Algorithm Design Homework 02

Reference Book: Algorithm Design - Jon Kleinberg Eva Tardos

Eleonora Beatrice Rossi 1749038 - Andrea Dito 1844760

10/01/2023

1 Exercise 1

A vertex cover of a graph is a subset of its vertices such that every edge in the graph is incident to at least one vertex in the subset. The weighted vertex cover is a set of vertices such that the sum of the weights of the vertices in this set is minimized. In this exercise it's proposed a particular weighted vertex cover where the weight of each vertex corresponds to its degree: $w_v = d(v)$.
To prove that any greedy algorithm is a 2-approximation for the vertex cover problem, we need to show that the weight of the vertex cover found by any greedy algorithm is at most twice the weight of the optimal vertex cover OPT.

$$AnyGreedy \leq 2 \cdot OPT.$$

We can start by defining C^*, which is the set of vertices composing the optimal vertex cover in G=(V,E):

$$OPT = w_{C^*} = \sum_{v \in C^*} w_v = \sum_{v \in C^*} d(v) = \sum_{(u,v) \in E} f(x)$$

where *f(x)* is a function defined as follows:

$$f(x) = \begin{cases} 1 & \text{if } v \in C^* \text{ and } u \notin C^* \textit{ or vice versa} \\ 2 & \text{if both } v \in C^* \text{ and } u \in C^* \end{cases}$$

The quantity described above must be greater or equal to the number of edges $|E|$ in the graph, because each edge must be covered by at least one of its endpoints; in fact the function *f(x)* will sum a value equal to 1 or 2 for every single edge. More specifically in order to cover each edge (u,v), either vertex u or v must be included in the vertex cover.

$$OPT = w_{C^*} = \sum_{v \in C^*} w_v = \sum_{v \in C^*} d(v) = \sum_{(u,v) \in E} f(x) \geq |E|$$

Now we can define instead the structure of *AnyGreedy*. Since we need to prove that any greedy algorithm gives a 2-approximation to the OPT solution, we can take in consideration the worst possible greedy algorithm *WorstGreedy*, which is the one that, in order to obtain a weighted vertex cover, selects all vertices. Once that we have proved that even *WorstGreedy* gives a 2-approximation, then *AnyGreedy* would guarantee a 2-approximation as well.

$$WorstGreedy = w_V = \sum_{v \in V} w_v = \sum_{v \in V} d_v = \sum_{(u,v) \in E} f(x)$$

In the case described above, the one in which the algorithm *WorstGreedy* takes all vertices, the function *f(x)* assumes value 2 for each edge considered. Therefore we have:

$$WorstGreedy = w_V = \sum_{v \in V} w_v = \sum_{v \in V} d_v = \sum_{(u,v) \in E} f(x) = 2 \cdot |E|$$

This is confirmed by a well-known result in graph theory (p. 89 of the book) that states:
Let G = (V, E) be a graph with $|V|$ vertices and $|E|$ edges; then, the sum of the degrees of the vertices in G is equal to twice the number of edges in G.

$$\sum_{v \in V} d(v) = 2 \cdot |E|$$

It implies that:

$$\begin{cases} 1. & OPT \geq |E| \\ 2. & AnyGreedy \leq WorstGreedy = 2|E| \end{cases}$$

We can conclude that the size of the weighted vertex cover generated by any greedy algorithm is at most twice the size of an optimal vertex cover. In other words:

$$AnyGreedy \leq 2 \cdot |E| \leq 2 \cdot OPT$$

Therefore in a weighted vertex cover where $w_v = d(v)$, any possible greedy algorithm guarantees a 2-Approximation to the best vertex cover.

2 Exercise 2

2.1 The Strategy

The strategy that we should implement in order to select the card with the maximal value at least $\frac{1}{4}$ of the times is the following:

1. Initialize a local variable *max = 0*
2. We flip the first half of the cards and reject them all. For each one of the flipped cards we update the local variable *max* if the value of the flipped card is greater or equal than *max*.
3. After the first $\frac{n}{2}$ cards we have stored in *max* the highest value seen so far and we stop to update it.
4. From now on for every flipped card having value v we verify if $v \geq max$. If that check is passed we select it, otherwise we continue to flip the cards one by one.
5. If in the second half of the cards we didn't find a value $v \geq max$ then we pick the last one.

```
def select_max_card(cards):
  max = 0
  for i in range(0, len(cards) // 2):
    if cards[i] >= max:
      max = cards[i]
  for i in range(len(cards) // 2, len(cards)):
    if cards[i] >= max:
      return cards[i]
    if i == len(cards)
      return cards[i]
```

