Homework 2
class in "Algorithm Design", Fall 2016/17

Marco Favorito
Master of Science in Engineering in Computer Science
Department of Computer, Control, and Management Engineering
University of Rome "La Sapienza'
favorito.1609890@studenti.uniroma1.it

January 8, 2017

Contents

1 Exercise 1

Question 1.1

We will prove the statement by induction over *the number of different edge* between a generic pair of spanning trees, T_1 and T_2, showing that exists a path over the graph G' for such pairs, that is the definition of graph connectivity.
Note: We express this number as $|E(T_1) \setminus E(T_2)|$. It is evident that two spanning tree are *adjacent* iff $|E(T_1) \cap E(T_2)| = n-2$ where n is the number of nodes in G, i.e. T_1 and T_2 differ by only one edge (since each spanning tree contains exactly $n-1$ edges). An equivalent definition is that $|E(T_1) \setminus E(T_2)| = 1$ and, implicitly, $|E(T_2) \setminus E(T_1)| = 1$. Without loss of generality, we consider the former number.

Proof. **Base case:** $|E(T_1) \setminus E(T_2)| = 0$. It means that $T_1 = T_2$ and, trivially, exists a path from T_1 and T_2 in G'.

Inductive step: Assume that $\forall T_1, T_2 \in \mathcal{T}$,$|E(T_1) \setminus E(T_2)| \leq k$ the statement holds, i.e. exists a path in G', for a generic pair of spanning trees T_1 and T_2, from T_1 to T_2, where T_1 differs from T_2 by at most k edges. We will show that for $|E(T_1) \setminus E(T_2)| = k+1$, there exists a path from T_1 and T_2.

Consider such pair T_1, T_2 with $k+1$ different edges. There exists an edge $e_1 \in T_1$ such that $e_1 \notin T_2$. If we add e_1 to T_2 we obtain a fundamental cycle, let say C. Let define P as a path containing $C - e_1$. Since P is a path from two ends of e_1, there exists an edge $f \in P$ which has one end in each component of $T_1 - e_1$. Now $T_1' = T_1 - e + f$ is a tree, since, thanks to $f \in T_2$, both disconnected components of $T_1 - e$ have been connected. Moreover, T_1' is adjacent with T_1, since they differ by one edge.

Now we notice that $|E(T_1') \setminus E(T_2)| = (k+1) - 1 \leq k$ because T_1' no longer have e, which is an unshared edge, instead of f, which is a shared edge with T_2. So the number of different edge is decreased by one and, by induction, there exists a path from T_1' to T_2 in G'. It is immediate to notice that, since T_1 is adjacent with T_1', exists a path from T_1 to T_1', which implies that exists a path from T_1 to T_2.

We have shown that exists a path in G' for a generic pair of spanning trees. Therefore G' is connected. □

Question 1.2

Proof. Assume by contradiction that exist two distinct MST, T_a and T_b. Then there is an edge, let's say e_a, which is not in T_b. $T_b \cup \{e_a\}$ generates a fundamental cycle, C. Consider the most expensive edge in C, let's say e_b. For Lemma 1, and in particular for the *cycle property*, e_b must not belong to any minimum spanning tree. But only two cases can happens: either $e_b = e_a$, and then $e_b \in T_a$, or $e_b \neq e_a$, which means $e_b \in T_b$. Both cases contradicts Lemma 1 and so the statement about uniqueness must be true. □

Question 1.3

Proof. Consider a generic spanning tree T and the unique minimum spanning tree $\bar{T}$. Both of them are characterized by $n-1$ edges of E, the set of edges in G. Since $T \neq \bar{T}$, there exist at least one edge e such that $e \in T$ and $e \notin \bar{T}$.
Now, meditate on e. Why it should not be in $\bar{T}$? We know the answer thanks to Lemma 1: e belongs to a cycle C in G, and it is the most expensive edge in C (it exists and is unique by hypothesis on distinctness of edge costs). So, as we done in Question 1.1, consider $\bar{T} \cup \{e\}$ and the fundamental cycle C due to e in $\bar{T}$. Moreover, consider the path $P = C - e$ from u to v, both end of e. Removing e from T, i.e. $T' = T \setminus \{e\}$, generates T_1 and T_2, two components that are connected through $e \in T$. Since P is a path from u to v, there exists an edge $f \in \bar{T}$ that has one end in T_1 and the other in T_2. Therefore, $\hat{T} = T - e + f$ is a tree adjacent to T and $w(\hat{T}) < w(T)$, because e is the most expensive edge in C and so $w(e) > w(f)$.

By changing $e \notin \bar{T}$ with $f \in \bar{T}$ we found an adjacent tree with less total cost. This can be done until there are some edges in T that are not in the minimum spanning tree $\bar{T}$. Obviously, all trees T adjacent to the minimum spanning tree $\bar{T}$ have $w(T) > w(\bar{T})$. So, the statement does not hold for minimum spanning trees. This ends the proof. □

Question 1.4

The Algorithm 1 is the implementation of the local search required. It simply checks, given any spanning tree T of G, whether exists a pair of edge, one in T and the other not in T, such that the cost of T after the swap decreases; this operation is done until a better neighbour exists.

