2 $\sqrt{n}$ coloring

(a) Let the color palette be $[\Delta + 1]$. While there is a vertex with an unassigned color, give it a color that is distinct from the color assigned to any of its neighbors (such a color always exists since we have a palette of size $\Delta + 1$ but only at most Δ neighbors).

(b) In an induced graph, we only care about the direct vertices adjacent to v and v itself. In any 3 - coloring of G, v must have a different color from its neighborhood and therefore the neighborhood must be 2 - colorable.

(c) While there is a vertex of *degree* $\leq \sqrt{n}$, choose the vertex v, pick 3 colors, use one color to color v, and the remaining 2 colors to color the neighborhood of v (2 - coloring is an easy problem). Never use these colors ever again and delete v and its neighborhood from the graph. Since each step in the while loop deletes at least $\sqrt{n}$ vertices, there can be at most $\sqrt{n}$ iterations. This uses only $3(\sqrt{n}+1)$ colors. After this while loop is done we will be left with a graph with max degree at most $\sqrt{n}$. We can take $\sqrt{n}+1$ color with fresh colors for this using the greedy strategy from the solution to part b. The total number of colors used is $\mathcal{O}(\sqrt{n})$.

3 Randomization for Approximation

(a) Consider randomly assigning each variable a value. Let X_i be a random variable that is 1 if clause i is satisfied and 0 otherwise. We can see that the expectation of X_i is $\frac{7}{8}$.

Note that $\sum X_i$ is the total number of satisfied clauses. By linearity of expectation, the expected number of clauses satisfied is $\frac{7}{8}$ times the total number of clauses. Since the optimal number of satisfied clauses is at most the total number of satisfied clauses, a random assignment will in expectation have value at least $\frac{7}{8} \cdot k$.

(b) Our randomized algorithm satisfies fraction 7/8 of clauses in expectation for any instance. So any instance must have a solution that satisfies at least fraction 7/8 of clauses (a random variable must sometimes be at least its mean).This lower bound is tight: Consider an instance with 3 variables and all 8 possible clauses including these variables. Then any solution satisfies exactly 7 clauses.

4 Independent Set Approximation

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

1. Pick an arbitrary node v in G 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. Since $K \leq |V|$, we can use the previous argument to get:

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

5 Coffee Shops

(a) There is a variable for every block x_{ij} , i.e., $\{x_{ij} \mid i \in \{1, \ldots, m\}, j \in \{1, \ldots, n\}\}$. This variable corresponds to whether we put a coffee shop at that block or not.

(b) $\min \sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij}$. Alternatively, $\min t$ is correct as well as long as the correct constraint is added.

(c) (i) $x_{ij} \geq 0$: This constraint just corresponds to saying that there either is or isn't a coffee shop at any block. $x_{ij} \in \{0, 1\}$ or $x_{ij} \in \mathbb{Z}_+$ is also correct.

(ii) For every $1 \leq i \leq m, 1 \leq j \leq n$:

$$x_{ij} + x_{(i+1),j} \mathbb{1}_{\{i+1 \leq m\}} + x_{(i-1),j} \mathbb{1}_{\{i-1 \geq 1\}} + x_{i,(j+1)} \mathbb{1}_{\{j+1 \leq n\}} + x_{i,(j-1)} \mathbb{1}_{\{j-1 \geq 1\}} \geq 1$$

This constraint corresponds to that for every block, there needs to be a coffee shop at that block or a neighboring block.

$\mathbb{1}_{\{i+1 \leq m\}}$ means "1 if $\{i+1 \leq m\}$, and 0 otherwise". It keeps track of the fact that we may not have all 4 neighbors on the edges, for instance.

(iii) If the objective was $\min t$, then the constraint $\sum_{i=1}^{m} \sum_{j=1}^{n} r_{ij} x_{ij} \leq t$ needs to be added.

(d) Round to 1 all variables which are greater than or equal to 1/5. Otherwise, round to 0. In other words, put a coffee shop on (i, j) iff $x_{i,j} \geq 0.2$.

(e) Using the rounding scheme in the previous part gives a 5-approximation. Notice that every constraint has at most 5 variables. So for every constraint, there exists at least one variable in the constraint which has *value* $\geq 1/5$ (not everyone is below average). The total cost of the rounded solution is at most $5 \cdot$ LP-OPT, since $r_{ij} \leq 5 r_{ij} x_{ij}$ for any x_{ij} that gets rounded up, and the other i, j pairs contribute nothing to the cost of the rounded solution. Since Integral-OPT $\geq$ LP-OPT (the LP is more general than the ILP), our rounding gives value at most 5 LP-OPT ≤ 5 Integral-OPT. So we get a 5-approximation.

6 One-Sided Error and Las Vegas Algorithms

(a) Assume we have some problem that is in RP and let A be an algorithm that solves this problem. If A is given an input x such that the correct answer given input x is 'YES', then A will return 'YES' will probability greater than 1/2. We want to use A to construct an NP algorithm for x.

Randomized algorithms run like regular algorithms, but will occasionally use random coin-flips to make decisions. For any randomized algorithm B, we can build a deterministic algorithm B' that takes the outcome of those coin-flips beforehand and runs the same computation as B. Let A' be that deterministic algorithm for A.

Since A is an RP algorithm, there must be a poly-size sequence of coin-flip outcomes $\vec{b} = b_1, \ldots, b_k \in \{0, 1\}^k$ such that A returns 'YES' given input x and outcomes $\vec{b}$. If we treat $\vec{b}$ as the 'solution' to x, we can see that A' is an NP algorithm for the given problem. Thus every problem in RP is in NP.

(b) Let A be any ZPP algorithm and assume A runs in expected time at most n^k for some constant k. We construct an RP algorithm A' that given input x of size n, does the following:

- Run A for $2n^k$ steps , then stop.
- If A returns 'YES' in that time , return 'YES'.
- If A returns 'NO' in that time , return 'NO'.
- Otherwise , return 'NO'.

How do we know A' is an RP algorithm? First of all, if the correct answer on input x is 'NO', then either A will stop and A' will say 'NO' or A will not stop and A' will still say 'NO'. So A' will always give the right answer when the right answer is 'NO'.

Now assume the correct answer is 'YES'. If A stops in time, A' will give the correct answer. But if A does not stop in time, then A' will give the wrong answer. So we want to bound the probability that A does not stop in time.

Let X be a random variable representing the number of steps A will take on input x. We know that $E[X] \leq n^k$. By Markov's inequality this gives

$$\Pr[X \geq a] \leq \frac{n^k}{a}$$

If we set $a = 2n^k$, this gives

$$\Pr\left[X \geq 2n^k\right] \leq \frac{n^k}{2n^k} = \frac{1}{2}$$

Thus if the correct answer on input x is 'YES', A' will give the correct answer with probability greater than 1/2.

So A' is an RP algorithm.