CS 170 HW 12

Due **2021-11-22, at 10:00 pm**

1 Study Group

List the names and SIDs of the members in your study group. If you have no collaborators, you must explicitly write none.

In addition, we would like to share correct student solutions that are well-written with the class after each homework. Are you okay with your correct solutions being used for this purpose? Answer “Yes”, “Yes but anonymously”, or “No”

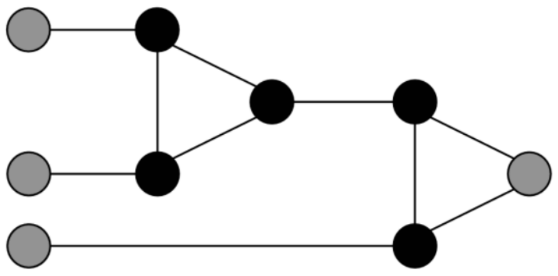
2 Reduction to 3-Coloring

Given a graph $G = (V, E)$, a valid 3-coloring assigns each vertex in the graph a color from $\{0, 1, 2\}$ such that for any edge (u, v), u and v have different colors. In the 3-coloring problem, our goal is to find a valid 3-coloring if one exists. In this problem, we will give a reduction from 3-SAT to the 3-coloring problem. Since we know that 3-SAT is NP-Hard (there is a reduction to 3-SAT from every NP problem), this will show that 3-coloring is NP-Hard (there is a reduction to 3-coloring from every NP problem).

In our reduction, the graph will start with three special vertices, labelled “True”, “False”, and “Base”, and the edges (True, False), (True, Base), and (False, Base).

(a) For each variable x_i in a 3-SAT formula, we will create a pair of vertices labeled x_i and $\neg x_i$. How should we add edges to the graph such that in any valid 3-coloring, one of $x_i, \neg x_i$ is assigned the same color as True and the other is assigned the same color as False?

(b) Consider the following graph, which we will call a “gadget”:



Show that in any valid 3-coloring of this graph which does not assign the color 2 to any of the gray vertices, the gray vertex on the right is assigned the color 1 only if one of the gray vertices on the left is assigned the color 1.

(c) We observe the following about the graph we are creating in the reduction:

(i) For any vertex, if we have the edges (v, False) and (v, Base) in the graph, then in any valid 3-coloring v will be assigned the same color as True.

(ii) Through brute force one can also show that in the gadget, for any assignment of colors to gray vertices such that:

(1) All gray vertices are assigned the color 0 or 1

(2) The gray vertex on the right is assigned the color 1

(3) At least one gray vertex on the left is assigned the color 1

Then there is a valid coloring for the black vertices in the gadget.

Using these observations and your answers to the previous parts, give a reduction from 3-SAT to 3-coloring. Prove that your reduction is correct.

Solution:

(a) We add the edges (x_i, $\neg x_i$), (x_i, Base) and ($\neg x_i$, Base). Since $x_i, \neg x_i$ are both adjacent to Base they must be assigned a different color than Base, i.e. they both are assigned either the color of True or the color of False. Since we added an edge between x_i and $\neg x_i$, they can't be assigned the same color, i.e. one is assigned the same color as True and one the same color as False.

(b) It is easier to show the equivalent statement that if all the gray vertices on the left are assigned the color 0, then the gray vertex on the right must be assigned the color 0 as well. Consider the triangle on the left. Since all the gray vertices are assigned 0, the two left points must be assigned the colors 1 and 2, and so the right point in this triangle must be assigned 0 in any valid coloring. We can repeat this logic with the triangle on the right, to conclude that the gray vertex on the right must be assigned 0 in any valid coloring.

(c) Given a 3-SAT instance, we create the three special vertices and edges described in the problem statement. As in part a, we create vertices x_i and $\neg x_i$ for each variable x_i, and add the edges we gave in the answer to part a. For clause j, we add a vertex C_j and edges (C_j, False), (C_j, Base). Lastly, for clause j we add vertices and edges to create a gadget where the three gray vertices on the left of the gadget are the vertices of three literals in the clause, and the gray vertex on the right is the vertex C_j (All black vertices in the gadget are only used in this clause's gadget).

If there is a satisfying 3-SAT assignment, then there is a valid 3-coloring in this graph as follows. Assign False the color 0, True the color 1, and Base the color 2; assign x_i the color 1 if x_i is True and 0 if x_i is False (vice-versa for $\neg x_i$). Assign each C_j the color 1. Lastly, fix any gadget. Since the 3-SAT assignment is satisfying, in each gadget at least one of the gray vertices on the left is assigned 1, so by the observation (ii) in the problem statement the gadget can be colored.