Assuming that all edges in G have distinct weights, the proposed algorithm always returns an optimal solution, i.e. a MST. Indeed, the algorithm always terminates because in each iteration the cost of our partial result strictly decreases (by

Algorithm 1 Find MST with local search

```
1: function MST-LocalSearch(G)
2:     Let T be any spanning tree of G
3:     new-cost ← w(T)
4:     repeat
5:         cost ← new-cost
6:         for each neighbour T' of T do
7:             if w(T') < w(T) then
8:                 T ← T'                          ▷ T' is a better neighbour
9:                 new-cost ← w(T')
10:            end if
11:        end for
12:    until new-cost = cost
13:    return T
14: end function
```

construction of the algorithm and by our assumption on edge weights), and its termination means that the current spanning tree T has no neighbours with less cost, implying that T is a minimum spanning tree (see Question 1.3).
About the running time: in the worst case, in each iteration the cost $w(T)$ improves by only 1, and eventually it reaches $w(MST)$. So, the number of iteration of the while loop is not greater than $C = \max_T w(T) - w(MST)$, which can be upper bounded with $C = w(G)$, the cost of the total graph. Checking for a better neighbour means looking for an improving-edge-swap; it means that we need to scan all edges $f \notin T$, insert it in T, look for the fundamental cyle and detect the most expensive edge $e \in T$, which will be the next edge to remove from T. If $e = f$, we need to look for another $f' \notin T'$.

- The cost of find an improving-edge-swap (e, f) is $\mathcal{O}(nm)$, since in the worst case we need to consider all edges $f \notin T$ which are in number $m-(n-1) = \mathcal{O}(m)$ and, for each of them, we need to find the fundamental cycle, which costs $\mathcal{O}((n-1)+n) = \mathcal{O}(n)$
- The cost of computing $w(T)$ and $w(T')$ given (e, f) is constant, since $w(T) = new - cost$ and $w(T') = w(T) - w(e) + w(f)$;

Then the total cost is $\mathcal{O}(C \cdot nm)$, which is not very efficient when C is too big (in fact, it is *exponential* in the size of the input C).
What happens if we eliminate the assumption that all edge costs are distinct? As done in [4] ch. 4.5, we can simply perturbe all edge cost by "different, extremely

small numbers", so that they all become distinct. The relative order between all edges is preserved, and the comparisons do not have ambiguity, since all weights are different. Therefore, also in this case the algorithm still works, and the running time is the same.

2 Exercise 2

The proposed TRUCK LOADING problem is actually the same of Load Balancing problem (see ch. 11.1 in [4]) aka Minimum Makespan Scheduling (see ch. 10 in [6]); changes the "dress", but the stuff is the same. My answers may be strongly inspired by the above mentioned references.

Question 2.1

An approximate polynomial time algorithm for this problem is the following [6]:

- Order the boxes arbitrarily;
- Allocate boxes to the trucks in this order, allocating the next box on the truck that has been assigned the least amount of weight so far.

It is easy to see that it gives a 2-approximation solution. First of all, consider following Lower Bound:

$$LB = max \left\{ \frac{1}{k} \sum_{i=1}^{n} w_i, max_i \; w_i \right\}$$

Which is simple to see: the optimal value (OPT), i.e. the minimum maximum for the allocation, is at least the highest weight among boxes or at least the average of the total weight among all trucks.
Now we will prove the following proposition:

Proposition 1. *Our greedy algorithm yields solutions with 2-approximation.*

Proof. Consider the truck T_i to whom is allocated the highest total weight W_i, which is our maximum weight, the quantity that we want to minimize. What we can say about it? That before the allocation of the last box, let's say b_j, the total weight assigned to T_i, $W_i' = W_i - w_j$, is the lowest among all other trucks. Therefore, we can say that:

$$W_i - w_j \leq \frac{1}{k} \sum_{i=1}^{n} w_i$$

Notice that the right member is one of our lower bound, so:

$$W_i - w_j \leq \text{OPT}$$

Now, just add both member the other lower bound, i.e.:

$$w_j \leq \text{OPT}$$

then:

$$(W_i - w_j) + w_j = (\text{OPT}) + \text{OPT}$$

which gives:

$$W_i \leq 2 \times \text{OPT}$$

Which is what we are looking for. □

A tight example that shows this result is the following: consider an istance of the problem with k^2 boxes with unit weight plus one box with weight k, where k is also the number of trucks. If we pick firstly lightest boxes and secondly the weighty one, the allocation returned by the algorithm has cost $2k$, because it distributes light boxes uniformly among the trucks and allocate the last box with weight k to one of them, let's say T_i; as result, as just said, we have that the maximum weight, that is on T_i, is $2k$, instead of the optimum $k + 1$ (allocate the weighty box on one truck and distribute the k unit-weighted boxes destined to T_i among all trucks). So, the approximation ratio is:

$$\frac{\text{SUBOPT}}{\text{OPT}} = \frac{2k}{k+1} < 2$$

Question 2.2

The answer to this question is inspired to cap. 10.2 of [6], where is considered the *bin packing problem* in relation to *minimum makespan problem*, which is equivalent to our Truck loading problem. In the following I prove that each instance of BIN PACKING can be transformed into an instance of the TRUCK LOADING. Doing so, since BIN PACKING is NP-hard, also TRUCK LOADING will be NP-hard.

