Department of Mathematics and Computer Science

Homework I

Benedetto Buratti Magnus Malmström

Eindhoven, Mars 2017

Chapter 1

Exercises

1.1 Exercise one

G is a 3-regular 2-edge connected graph (i.e. you have to delete at least 2 edges from G to make it disconnected). Prove that G has a perfect matching.

1.1.1 Solution

Let $G = (V, E)$ be a 3-regular 2-edge connected graph, and consider $S \subseteq V$. We want to show that $o(G - S) \leq |S|$.

Let $H_1, ..., H_t$ be the odd components of $G - S$, and let m_i be the numbers of edges from S to H_i. Then

$$\sum_{v \in V(H_i)} d(v) = 3|V(H_i)| \tag{1.1}$$

where $3|V(H_i)|$ is odd. Since G is 3-regular.

$$\sum_{v \in V(H_i)} d(v) = m_i + 2|E(H_i)| \tag{1.2}$$

This gives us

$$m_i = \sum_{v \in V(H_i)} d(v) - 2|E(H_i)| \tag{1.3}$$

Since G i 2-edged it has no cut edge and $m_i \neq 1$. Thus, since m_i is odd $m_i \geq 3$. It follows that

$$o(G - S) = t = \sum_{i=1}^{t} 1 \leq \frac{1}{3} \sum_{i=1}^{t} m_i \leq \frac{1}{3} \sum_{v \in V(H_i)} d(v) = |S| \tag{1.4}$$

QED

1.2 Exercise two

Consider a set N of elements. We are also given 4^{n1} sets $S_i \subset N$ for $1 \leq i \leq 4^{n1}$ each of size exactly n. Prove that there exists a colouring of the elements of N with 4 colors such that each set S_i is not monochromatic i.e. not all elements of S i have the same color

(A) The first strategy one might think of is this: randomly color each element of N and show that the expected number of monochromatic sets is less than 1. The probabilistic method assures us that some solution exists. Why does this method not work?

(B) Prove the statement. (Hint: Strategy 1: what goes wrong in the first part? Prove that there is some solution which has at most one of the following configuration: some set S i is all red, another set S j has one red point and all other points green and S i , S j intersect in only one point. Once you have such a solution, fix it to get the colouring we want. Strategy 2: You could instead also show that when not all sets are disjoint, the probability of a random colouring having a monochromatic set is strictly smaller than 1.)

1.2.1 Solution A

In this section we are going to apply the probabilistic method straightforwardly. We are going to show that the probabilistic itself is not sufficient in order to prove the existence of a 4-colouring for such graph.

We are given a set U with N elements and 4^{n-1} subsets $S_i \subset U$ each of size n. In order to apply the probabilistic method we first randomly and independently colour each element of the set U with a probability p, where p is equal to $\frac{1}{4}$ for each one of the four colors.

Now we continue by defining for each subset a random variable X_i defined as follows:

$$f(n) = \begin{cases} 1, & \text{if } n \text{ monochromatic} \\ 0, & \text{if } n \text{ not monochromatic} \end{cases} \tag{1.5}$$

Since every subset S_i is composed by exactly by n element its probability to be color specific monochromatic, let's say blue-monochromatic, is $P_r[S_i \text{ is } bluemono] = (\frac{1}{4})^n$. Since we have four different colours we get: $P_r[S_i\ mono] = (\frac{4}{4^n}) = \frac{1}{4^{n-1}}$

Now we are able to compute the expectation that U has at least one monochromatic set. Since we have 4^{n-1} subsets we get that

$$\mathbf{E}[X] = \sum_{n=1}^{4^{n-1}} \mathbf{E}[X_i] = \frac{4^{n-1}}{4^{n-1}} = 1 \tag{1.6}$$

This result implies that the probabilistic method is not able to give us any direct proof that a 4-colouring exist, since the expectation is not strictly less than 1.

1.2.2 Solution B

In the previous section we discussed that the probabilistic method is not able to give us not assurance about the existence of a solution since there could be two situation which lead us to get and $E[X] = 1$, or:

- the random variables X_i get always value 1
- the random variables X_i sometimes assumes values bigger and smaller than 1 such that we have an expected value equal to 1

To prove the existence of a 4-colouring with no monochromatic subset we show an example that proves the second statement holds.

