

Problem Set 5: NP-Completeness and Approximations

Due: Friday 12/10 11:59PM

Feel free to work in groups of at most 4 for these - if you have a group of more than 4, please run it by me first. If you do work in a group, please include the names of those that you worked with. **However: each student should submit a separate copy where the solutions have been written by yourself.**

Our first goal is to show the following problem is NP-COMPLETE for $k \geq 3$:

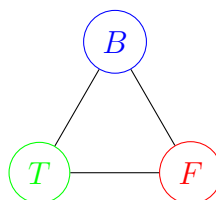
k -COLOR

INPUT: A graph G (you can assume it's just one connected component) OUTPUT: TRUE if you can color the graph in k colors so that no two adjacent vertices are colored the same.

1. (5 points) Show that 2-COLOR is in P
2. (10 points) Show that 3-COLOR is NP-COMPLETE.

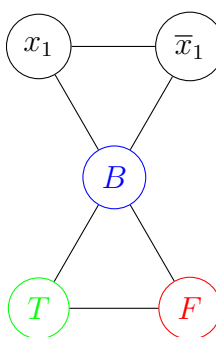
Some helpful thoughts below: suppose we have an instance for 3SAT with variables $x_1, \dots, x_n$ and clauses $C_1, \dots, C_m$:

- i) Consider the following graph:



Convince yourself that these three vertices must be colored 3 different colors in a valid coloring.

- ii) Consider the following subgraph:

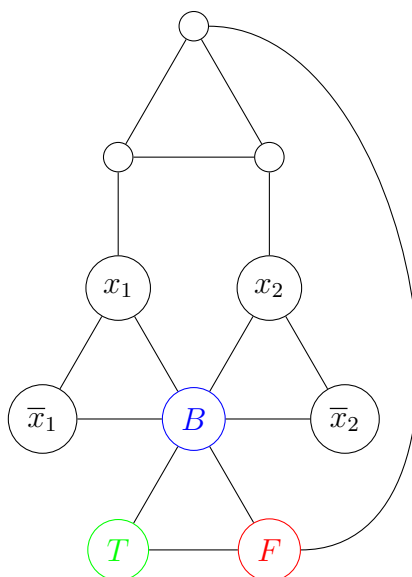


Convince yourself that one of x_1 or $\bar{x}_1$ is colored the same color as T and the other must be colored the same as F .

- iii) Consider the following subgraph:

Problem Set 5: NP-Completeness and Approximations

Due: Friday 12/10 11:59PM



Convince yourself that there's no valid coloring for this graph where both x_1 and x_2 are the same color as the F vertex.

The next problem is based on https://en.wikipedia.org/wiki/Wolf,_goat_and_cabbage_problem.

3. (10 points) Imagine we have a collection of n items to transport across a river and a boat that can hold ourselves plus k items, under the constraint that certain items cannot be left on the same side of the river as each other. Show that determining whether we are able to get all the items across the river is NP-COMPLETE.

Hint: You're free to reduce from whatever you want, but I think that VERTEX COVER might be the most straightforward?

The rest of the problem set is based around investigating the online k -cache problem. For this problem, imagine we are writing an algorithm overseeing a memory cache in a computer which our client is interacting with: sequentially the client will request specific “pages” of memory $r_1, r_2, \dots$. If the requested page is not in our memory cache at the moment of its request, we suffer a “miss” and must add it to our memory. However, we can only store k pages in our cache: if we are to add another page when we are already storing k pages, we have to evict a page. Our goal is to minimize how many misses we suffer.

Example: Suppose our request sequence is A, B, A, C, A and $k = 2$. Then the algorithm suffers two misses immediately for the first two requests, after which we have a memory cache of $[A, B]$. When A is requested again, we do not have a miss since A is still in our cache. When C is requested, we have to evict a page: suppose we evict A . Then when A is requested one last time, we suffer another miss since we have evicted it for C .

Months ago we showed that the algorithm LONGEST-FORWARD-DISTANCE (the algorithm that evicts the page that will be requested furthest in the future) is optimal. However, suppose we now consider this problem from an *online* setting: we can only base our decision of which pages to evict based on everything we've seen so far, and not what will come in the future.

Problem Set 5: NP-Completeness and Approximations

Due: Friday 12/10 11:59PM

4. (5 points) Convince me that no algorithm can expect to perfectly match the optimal solution on every input. That is, for every online algorithm, there is an input on which it incurs more misses than the optimal algorithm.

Knowing this, we still want to rank our algorithms by how well they perform: let $C(A, I)$ denote the number of misses an algorithm A incurs on input I and $C(\text{OPT}, I)$ the optimal number of misses on input I (ie, the number of misses LFD would have). We say A is $\rho(A)$ -competitive if there is some constant c so that for all possible inputs I we have that $C(A, I) \leq \rho(A)C(\text{OPT}, I) + c$ (in English, for every possible input the cost of our algorithm on that input is at most a factor of $\rho(A)$ worse than the optimal cost on that input plus some constant).

5. (5 points) Consider the algorithm LFU that always evicts the page that has been requested least overall so far. Convince me that $\rho(\text{LFU}) \geq x$ for every number x (an immediate consequence of this is that there's no way to upper bound the competitive ratio of this algorithm).
6. (10 points) Convince me that any algorithm must have competitive ratio at least k .
7. (10 points) Let's try to break a sequence of pages into phases: each phase is the maximal length sequence of page requests since the previous phase that contains at most k distinct page requests. Let's call an algorithm "good" if in any phase it only evicts pages not yet seen in the current phase. Convince me that any "good" algorithm has competitive ratio $O(k)$.
8. (5 points) Consider the algorithm that uses phases as defined as above: this algorithm only evicts pages not requested in this phase, chosen arbitrarily. Convince me that this algorithm is 'good' as defined above.
9. (10 points, EXTRA CREDIT) Let's say a randomized algorithm A is $\rho(A)$ competitive if $E[C(A, I)] \leq \rho(A)C(\text{OPT}, I) + c$ for all inputs I , where $E[C(A, I)]$ denotes the expected cost of A on input I . Consider the same algorithm as above, but it picks the pages to evict randomly. Show that this algorithm is $\log k$ -competitive.