For the sake of completeness, before proceeding in the proof, I state the BIN PACKING problem:
Given m items of size $a_1, \ldots, a_k$ minimize the number n of bins needed to pack the items, given that each bin has size B.

Proof. We proceed as follows: denote the weight of the m boxes with weights $a_1, \ldots, a_k$ by I and let allocate(I, k) be the maximum weight $\hat{W}$ using at most k. Then, the minimum number of bins required is:

$$\min\{k : \text{allocate}(I, k) \leq B\}$$

In a nutshell, the core of the reduction is the analogy between the number of bins and the number of trucks, between the items to be packed and the boxes to be allocated, and between the capacity of bins and the maximum weight allowed for

the allocation in the TRUCK LOADING problem. Since we want to minimize the number of bins, we need to fix the maximum weight allowed for all the trucks and find the minimum number of trucks such that the just mentioned constraint is satified.
Since we can easily find a 2 approximation for BIN PACKING, just perform a binary search among LB and 2LB yields a solution for the BIN PACKING problem using TRUCK LOADING problem as a "black box". Then BIN PACKING $\leq_P$ TRUCK LOADING and, since the former is NP-hard, also the latter is NP-hard. □

3 Exercise 3

Introduction

The following arguments are inspired by [2], since the POINT PARTITIONING problem is actually the same of *max-diameter clustering*.
In this section I will prove the suggested fact about representatives $\mu_1, \mu_2, \ldots, \mu_k$, i.e.:

$$\forall j : \max_{x,y \in C_j} d(x, y) \leq 2 \max_{z \in C_j} d(z, \mu_j)$$

Proof. Very simple: let x, y belonging to some C_j be two points such that the first member of the disequality is maximized. Then:

$$d(x, y) \leq d(x, \mu_j) + d(y, \mu_j)$$

By triangular inequality, and:

$$d(x, \mu_j) + d(y, \mu_j) \leq \max_{z \in C_j} d(z, \mu_j) + \max_{z \in C_j} d(z, \mu_j) = 2 \max_{z \in C_j} d(z, \mu_j)$$

trivially: any distance between any point $x \in C_j$ and another point μ_j is lesser or equal than the maximum of that distance. □

3.1 Proposed algorithm

Farthest-first heuristic will be the right heuristic in order to choice the k centroids of our points in order to reach a factor-2 approximation.
Indeed, consider the list of partition $\mathbf{C}$ and the relative representatives $\mu_1, \mu_2, \ldots, \mu_k$ returned by the algorithm. Consider $r = \max_{z \in C_j} d(z, \mu_j)$ for some j s.t. $1 \leq j \leq k$, and be $\bar{z}$ the point that maximizes this value. Due to our greedy heuristic, all other representatives have distance from μ_j at least r, the distance between $\bar{z}$ and μ_j. Moreover, notice the following obvious fact:

$$\text{OPT} \geq r$$

Which means that the maximum diameter can be no lower than the maximum distance between any point and its representative in the partition which they belong to.[1]

[1] To see this intuitively, consider an example with n points and $k = n - 1$ partitions. There must exist two points that belong to the same partition (by the pigeonhole principle). In this tight case, the max diameter is equal to the max distance to the representative, because there are no other choices. To see why it is an inequality, consider a case in which there are not only two points mbut more than two, let's say three, x, x', μ and suppose that the max diameter is between x and x' and that the representative is μ. For the triangular inequality, the inequality is trivially verified.

But for what we have proved in the previous section, the maximum diameter cannot be greater than the maximum representative distance times 2, i.e. OPT $\leq 2r$. The algorithm that I show is entirely taken from [2], which has already solved this problem: Essentially this algorithm yields $\mathbf{B} = (B_1, \ldots, B_k)$ which is exactly $\mathbf{C}$

```
Algorithm APPROX
  let {v_1, ..., v_n} be the set of elements to be clustered;
  let head_1 represent v_1; let B_1 ← {v_1, ..., v_n};
  for l ← 1 to k − 1 do
    h ← max{W(head_j, v_i) | v_i ∈ B_j and 1 ≤ j ≤ l};
    let v_i be one of the nodes whose distance to the head of the cluster it belongs
      to is h;
    move v_i to B_{l+1};
    let head_{l+1} represent v_i;
    for each v_t ∈ (B_1 ∪ ··· ∪ B_l) do
      let j be such that v_t ∈ B_j;
      if W(v_t, head_j) ≥ W(v_t, v_i) then move v_t from B_j to B_l;
      endif
    endfor
  endfor
  return (B_1, ..., B_k);
end of algorithm;
```

Figure 1: Farthest-first traversal algorithm [2] for POINT PARTITIONING

above defined.
In words: pick any point v_1 from X and set the first partition as X. Select the maximum distant point from any "head" (i.e. representative" already set, and make it the new head of a new partition. Update all other partitions such that points that are nearer to the new head instead of the current head has to be moved in the new partition.