2.2 Proof of Correctness

The proposed strategy grants us to obtain the card with the maximal value at least $\frac{1}{4}$ of the times. We can give a proof of the correctness of our algorithm by following this approach:

Let's define max_1 the card with the highest value between the entire set of n cards, while max_2 is the card with the second highest value. The adopted strategy is a winning one if max_2 appears in the first half of the cards and max_1 belongs to the second one instead. We can analyze all those permutations having the defined structure:

- The card with value max_2 can be found in $\frac{n}{2}$ positions (the first half of the cards).
- The card with value max_1 can be found in $\frac{n}{2}$ positions as well (the second half of the cards).
- The remaining $(n-2)$ cards can be organized in any order, then we have $(n-2)!$ possible permutations.

$$\textit{Winning Strategy} = \frac{n}{2} \cdot \frac{n}{2} \cdot (n-2)! = \frac{n^2 \cdot (n-2)!}{4}$$

To compute the probability of *Winning Strategy* we can simply calculate it as the number of favorable outcomes over all the possible outcomes, where all the possible outcomes are all the permutations of n cards: *n!*

$$\begin{aligned} P(\textit{ Winning Strategy }) &= \frac{\text{Favorable Outcomes}}{\text{Total Outcomes}} \\ &= \frac{\frac{n^2 \cdot (n-2)!}{4}}{n!} = \frac{n^2 \cdot (n-2)!}{4 \cdot n!} \\ &= \frac{n^2 \cdot (n-2)!}{4 \cdot n(n-1)(n-2)!} \\ &= \frac{n}{4(n-1)} = \frac{1}{4} \cdot \frac{n}{(n-1)} \geq \frac{1}{4} \end{aligned}$$

The probability of our *Winning Strategy* results $\geq \frac{1}{4}$ since it corresponds to $\frac{1}{4} \cdot \frac{n}{(n-1)}$, which is basically $\frac{1}{4}$ multiplied by a value always greater or equal than 1. Therefore we have proved that our strategy selects the card with the maximal value with a probability of at least $\frac{1}{4}$.

3 Exercise 3

3.1 IP Formulation and LP Relaxation

Taken the graph G = (V, E), the problem proposed asks us to find a multi-set S that d-covers all the edges by minimizing the weight. We can formulate it with the following IP instance:

$$\begin{aligned} \text{Minimize} \quad & \sum_{v \in V} w_v \cdot x_v \\ \text{Subject to} \quad & 2x_u + x_v \geq 2 \qquad \forall (u,v) \in E \\ & x_v \geq 0 \qquad\qquad \forall v \in V \\ & x_v \in \mathbb{Z} \qquad\qquad \forall v \in V \end{aligned}$$

This formulation aims to minimize the weight of the multi-set S, obtained through the sum of all vertices belonging to it. The first constraint ensures that either $x_u \geq 1$ or $x_v \geq 2$, allowing the creation of a multi-set that d-covers the graph G, while the second and the third impose that each vertex can be added to the multi-set an integer number of times.

The corresponding LP relaxed version is:

$$\begin{aligned} \text{Minimize} \quad & \sum_{v \in V} w_v \cdot \widehat{x}_v \\ \text{Subject to} \quad & 2\widehat{x}_u + \widehat{x}_v \geq 2 \qquad \forall (u,v) \in E \\ & \widehat{x}_v \geq 0 \qquad\qquad \forall v \in V \end{aligned}$$

The relaxation aims to remove the integer constraint over the $\widehat{x}_v$ variables, allowing to generate a solution whose result belongs to $\mathbb{R}$.

