorithm (Algorithm 2) which outputs a valid coloring of G's vertices using $\mathcal{O}(\sqrt{n})$ colors. For convenience, we label different colors as $0, 1, 2, \cdots$.

Since Algorithm 2 needs at most n recursions, and each recursion takes polynomial time, the total runtime is polynomial.

Now, we show that this algorithm using no more than $3\sqrt{n}$ colors. We prove it by induction on n. When $n \leqslant 3$, for any vertex v, $\deg(v) \leftarrow n-1 \leqslant 3\sqrt{n}-1$, so

Algorithm 2. $\sqrt{n}$**-Coloring**

```
procedure √n-color(G, k)
Input:  a graph G with n vertices; k is a non-negative integer
Output:  a valid coloring of G's vertices using color m, m+1, ···

v ← one of G's highest-degree vertices
If deg(v) ⩽ 3√n − 1:
    return (Δ+1)-color(G)
assign v color m
assign v's neighbors color m+1 or color m+2
G̃ ← remove from G v, v's neighbors and edges meeting these vertices
return √n-color(G̃, m+3) and the assignment of v and v's neighbors
```

$(\Delta+1)$-color will give an valid coloring using no more than $3\sqrt{n}$ colors. Suppose this proposition holds for all 3-colorable graph with less than n vertices ($n \geqslant 4$). For a 3-colorable n-vertex graph G, if G is of maximum degree $3\sqrt{n}-1$, then $(\Delta+1)$-color will give an valid coloring using no more than $3\sqrt{n}$ colors. Otherwise, $\tilde{G}$ has at most $n-3\sqrt{n}$ vertices. By induction hypothesis, the number of colors used in Algorithm 2 is no more than

$$3\sqrt{n-3\sqrt{n}}+3 \leqslant 3\sqrt{n}.$$

So, the proposition holds for graph G.

3 Randomization for Approximation

(a) The algorithm is to assign each variable `true` or `false` randomly. Suppose we have n clauses, $c_1, c_2, \cdots, c_n$, and, without loss of generality, $c_1, c_2, \cdots, c_k$ can be satisfied in the maximum satisfaction. The expectation of the number of satisfied clause is

$$\begin{aligned} & \mathbb{E}[\#\text{ of satisfied clause among } c_1, c_2, \cdots, c_n] \\ = & \sum_{i=1}^{n} \mathbb{E}[\#\text{ of satisfied clause in } \{c_i\}] \\ = & \sum_{i=1}^{n} \mathbb{P}[c_i \text{ is satisfied}] \\ = & \frac{7}{8} n \geqslant \frac{7}{8} k. \end{aligned} \tag{1}$$

(b) In (a), we have shown that the randomized algorithm satisfies at least a fraction of 7/8 clauses in expectation. So, $OPT_I \geqslant 7/8$. Consider the following instance:

$$(x \vee y \vee z)(x \vee y \vee \bar{z})(x \vee \bar{y} \vee z)(x \vee \bar{y} \vee \bar{z})(\bar{x} \vee y \vee z)(\bar{x} \vee y \vee \bar{z})(\bar{x} \vee \bar{y} \vee z)(\bar{x} \vee \bar{y} \vee \bar{z}).$$

Apparently, exactly one of 8 clauses cannot be satisfied. $OPT_I = 7/8$.

As a result, $\min OPT_I \geqslant 7/8$.

4 Independent Set Approximation

The procedure is to repeatedly select a vertex of minimum degree and remove the vertex and its neighbors and all edges meeting these removed vertices until no vertex remains.

Suppose $|V| = n$. Every iteration we remove at most $(d+1)$ vertices. We remove at most $|V'| \cdot (d+1)$ vertices, so $|V'| \cdot (d+1) \geqslant n$. We have

$$|V'| \geqslant n/(d+1) \geqslant \frac{1}{d+1} \cdot (\text{the size of the largest independent set}). \tag{2}$$

5 Coffee Shops

(a) We have mn variables: $x_{ij}(i = 1, 2, \cdots, m, j = 1, 2, \cdots, n)$. x_{ij} means the number of coffee shops we set up within block ij.

(b) Our objective function is

$$\min \sum_{i=1}^{m} \sum_{j=1}^{n} x_{ij} r_{ij}. \tag{3}$$

(c) The constrains are

$$\begin{cases} 0 \leqslant x_{ij} \leqslant 1 \\ x_{ij} + x_{i(j+1)} + x_{i(j-1)} + x_{(i+1)j} + x_{(i-1)j} \geqslant 1 \end{cases} \tag{4}$$

where $x_{0j} = x_{(m+1)j} = x_{i0} = x_{i(n+1)} = 0$ $(i = 1, 2, \cdots, m, j = 1, 2, \cdots, n)$.

(d) If x_{ij} is less than 0.2, we round x_{ij} down to 0; otherwise, we round it up to 1.

(e) We use $\tilde{x}_{ij}$ to denote the optimal solution of the real-valued LP, and $\bar{x}_{ij}$ to denote the optimal solution of integer LP. Also, we define the rounding rules in (d) as

$$f(x) = \begin{cases} 1, & \text{if } x \geqslant 0.2, \\ 0, & \text{if } x < 0.2. \end{cases} \tag{5}$$

So, $f(x) \leqslant 5x$. Our algorithm gives the total cost as

$$\sum_{i=1}^{m} \sum_{j=1}^{n} f(\tilde{x}_{ij}) r_{ij} \leqslant 5 \sum_{i=1}^{m} \sum_{j=1}^{n} \tilde{x}_{ij} r_{ij} \leqslant 5 \sum_{i=1}^{m} \sum_{j=1}^{n} \bar{x}_{ij} r_{ij}. \tag{6}$$

Therefore, the approximation ratio is 5.

6 One-Sided Error and Las Vegas Algorithms

(a) Suppose x is an RP problem and R is its corresponding randomized algorithm so that for any instance I of x, $R(I)$ is either "YES" or "NO". Consider $R(I)$ as a deterministic algorithm $A(I, r)$ where I is the instance and r is the result of the "coin flips" which the algorithm uses for its randomness. Given I and r, $A(I, r)$ can be computed in polynomial time. Also, we can consider a *search problem* specified by A where given an instance I we search for a solution r so that $A(I, r) =$ YES or output NO if no solution exists.

(b) Suppose $\mathcal{L}$ is the Las Vegas algorithm which runs in expected polynomial time. Given an instance I, the runtime T is a random variable. By Markov's inequality, $\mathbb{P}(T > 2E(T)) \leqslant 1/2$. So we can construct a polytime randomized algorithm as follows: run $\mathcal{L}(I)$; if the runtime exceeds $2E(T)$, stop and output "NO". This algorithm gives the correct answer when the correct answer is "NO", but only gives the correct answer with probability greater than $1/2$ when the correct answer is "YES".

As a result, $ZPP \subset RP$.