Running time

The running time of this algorithm is $\mathcal{O}(nk)$, which is polynomial, since $k < n$. The proof is straightforward: the algorithm has two nested loops, the outer loop makes k iterations (one for each partition, detecting its representative), the inner one n, because for all point the algorithm has to update the partitions they belong to.

4 Exercise 4

This problem is fully reducible to WEIGHTED SET COVER. We will prove the following:

1. *There exist a solution for* WAREHOUSE OPENING *iff there exists a solution for* WEIGHTED SET COVER.

Proof. Sufficiency: compute, for every site s_i, the subset $\widehat{S}_i \subseteq U$ defined as follows:

$$\widehat{S}_i = \{c_j : s_i \in S_j\} \tag{1}$$

Now we have $\widehat{\mathbf{S}}$, which is the set of $\widehat{S}_i \in \widehat{\mathbf{S}}$, the subset of cities that s_i can serve. If the instance of the problem can be solved, we have that $\widehat{S_1} \cup \widehat{S_2} \cup \cdots \cup \widehat{S_k} = U$, otherwise there exists a city that it cannot be served by any sites, by our definition of $\widehat{\mathbf{S}}$.
Now we have an instance for the WEIGHTED SET COVER:

- a set of elements $U = c_1, \ldots, c_n$;
- a set of subsets of U, $\widehat{\mathbf{S}}$ defined above;
- a cost function $w : \widehat{\mathbf{S}} \to \mathbb{N}^2$; (simply associate the cost of s_i to $\widehat{S}$)

Now just find the subcollection $\widehat{\mathcal{S}} \subseteq \widehat{\mathbf{S}}$ such that $\sum_{\widehat{S} \in \widehat{\mathcal{S}}} w(\widehat{S})$ is minimized.

Necessity: just perform the inverse operations, i.e. define, for every element $c_i \in U$, a set S of $S_i \subseteq U$ defined as follows:

$$S_i = \{s_j : c_i \in \widehat{S}_j\}$$

Where $\widehat{S}_j \in \widehat{\mathbf{S}}$ is the subset of all cities that the j_{th} site can serve. The cost function is converted as in the previous case. Notice that, if the instance of WEIGHTED SET COVER can be solved, i.e. all elements $\in U$ can be covered by at least one subset $\widehat{S} \in \widehat{\mathbf{S}}$, by our definition of S we have that for each city there must exists a site such that it can serve some cities. It is easy to see that we have an instance of WAREHOUSE OPENING problem:

- INPUT: set U of elements (cities), set S of sites and relative cost function $w : S \to \mathbb{N}$, for each element $\in U$ we have $\mathbf{S}$ as the set of subsets of S, let's say S_i, that contains all the sites that can serve the city c_i.

[2] We consider the integer version, since it is not specified what is the domain of w.

- OUTPUT: $W \subseteq S$, which is the set of sites to be selected in order to cover all cities.
- GOAL: minimize the total cost.

In a nutshell: the core of the analogy is on the collection of subsets of the universe U and the set of sites S, that after some transformation it is the collection of subset of al the cities that have to be covered. □

Since WEIGHTED SET COVER $\leq_P$ WAREHOUSE OPENING, and since WEIGHTED SET COVER is NP-hard, then also WAREHOUSE OPENING is NP-hard.

Proposed algorithm

For the solution of the problem (see Algorithm 2) we can simply apply the same approach used in ch. 2.1 of [6], i.e. a greedy heuristic for solve the SET COVER problem. The only needed thing is to transform the instance of the WAREHOUSE OPENING into one of SET COVER, as it has been shown before.

Correctness

As explain at the beginning of this exercise, we have just reduced WAREHOUSE OPENING to SET COVER; since we used the same algorithm to approximate SET COVER[3], we already know that it is correct.

Complexity

The reduction from WAREHOUSE OPENING, since we need to computing $\widehat{\mathbf{S}}$, costs $\mathcal{O}(nk)$, where n is the number of cites and k is the number of sites (for a generic city c_i we look for all s_i in S_i; in the worst case $|S_i| = k$).
In GREEDY-SET-COVER the number of iteration of the while-loop is $\mathcal{O}(n)$ (in the worst case we add only one element to C in each iteration), times the time needed to compute the set that reach the minimum cost-effectiveness, which is $\mathcal{O}(k)$. So, we have again $\mathcal{O}(nk)$. The transformation of the output of SET COVER to an output of WAREHOUSE OPENING is negligible[4]. So, the overall cost is $\mathcal{O}(nk)$.