3.2 Rounding scheme and 2-Approximation to the best multi-set

Let's define $S_{\mathbb{Z}}$ and $S_{\mathbb{R}}$ the two solutions obtained respectively by the Integer Programming formulation and the Linear Programming relaxation. Now we can round the LP solution to a feasible integer solution S_r by applying the following function:

$$f(\widehat{x}) = \begin{cases} ceil(\widehat{x}_v) & \text{if } \widehat{x}_v \geq 0.5 \\ floor(\widehat{x}_v) & \text{if } \widehat{x}_v < 0.5 \end{cases}$$

We can prove how this rounding scheme is a 2-Approximation to the best multi-set obtained by the Integer Programming solution: $S_r \leq 2 \cdot S_{\mathbb{Z}}$. First of all we need to show that $\mathbf{S}_r \leq \mathbf{2} \cdot \mathbf{S}_{\mathbb{R}}$:

$$S_r = \sum_{v \in V} w_v \cdot f(\widehat{x}_v) \Rightarrow \begin{cases} \text{for } \widehat{x}_v \geq 0.5 & \Rightarrow w_v \cdot ceil(\widehat{x}_v) \leq w_v \cdot 2\widehat{x}_v & \textbf{TRUE} \\ \text{for } \widehat{x}_v < 0.5 & \Rightarrow w_v \cdot floor(\widehat{x}_v) \leq w_v \cdot 2\widehat{x}_v & \textbf{TRUE} \end{cases}$$

We move on and we show now the relation between the solutions that we have generated: if the defined rounding scheme produces a S_r whose weight is less than the weight of the optimal solution $S_{\mathbb{Z}}$, then S_r cannot be a correct solution by definition (the rounding scheme can't produce a solution better than the optimal one). Therefore it's possible to prove that $\mathbf{S}_{\mathbb{Z}} \leq \mathbf{S}_r$ just by demonstrating that S_r is a correct solution, since it satisfies all our constraints.

$$S_{\mathbb{Z}} > S_r \Rightarrow \neg\text{Correct Solution} \qquad \text{Correct Solution} \Rightarrow S_{\mathbb{Z}} \leq S_r$$

In order to prove that S_r is a correct solution we just need to verify if after applying the rounding scheme the constraints are satisfied. We can just analyze all the possible cases:

$$\begin{cases} 0 \leq \widehat{x}_u < 0.5 & \Rightarrow \widehat{x}_v > 1 & 2floor(\widehat{x}_u) + f(\widehat{x}_v) \geq 2 & \forall (u,v) \in E & \Rightarrow f(\widehat{x}_v) = ceil(\widehat{x}_v) \geq 2 \\ \widehat{x}_u \geq 0.5 & & 2ceil(\widehat{x}_u) + f(\widehat{x}_v) \geq 2 & \forall (u,v) \in E & \Rightarrow \quad ceil(\widehat{x}_u) \geq 1 \\ 0 \leq \widehat{x}_v < 0.5 & \Rightarrow \widehat{x}_u > 0.75 & 2f(\widehat{x}_u) + floor(\widehat{x}_v) \geq 2 & \forall (u,v) \in E & \Rightarrow f(\widehat{x}_u) = ceil(\widehat{x}_u) \geq 1 \\ 0.5 \leq \widehat{x}_v \leq 1 & \Rightarrow \widehat{x}_u \geq 0.5 & 2f(\widehat{x}_u) + ceil(\widehat{x}_v) \geq 2 & \forall (u,v) \in E & \Rightarrow f(\widehat{x}_u) = ceil(\widehat{x}_u) \geq 1 \\ \widehat{x}_v > 1 & & 2f(\widehat{x}_u) + ceil(\widehat{x}_v) \geq 2 & \forall (u,v) \in E & \Rightarrow \quad ceil(\widehat{x}_v) \geq 2 \end{cases}$$

We demonstrated that $S_r \leq 2 \cdot S_{\mathbb{R}}$ and $S_{\mathbb{Z}} \leq S_r$, but we also know by definition $S_{\mathbb{R}} \leq S_{\mathbb{Z}}$. It implies:

$$S_{\mathbb{R}} \leq S_{\mathbb{Z}} \leq S_r \quad \Rightarrow \quad \mathbf{2} \cdot S_{\mathbb{R}} \leq \mathbf{2} \cdot S_{\mathbb{Z}} \leq \mathbf{2} \cdot S_r \quad \Rightarrow \quad \mathbf{S}_r \leq 2 \cdot S_{\mathbb{R}} \leq 2 \cdot S_{\mathbb{Z}} \leq 2 \cdot S_r \quad \Rightarrow \quad \mathbf{S}_r \leq \mathbf{2} \cdot \mathbf{S}_{\mathbb{Z}}$$

Since we have proved that $S_r \leq 2 \cdot S_{\mathbb{Z}}$ it follows that the applied rounding scheme guarantees a 2-approximation to the best multi-set.