Indeed, given the integer-valued nature of the problem, if we find a solution which has $\sum_{n=1}^{4^{n-1}} \mathbf{E}[X_i] >$ 1 and a probability bigger than zero, it will imply that there must exist with a solution with $\sum_{n=1}^{4^{n-1}} \mathbf{E}[X_i] = 0$ (proving our statement).

The most naive way to do this is to take into consideration the coloring of the whole graph with one single color, which namely has probability $P_r[\forall v_i inV = red] = \frac{1}{4^N}$m which is bigger than zero. This little tweaking of the probabilistic method implies necessarily that there exist exist a 4-colouring without any monochromatic subset.

1.3 Exercise three

Solve 100-CNF-Sat in $O^*((2-\epsilon)^n)$ time for some $\epsilon > 0$, where n denotes the number of variables.

1.3.1 Solution

To solve this problem we will use a branching algorithm. First pick a clause of maximum size and branch on all assignments of its variables satisfying it. If the we pick a clause of k literals we recurse on the CNF-formula setting those k literals to all 2^k assignments except the one that we know is false. This result in a $2^k - 1$ new recursive calls on formula with at most $n - k$ variables. If there is no clause of length k but a clause $m < k$ then recurse the CNF-formula with those $2^m - 1$ assignments of most m literals. So we can use the the folowing recurrence for the number of leaves of the branching tree $T(0) = 1$ and for $n > 0$

$$T(n) = max[(2^k - 1)T(n-k), (2^m - 1)T(n-m)]. \tag{1.7}$$

Setting $T(n) = (2^k - 1)^{n/k} < (2-\epsilon)^n$ for some $\epsilon > 0$ works since

$$max(2^k - 1)(2^k - 1)^{(n-k)/k}, (2^m - 1)(2^k - 1)^{(n-m)/k} \leq (2^k - 1)^{n/k}. \tag{1.8}$$

This holds for any k and espesially $k = 100$.

1.4 Exercise four

Give an $O((k+1)^{4k})$-time algorithm that takes as input n points in the plane R^2 and an integer k, and determines whether there exist k straight lines such that every point is on some line. Hints: First look at $k+1$ points that are on one line to find a reduction rule. Second conclude something if n is too large when compared with k^2 and your reduction rule does not apply. Third design an $O(n^{2k})$ time algorithm.

1.4.1 Solution

Let S be the set of points. A naive case of our problem is when we have that $|S| \leq 2k$, where S is the set of all points, since it obviously exist a covering of the points using k lines

For more general situations we use the following lemmas (of which we reported the proof at the end of the exercise):

- **Lemma:** If a set S of n points can be covered with k lines (k minimal or not), then: any subset of S at least $k^2 + 1$ points must contain at least k + 1 collinear points
- **Corollary:** If in any subset of S containing at least k^{2+1} points we do not find $k+1$ collinear points, we can conclude that S cannot be covered with k lines.

Taking into account the previous results, we designed an algorithm that checks in two steps if the k lines cover exist:

1. look for k+1 points which lies on the same line and reduce them to 1 point. Keep searching until the remaining point set cardinality is less than k^2+1. If such reduction is not possible, than it do not exist k lines set to cover the points (corollary)
2. on the remaining $k^2 + 1$ points check if they are covered using brute force

Defined those concepts, the algorithm follows:

Algorithm 1 $|S| \geq K^2 + 1 (reducing - rountine)$

procedure REDUCE
    while $|S| \geq K^2 + 1$ **do**
        $select\ U \subset S : |U| = k^2 + 1$
        $reduction = false$
        for $\forall Q \subset U : |Q| = k + 1$ **do return** false
            if $\forall P_i \in Q$ lie on a line l **then**
                reduce the $k + 1$ points to one point.
                if $|S| \leq K^2$ **then**
                    return BRUTE
                $k -= 1.$
                $reduction = True.$
        if reduce == False **then**
            return False

In case we have successfully reduced our set to a cardinality of K^2, we can use the $bruteforce-routine$ on the reduced point set S.