[3] GREEDY-SET-COVER is the same function in [6], ch. 2.1

[4] The exact time estimation depends on implementation, but is negligible. With the "worst" implementation, it runs in linear time, $\mathcal{O}(k)$, but associating each s_i to each $\hat{S}_i$ in some way could give $\mathcal{O}(1)$.

Algorithm 2 Greedy-Warehouse Opening

function GREEDY-WAREHOUSE-OPENING($U, S, \mathbf{S}, \mathbf{w}_s$)
    INPUT:
        U, set of cities;
        S, set of sites;
        $\mathbf{S}$, subsets S_i of sites from which c_i can be served;
        $\mathbf{w}_s$, costs for opening a warehouse at site s.
    OUTPUT:
        $W' \subseteq S$ such that each city can be served by at least one warehouse.

    Compute $\widehat{\mathbf{S}}$, defined above (1)
    $W \leftarrow$ GREEDY-SET-COVER($U, \widehat{\mathbf{S}}$)
    $W' \leftarrow$ The set of sites relative to W
    return W'
end function

function GREEDY-SET-COVER($U, \mathbf{S}, \mathbf{w}_s$)
    INPUT:
        U, set of elements;
        $\mathbf{S}$, collection of subset $S_i \subseteq U$;
        $\mathbf{w}_s$, costs for opening a warehouse at site s.
    OUTPUT:
        $W \subseteq \mathbf{S}$, the selected sets
    Start with $C = \emptyset$ and no sets selected
    while $C \neq U$ **do**
        Select set S_i that minimizes $\frac{w_i}{|S_i \setminus C|}$
        $C \leftarrow C \cup \{S_i\}$
    end while
    return the selected sets, W
end function

Approximation ratio

The approximation ratio of GREEDY-SET-COVER, as proven in [6] (ch. 2.1), is H_n, and so also for GREEDY-WAREHOUSE-OPENING.

5 Exercise 5

5.1 GBAS

The problem is to estimate the probability of ON(p) to be true, i.e. $\rho = \frac{A}{N}$, where N is the number of pixels. ON then can be modeled as $Y \sim \text{Ber}(\rho)$. Once estimated this probability, let's say $\hat{\rho}$, X is simply $N \cdot \hat{\rho}$. The requirement is that the estimation has to be quite accurate in terms of *multiplicative error* ϵ, with probability of failure δ ad with the minimum number of samples n needed to accurate the estimation.
This kind of problem is known in the literature as an $(\epsilon, \delta)-$*approximation* of a mean μ of some random distribution, which means that:

$$\mathbf{Pr}(|\hat{\mu} - \mu| \leq \epsilon\mu) \geq 1 - \delta \tag{2}$$

Where $\hat{\mu}$ is such estimation. In our case:

$$\mathbf{Pr}(|X - A| > \epsilon A) \leq \delta \tag{3}$$

A naive approach suggests the following arguments: let:

$$Z = \sum_{k=1}^{n} Y_i \sim \mathbf{Bin}(n, \rho)$$

where $Y_1, \ldots, Y_n$ are i.i.d. random variables. From 3:

$$\mathbf{Pr}(|X\frac{n}{N} - A\frac{n}{N}| > \epsilon A\frac{n}{N}) \leq \delta$$

$$\iff$$

$$\mathbf{Pr}(|Z - n\rho| > \epsilon n\rho) \leq \delta$$

The last formula, since $\mathbf{E}[Z] = n\rho$, is in the form of Chebyshev Inequality. Indeed, given that $Var[Z] = n\rho(1-\rho)$:

$$\mathbf{Pr}(|Z - n\rho| > \epsilon n\rho) \leq \frac{Var[Z]}{(\epsilon n\rho)^2} = \frac{n\rho(1-\rho)}{\epsilon^2} n^2\rho^2 = \frac{1-\rho}{\epsilon^2 n\rho} \leq \frac{1}{\epsilon^2 n\rho}$$

Since we want to limit the probability with δ, we need that:

$$\delta \geq \frac{1}{\epsilon^2 n\rho}$$

And solving for n:

$$n \geq \frac{1}{\epsilon^2 \delta \rho} \tag{4}$$

The problem is that we do not know ρ before running the algorithm: it is the parameter that we want to estimate. We can consider a higher lower bound for n, remembering the value of ρ:

$$n \geq \frac{N}{\epsilon^2 \delta}$$

But it can be a very high if ϵ and δ are small, make our considerations useless, since if $\epsilon^2\delta > 1$, it means that $n > N$, and then just pick all the pixels gives the optimal solution, that is what we are trying to avoid.
A good algorithm (not the best one) for this kind of problem is provided by the *Stopping Rule Algorithm* (SRA) [1], assuming $\mathbf{E}[Y] = \rho > 0$ (i.e. $A \neq 0$).