4 Exercise 4

The problem asks us to use the provided Rosenthal's Potential Φ in order to show that:

$$PoS = \min_{s \in NE} \frac{SC(s)}{SC(s_{OPT})} \leq \lambda \qquad \Rightarrow \qquad \min_{s \in NE} SC(s) \leq \lambda \cdot SC(s_{OPT})$$

The demonstration that we will provide uses the following approach:

1. First of all we thought about the given hint, focusing on the presence of two different states s and s_{OPT} such that $\Phi(s) \leq \Phi(s_{OPT})$, and then we analyzed the technique adopted in the slides to bound the Price of Anarchy *(cgt2)*.

2. We already know that $\Phi(s) \leq \Phi(s_{OPT})$, but given the fact that in a congestion game cost functions are non-decreasing, since the more players use a resource the greater the cost will be, we can even assume that the cost of a flow is always at least its potential function value. In particular if $\Phi(s) \leq SC(s)$ is true for every state, then $\Phi(S_{OPT}) \leq SC(s_{OPT})$ holds.

$$\Rightarrow \quad \Phi(s) \leq \Phi(s_{OPT}) \leq SC(s_{OPT})$$

3. We consider also the fact that finding a Nash equilibrium in an exact potential game (and hence in a congestion game) is equivalent to finding a local minimum of the potential function. This implies that the strategy profile $\alpha = \{\alpha_1, .., \alpha_n\}$ is an equilibrium if for each player i the strategy α_i minimizes his personal cost, given the strategies of the other players.
   It follows that if $\Phi(s_{NE})$ is the global minimum of the potential function, then s_{NE} is a Nash equilibrium state (proposition 3.3 *ctg2* on piazza).

$$\Rightarrow \quad \Phi(s_{NE}) \leq \Phi(s) \leq \Phi(s_{OPT}) \leq SC(s_{OPT})$$

4. The demonstration would be concluded if we can show that $SC(s_{NE}) \leq \lambda \cdot SC(s_{OPT})$, therefore we just need to prove that $SC(s) \leq \lambda \cdot \Phi(s)$ for any state.

$$\begin{aligned} SC(s) &= \sum_{i \in N} cost_i(s) \\ &= \sum_{i \in N} \sum_{r \in s_i} c_r(\#(r, \alpha)) \\ &= \sum_{r \in R} \#(r, \alpha) \cdot c_r(\#(r, \alpha)) \\ &= \#(r, \alpha) \sum_{r \in R} c_r(\#(r, \alpha)) \end{aligned} \qquad \begin{aligned} \Phi(s) &= \sum_{r \in R} \sum_{i=1}^{n_r(s)} c_r(i) \\ &= \sum_{r \in R} \sum_{i=1}^{\#(r,\alpha)} c_r(i) \\ &= \sum_{r \in R} c_r(1) + ... + c_r(\#(r, \alpha)) \\ &\geq \sum_{r \in R} c_r(\#(r, \alpha)) \end{aligned}$$

   Since we know by hypothesis that no resource is ever used by more than λ players, $\#(r, \alpha)$ will always be a value smaller or equal than λ and this implies:

$$SC(s) \; = \; \#(r, \alpha) \sum_{r \in R} c_r(\#(r, \alpha)) \;\; \leq \;\; \lambda \cdot \sum_{r \in R} c_r(\#(r, \alpha)) \;\; \leq \;\; \lambda \cdot \Phi(s) \quad \Rightarrow \quad SC(s) \leq \lambda \cdot \Phi(s)$$

5. Considering what we obtained in the third point of our process, we can multiply all terms by λ without loss of generality and add to the left side the lower bound that we just found.

$$\Rightarrow \quad SC(s_{NE}) \leq \lambda \cdot \Phi(s_{NE}) \leq \lambda \cdot \Phi(s) \leq \lambda \cdot \Phi(s_{OPT}) \leq \lambda \cdot SC(s_{OPT})$$

6. Now we can go back to the definition of price of stability and prove the statement: "the Price of Stability is at most λ" in this way:

$$SC(s_{NE}) \leq \lambda \cdot SC(s_{OPT}) \quad \Rightarrow \quad \frac{SC(s_{NE})}{SC(s_{OPT})} \leq \lambda \quad \Rightarrow \quad PoS \leq \lambda$$

5 Exercise 5

The exercise asks us to verify that the lower bound of the expected value of the proposed algorithm is equal to a fraction $1/d_{max}$ of the optimal solution, where d_{max} is the maximum degree of a vertex $v \in V$.