Algorithm 2 $|S| \leq k^2 - (bruteforce - routine)$

procedure BRUTE
    for $\forall L = l_1, l_2, ...l_k$ **do**
        for $p \in S$ **do**
            for $\forall l \in L$ **do**
                if $p \in l$ **then**
                    BREAK
                return False
    return True

The overall algorithm have a running time of $\mathcal{O}^*((k+1)^{4k})$ since the REDUCE part takes $\mathcal{O}^*(k^{2k})$ (we have $\binom{K^2+1}{k+1} \leq K^{2k}$ combinations of class k and the check part is computable in linear time) and the BRUTE part $\mathcal{O}^*(k^k)$: (we have $\binom{K^4}{k} \leq K^{4k}$ combinations of class k the check part is computable in cubic time)

1.4.2 Lemma Proof

Suppose there is a subset R S containing at least k^{2+1} points, but not containing k+1 collinear points. Then each of the k covering lines must contain at most k points in R. Hence with these at most k covering lines, each containing at most k points, we can cover at most k 2 points. Thus we cannot cover R (nor any superset of R, like S) with the k lines. This contradicts the assumption that S can be covered with k lines. [*Covering a Set of Points with a Minimum Number of Lines*, M.Grantson, C.Levcopoulos]

1.5 Exercise five

A triangle of a graph $G = (V, E)$ is a triple $u, v, w \in V$ such that $(u, v), (v, w), (u, w) \in E$. A triangle partition is a partition of V into triangles, e.g., a set of triangles $T_1, ..., T_{n/3}$ such that $T_i \cap T_j = \emptyset$ and $\cup_i T_i = V$

- Give an algorithm that determines whether there is a triangle partition of a graph on n vertices in $O^*(2^n)$ time.
- If your algorithm uses $O^*(2^n)$ time and polynomial space you get 5 additional points.

1.5.1 Solution

First see if n is divisible with three, if it is return false. Then to solve this problem we will use inclusion/exclusion. Let $V = 1, ..., n$. Then apply inlcusion/exclusion where U be all triangles $T_1, ..., T_{n/3}$ that satisfy $|T_1| + ... + |T_{n/3}| = n$ and $\overline{P_i} \subseteq U$ is those triangles avoiding vertex i. This gives us:

$$|\bigcap_{1}^{n} P_i| = \sum_{F \subseteq 1,..,n} (-1)^{|F|} |\bigcap_{i \in F} |\overline{P_i}| \tag{1.9}$$

And $|\bigcap_{i \in F} |\overline{P_i}|$ are thous triangles avoiding F. $\bigcap_1^n P_i$ is the sets that lies in non of the $\overline{P_i}$.

Now let Q denote all the triangles and let $\hat{f^{(l)}}(Y) = \sum_{\subseteq Y} f(S) \text{for} Y \subseteq N$ where N is the set of vertices and

$$f^{(l)}(S) = \begin{cases} 1 & \text{if } S \in Q \text{ and } |S| = l \\ 0 & \text{otherwise} \end{cases} \tag{1.10}$$

Once all of these are calculated you can calculate $|\bigcap_{i \in F} |\overline{P_i}|$ in for any fixed $F \subseteq N$ by dynamic programming. Define $g(j, m)$ to be the number of j-tuples $(S_1, ..., S_j)$ for which $S_c \cap F = \emptyset$ $c = 1, ..., j$ and $|S1| + \ + |Sj| = m$ as

$$g(j, m) = \sum_{l_1 + ... + l_j} \prod_{c=1}^{j} \hat{f^{(l_c)}}(N\ F). \tag{1.11}$$

Then $|\bigcap_{i \in F} |\overline{P_i}| = g(k, n)$, which we can compute by

$$g(j, m) = \sum_{l=0}^{m} g(j-1, m-l) \hat{f^{(l)}}(N\ F). \tag{1.12}$$

with $g(1, m) = \hat{f^{(m)}}(N\ F)$. Then finally sum all the $|\bigcap_{i \in F} |\overline{P_i}| = g(k, n)$ and if it is greater than zero it exist a triangle partition.