SRA algorithm :
input: (ϵ, δ) with $0 < \epsilon < 1$ and $\delta > 0$, random vars Y_i i.i.d.
output: $\hat{\rho}$ approximation of ρ

(1) $\Gamma = 4(\mathrm{e}-2)\ln(2/\delta)/\epsilon^2; \Gamma_1 = 1 + (1+\epsilon)\Gamma$;
(2) $\mathbf{for}(N = 0, S = 0; S \leq \Gamma_1; N\mathbf{++})S = S + Y_N$;
(3) $\hat{\rho} = S/N$; $\mathbf{return}\ \hat{\rho}$;

The correctness of the algorithm is proved in [1]. Moreover, it states that the expected value of trials N_Y is bounded, i.e.:

$$\mathbf{E}[N_Y] \leq [1 + (1+\epsilon)4(e-2)\ln(2/\delta)\epsilon^{-2}]/\rho$$

Of course, if $\Gamma_1 \geq N$, it is definitely more convenient to sample all N pixels and return the exact estimation.
Actually, a more stronger result has been achieved in [3] with the GBAS algorithm. It yields an estimate which requires a number of flips very close to the optimal.

I elaborated another approach to the problem that I will show in the following sections.

5.2 FPRAS

This kind of problems is known in the literature as *counting problems* ([6], ch. 28). For a lot of problems, the *exact counting version* (e.g. count the number of solution for a certain problem) requires exponential time; often, what we can do efficiently is only an approximate counting of this solutions through some randomized procedure. Exists a definition that captures such class of algorithm ([6]):
The algorithm $\mathcal{A}$ is a *fully polynomial randomized approximation scheme* (FPRAS) for a counting problem f if, for each possible instance x of the problem and error parameter $\epsilon > 0$,

$$\mathbf{Pr}(|\mathcal{A} - f(x)| \leq \epsilon f(x)) \geq \frac{3}{4} \tag{5}$$

or, equivalently:

$$\mathbf{Pr}(|\mathcal{A} - f(x)| > \epsilon f(x)) \leq \frac{1}{4} \tag{6}$$

and the running time of $\mathcal{A}$ is polynomial in $|x|$ and $1/\epsilon$. $\mathcal{A}(x)$ is called also a $(\epsilon, 1/4)$-approximation.
Notice that the constant in the right side of the inequality can be improved to $1 - \delta$ (for 5, and δ for 6) for any $0 < \delta < 1$ in the following way ([6] Exercise 28.1): run the algorithm $k = 12\log(1/\delta)$ times with error probability $1/4$, obtaining outputs $y_1, \ldots, y_k$, and consider m, the median of these values; m is the value that achieves the desired error probability.

Explanation In the following, I explain briefly why it is true. Let

$$X_i = \begin{cases} 1 & \text{if } y_i \notin (1 \pm \epsilon) f(x) \\ 0 & \text{otherwise} \end{cases}$$

Notice that $P(X_i = 1) \leq 1/4$ i.e. the probability that the i_{th} result returned by the algorithm is outside the range. Now let's define $Y = \sum_i^k X_i$. $E[Y] \leq k/4$ is an upper bound for the number of failures. Moreover, observe that if the median result, $m = y_{k/2}$ is outside the range (i.e. $X_{(k/2)} = 1$), then at least half of the k result must be outside the range. Then, using Chernoff Bound ([5], Theorem 4.4):

$$\mathbf{Pr}(X_{(k/2)} \notin (1 \pm \epsilon) f(x)) \leq \mathbf{Pr}(Y \geq k/2) = \mathbf{Pr}(Y \geq (1+1)k/4) \leq e^{-k/12}$$

Therefore, if we choose $k > 12\ln(1/\delta)$, the probability to fail is less than δ and the time complexity is $poly(1/\epsilon, \ln(\delta^{-1}, |x|)$. So, the definition given in 5 is equivalent to this one:
The algorithm $\mathcal{A}$ is a *fully polynomial randomized approximation scheme* (FPRAS) for a counting problem f if, for each possible instance x of the problem and parameter $\epsilon > 0, 0 < \delta < 1$,

$$\mathbf{Pr}(|\mathcal{A} - f(x)| \leq \epsilon f(x)) \geq 1 - \delta \tag{7}$$

and the the time complexity is $poly(1/\epsilon, \ln(\delta^{-1}, |x|)$. $\mathcal{A}(x)$ is called also a (ϵ, δ)-approximation.
In the following section we will analyze the problem and we will exploit the above results.

5.2.1 Analysis of the problem

What we need for estimate efficiently A is to upper bound the probability that, sampling n times a pixel, we fail the estimation outside the error range, and at the

same time we need to limit it as lower as possible (surely les than N, the number of pixels).
Our requirement is that:

$$\mathbf{Pr}(|X - A| > \epsilon A) \leq \delta \tag{8}$$

Where X represents our estimation. What we will do now is to upper bound the error probability to $1/4$ designing a procedure that samples a certain number of pixels $\hat{n}$ and makes an estimation; then run that procedure k times in order to achieve the error probability less than δ, in the same fashion of the previous section.