$$E[|ALG|] \geq \frac{1}{d_{max}} \cdot |OPT|$$

In order to prove that, we analyze the instance in which our randomized algorithm obtains the smallest number of vertices for an Independent Set and we will call it *Worst*. The key that allowed us to understand what is the size of *Worst* is through an approach really similar to the one used in class for solving a Vertex Cover 2-Approximation exercise (Chris' Grocery problem): in that case we focused on those vertices that the optimal solution would have taken, then we described how the expected value of the randomized algorithm that we were using would make him rent at most twice as many rooms as in the optimal one. For solving our exercise we focused instead on all those vertices that the optimal solution would **not** take.

Given a permutation σ of the vertices, for each vertex v, it belongs to the independent set S(σ) if and only if no neighbour of v precedes it in the permutation. The probability of all **isolated vertices** being included in the independent set is certain, therefore this type of vertices must be included in both the optimal solution and our solution. Instead, the probability of adding a vertex with a degree greater or equal than 1 must be equal to 1 (the number of favorable outcomes) over d_v+1 (the number of all possible outcomes, which is the number of the neighbours of the considered vertex plus itself). Therefore, we only consider non-isolated vertices, which are all the vertices that contribute to a difference between the optimal solution and the solution given by our algorithm: $K = \sigma - \{v_i \mid \forall v_i \in \sigma \text{ that is an isolated vertex}\} = \{v_1, ..., v_k\}$. So we can conclude by linearity of expectation that the expected size of $|Worst/I|$, which is the size of the smallest Independent Set that our randomized algorithm could obtain without considering isolated vertices, is the following:

$$E[|Alg/I|] = \sum_{i=1}^{k} Pr(i \in \text{Alg/I}) = \sum_{i=1}^{k} \frac{1}{d_i + 1} \geq \frac{k}{d_{avg} + 1} \geq \frac{k}{d_{max} + 1} = E[|Worst/I|]$$

Where d_{avg} is the average above all degrees.

Once that we have found the expected size of *Worst/I*, we can identify the size of *OPT/I* as well, since as previously stated it is composed at most by all those vertices not included in *Worst/I*.

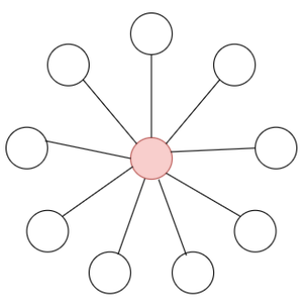
$$|OPT/I| \leq k - E[|Worst/I|] \leq k - \frac{k}{d_{max} + 1} = \frac{k \cdot d_{max} + k - k}{d_{max} + 1} = \frac{k \cdot d_{max}}{d_{max} + 1}$$

In this way we identify both the upper and the lower bound of the algorithm, so the best and the worst possible outcome that it could return. Now we can show how the lower bound is exactly $\frac{1}{d_{max}}$ fraction of the upper one:

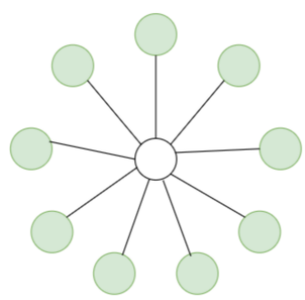
$$E[|ALG/I|] \geq E[|Worst/I|] \geq \frac{k}{d_{max} + 1} = \frac{1}{d_{max}} \cdot \frac{k \cdot d_{max}}{d_{max} + 1} \geq \frac{1}{d_{max}} \cdot |OPT/I|$$

So we have demonstrated that:

$$|I| + E[|ALG/I|] \geq \frac{|I|}{d_{max}} + \frac{|OPT/I|}{d_{max}} \quad \Rightarrow \quad E[|ALG|] \geq \frac{1}{d_{max}} \cdot |OPT|$$



$$E[|ALG/I|] = \sum_{i=1}^{k} Pr(i \in \text{ALG/I}) \geq E[|Worst/I|] = 1$$



$$|OPT/I| = k - E[|Worst/I|] \leq k - \frac{k}{d_{max} + 1} = 9$$

$$\Rightarrow E[|ALG/I|] \geq E[|Worst/I|] \geq \tfrac{1}{d_{max}} \cdot |OPT/I| = 1 \geq \tfrac{9}{9} = 1$$