Upper bound for the error probability. Let $\rho = \frac{A}{N}$, and let

$$Y = \sum_{k=1}^{n} Y_i \sim \mathbf{Bin}(n, \rho)$$

Where $Y_i \sim \text{Ber}(\rho)$ is the random variable that models ON(p_i): its value is 1 with probability ρ if the sampled pixel p_i is ON. Obviously, let $\mu = E[Y_i] = \rho$ be the mean, and $\sigma = \rho(1 - \rho)$ be the standard deviation. Since we consider $Y_1, \ldots, Y_n$ independent Bernoulli trials, we have that $\sigma_Y = \sigma\sqrt{n}$.
By Chebyshev inequality:

$$\mathbf{Pr}(|Y - n\mu| > t\sigma\sqrt{n}) < \frac{1}{t^2} \tag{9}$$

Choosing $t = 2$ and observing that $\mu > \sigma$, from (9) follows that:

$$\mathbf{Pr}(|Y - n\mu| > 2\mu\sqrt{n}) < \frac{1}{2^2} = \frac{1}{4}$$

Then,

$$\mathbf{Pr}(|\frac{N}{n} \cdot Y - \frac{N}{n} \cdot n\mu| > \frac{N}{n} \cdot 2\mu\sqrt{n}) < \frac{1}{4}$$
$$\iff$$
$$\mathbf{Pr}(|X - A| > \frac{2A}{\sqrt{n}}) < \frac{1}{4}$$

In order to obtain an $(\epsilon, 1/4)$-approximation we need $\epsilon > \frac{2}{\sqrt{n}}$ which implies that we need $n > \frac{4}{\epsilon^2}$, samples, and then $n = poly(\frac{1}{\epsilon})$.
In order to achieve an (ϵ, δ)-approximation, it is enough to perform $k = 12\ln(1/\delta)$ times the sampling for the $(\epsilon, 1/4)$-approximation. The total cost is then $\mathcal{O}(poly(\frac{1}{\epsilon}) \cdot \log\frac{1}{\delta})$

5.2.2 Designing the algorithm

So, what we need is a procedure that samples $n = \lceil \frac{4}{\epsilon^2} \rceil$ pixels randomly, counts the number of ON pixels, and estimate X. Then, the algorithm should run that procedure $\lceil k = 12 \ln(1/\delta) \rceil$ times, pick the median value $m = X_{k/2}$ and return m.
The total number of samples is $\hat{n} = \frac{4*k}{\epsilon^2} = \frac{48}{\epsilon^2} \cdot \ln(1/\delta)$. If this number if greater than the total number of pixel N, then it is more smart to samples all the N pixels and return the exact number of ON pixels, A. Otherwise, running the just described algorithm should reduce the number of pixels sampled.
Algorithm 3 should implement the just described procedure.

5.3 No-Repetition

5.3.1 Analysis of the problem

What we need is an upper bound, in function of the number of samples n, of the probability to fail the estimation of A, the number of pixel that are ON, more than an error ϵ, such that the probability is not greater than δ. in symbols:

$$\mathbf{Pr}(|X - A| > \epsilon A) \leq \delta \tag{10}$$

Where X represents such estimation.
Let's define ρ as the probability to select a pixel p which is ON, i.e.: $\rho = \frac{A}{N}$. Moreover, let define N as the number of total pixel on the screen.

Notice that `ON(`p_i`)` can be represented as the random variable Y_i distributed as a Bernoulli with probability to success (i.e. `ON(`p_i`)` is true) equal to ρ.

Now, if we assume that $Y_1, \ldots, Y_n$ are i.i.d., we have that:

$$Y = \sum_{k=1}^{n} Y_i \sim \mathbf{Bin}(n, \rho)$$

Which represents the number of `ON(p)` on n samples. Now, consider the following equivalences:

$$\hat{\rho} = \frac{Y}{n}$$

Which is the estimate probability of `ON(p)` after n samples, and

$$X = N\hat{\rho} = N \cdot \frac{Y}{n}$$

Algorithm 3 The algorithm COUNT ON PIXEL

function ESTIMATE-ON-PIXEL(S, N, ϵ, δ)
    INPUT:
        $S = \{p_1, p_2, \ldots, p_N\}$, a collection of pixels;
        N, the number of pixels;
        ϵ, the allowed error on estimating A;
        δ, max probability to fail the estimation with error greater than ϵ.
    OUTPUT:
        X, the estimated A.

    $n \leftarrow \lceil \frac{4}{\epsilon^2} \rceil$
    $k \leftarrow \lceil 12 \ln 1/\delta \rceil$
    if $n \cdot k \geq N$ **then**
        Do exact counting...
        return A
    else
        result-list $\leftarrow \{\}$
        for i from 1 to k **do**
            count $\leftarrow 0$
            for j from 1 to n **do**
                $p_j \leftarrow$ a pixel sampled randomly from S
                if ON(p_j) is true **then**
                    count $\leftarrow$ count+1
                end if
            end for
            result-list $\leftarrow$ result-list $\cup \{\lfloor \text{count}/n \rfloor\}$
        end for
        $m \leftarrow$ the median of result-list
        return m
    end if
end function

From which follows that:

$$Y = nX \cdot \frac{1}{N}$$

Then, from 10, multiplying both member of inequality inside the $\mathbf{Pr}$ function:

$$\mathbf{Pr}(|X\frac{n}{N} - A\frac{n}{N}| > \epsilon A\frac{n}{N}) \leq \delta \iff \mathbf{Pr}(|Y - n\rho| > \epsilon n\rho) \leq \delta$$

The last formula, since $\mathbf{E}[Y] = n\rho$, is in the form of Chebyshev Inequality. Indeed, given that $Var[Y] = n\rho(1-\rho)$:

$$\mathbf{Pr}(|Y - n\rho| > \epsilon n\rho) \leq \frac{Var[Y]}{(\epsilon n\rho)^2} = \frac{n\rho(1-\rho)}{\epsilon^2} n^2\rho^2 = \frac{1-\rho}{\epsilon^2 n\rho} \leq \frac{1}{\epsilon^2 n\rho}$$

Remembering the expression of ρ:

$$\frac{1}{\epsilon^2 n\rho} = \frac{N}{\epsilon^2 nA} \leq \frac{N}{\epsilon^2 n}$$

Since we want to limit the probability with δ, we need that:

$$\delta \geq \frac{N}{\epsilon^2 n}$$

And solving for n:

$$n \geq \frac{N}{\epsilon^2 \delta} \tag{11}$$

So we have a lower bound for the number of samples n that we need in order to estimate A with an error not greater than ϵ with probability $1 - \delta$.
Notice that, if $\epsilon^2\delta \leq 1$, it is not convenient to use this method, because in that case $n > N$, and then just pick all the pixels gives the optimal solution. So, our analysis suggests a procedure that compute the lower bound for n and, after n calls to `ON(p)` picking `p` randomly from the screen[5], we count the number that `ON(p)` is true and we estimate A by simply divide that number to the number of samples n (i.e. X).

5.3.2 Proposed algorithm

The algorithm 4 simply determines, taking into account 11 and the above considerations, how much samples it has to done, n. Then, pick randomly n times a pixel and call the subroutine `ON(p)`: if it returns true, then means that p is ON, and so increment the counter $count$. At the end, compute $\hat{\rho} = \frac{count}{n}$, the estimated probability of `ON`, and multiply it by N, which returns X, the required quantity.

[5]We omit the fact that we need two coordinates, since it is a detail that it is not crucial for the analysis

Algorithm 4 The algorithm COUNT ON PIXEL

function COUNT-ON-PIXEL(S, N, ϵ, δ)
    INPUT:
        S, a collection of pixels;
        N, the number of pixels;
        ϵ, the allowed error on estimating A;
        δ, max probability to fail the estimation with error greater than ϵ.
    OUTPUT:
        X, the estimated A.

    if $\epsilon^2 \cdot \delta \geq 1$ **then**
        $n \leftarrow N$
    else
        $n \leftarrow \lceil \frac{N}{\epsilon^2 \delta} \rceil$
    end if
    Initialize $count$ to 0
    for i from 1 to n **do**
        Pick a random pixel p from S
        if ON(p) **then**
            $count \leftarrow count$+1
        end if
    end for
    $X \leftarrow \frac{count}{n} \cdot N$
return X
end function

5.3.3 Ranges of ϵ, δ, A for $\mathcal{O}(n)$ running time

As said before, when $\epsilon^2\delta \leq 1$, it is more convenient to pick all N pixels. So, when $\epsilon^2\delta > 1$, the algorithm runs in $\mathcal{O}(N)$ times, since $n < N$. Moreover, $\delta \in (0, 1)$, since it is a probability, and $\epsilon > 1$, since δ cannot be greater than 1.

References

[1] Paul Dagum et al. “An Optimal Algorithm for Monte Carlo Estimation”. In: *SIAM J. Comput.* 29.5 (Mar. 2000), pp. 1484–1496. ISSN: 0097-5397. DOI: 10.1137/S0097539797315306. URL: http://dx.doi.org/10.1137/S0097539797315306.

[2] Teofilo F. Gonzalez. “Clustering to Minimize the Maximum Intercluster Distance”. In: *Theor. Comput. Sci.* 38 (1985), pp. 293–306. DOI: 10.1016/0304-3975(85)90224-5. URL: http://dx.doi.org/10.1016/0304-3975(85)90224-5.

[3] Mark Huber. *An unbiased estimate for the mean of a {0,1} random variable with relative error distribution independent of the mean*. arXiv: 1309.5413v2. URL: https://arxiv.org/abs/1309.5413v2.

[4] Jon Kleinberg and Eva Tardos. *Algorithm Design*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2005. ISBN: 0321295358.

[5] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. New York, NY, USA: Cambridge University Press, 2005. ISBN: 0521835402.

[6] Vijay V. Vazirani. *Approximation Algorithms*. New York, NY, USA: Springer-Verlag New York, Inc., 2001. ISBN: 3-540-65367-8.