If there is a valid 3-coloring, then there is a satisfying 3-SAT assignment. By symmetry, we can assume False is colored 0, True is colored 1, and Base is colored 2. Then for each literal where x_i is color 1, that literal is true in the satisfying assignment. By part a, we know that exactly one of $x_i, \neg x_i$ is colored 1, so this produces a valid assignment. By observation (i), we also know every node C_j must be colored 1. All literal nodes are colored 0 or 1, so by part b, this implies that for every clause, one of the gray literal nodes in the clause gadget is colored 1, i.e. the clause will be satisfied in the 3-SAT assignment.

3 Multiway Cut

In the multiway cut problem, we are given a graph $G(V, E)$ with k special vertices $s_1, s_2 \dots s_k$. Our goal is to find the smallest set of edges F which, when removed from the graph, disconnect the graph into at least k components, where each s_i is in a different component. When $k = 2$, this is exactly the min s-t cut problem, but if $k \geq 3$ the problem becomes NP-hard.

Consider the following algorithm: Let F_i be the set of edges in the minimum cut with s_i on one side and all other special vertices on the other side. Output F, the union of all F_i. Note that this is a multiway cut because removing F_i from G isolates s_i in its own component.

(a) Explain how each F_i can be found in polynomial time.

(b) Let F^* be the smallest multiway cut. Consider the components that removing F^* disconnects G into, and let C_i be the set of vertices in the component with s_i. Let F_i^* be the set of edges in F^* with exactly one endpoint in C_i. How many different F_i^* does each edge in F^* appear in? How do the sizes of F_i and F_i^* compare?

(c) Using your answer to the previous part, show that $|F| \leq 2|F^*|$. (Challenge: How could you modify this algorithm to output F such that $|F| \leq (2 - \frac{2}{k})|F^*|$?)

(As an aside, consider the minimum k-cut problem, where we want to find the smallest set of edges F whose removal disconnects the graph into at least k components. The following greedy algorithm for minimum k-cut gets a $(2 - \frac{2}{k})$-approximation: Initialize F to the empty set. While $G(V, E - F)$ has less than k components, find the minimum cut within each component of $G(V, E - F)$, and add the edges in the smallest of these cuts to F. Showing this is a $(2 - \frac{2}{k})$-approximation is fairly difficult.)

Solution:

(a) Consider adding a vertex t to the graph and connecting t to all special vertices except s_i with infinite capacity edges. Then F_i is the minimum s_i-t cut, which we know how to find in polynomial time.

(b) Each edge in F^* appears in exactly two of the sets F_i^*.

Note that F_i^* is the set of edges in a cut which disconnects s_i from the other special vertices. Then by definition F_i has fewer edges than F_i^* since F_i is the minimum cut disconnecting s_i from all other special vertices.

(c) We combine the answers to the previous part and note that F's size is at most the total size of all F_i to get:

$$|F| \leq \sum_i |F_i| \leq \sum_i |F_i^*| = 2|F^*|$$

To get the $(2 - \frac{2}{k})$-approximation, after computing all F_i, we instead output F as the union of all F_i except for the one with the most edges. Let this be F_j. This is still a multiway cut because each s_j is still disconnected from all other s_i. Then:

$$|F| \leq \sum_{i \neq j} |F_i| \leq (1 - \frac{1}{k}) \sum_i |F_i| \leq (1 - \frac{1}{k}) \sum_i |F_i^*| = (2 - \frac{2}{k}) |F^*|$$

4 Project

Please answer the following questions after reading the project specification. They will help you build intuition for how to construct your inputs and solvers in the project.

(a) Construct a set of igloos polishing tasks such that the problem, as specified in the project specifications, has a trivial solution. Explain why the input is trivial.

(Hint 1: Your answer should focus on the relationship between the parameters of tasks, instead of the actual numerical values.)

(Hint 2: Try setting the deadlines of all tasks to 1440.)

Solution: Set of disjoint tasks, such that the duration of the $i + 1^{th}$ task is less than or equal to the difference between the deadline of the $i + 1^{th}$ task and the deadline of the i^{th} task.

(b) For this question **do not** use the profit decay function specified in the specification. Instead assume that the profit for polishing is all or nothing. If an igloo is polished by the deadline, the profit gained is equal to the corresponding p_i. Otherwise if an igloo is polished after the deadline, then the profit gained is 0.

Next, find the best sequence of igloos to polish in a total of **100 minutes**, given the following igloo polishing tasks:

- Igloo 1: Duration = 30, Deadline = 65, Profit = 10
- Igloo 2: Duration = 40, Deadline = 100, Profit = 20
- Igloo 3: Duration = 15, Deadline = 80, Profit = 5
- Igloo 4: Duration = 20, Deadline = 30, Profit = 30
- Igloo 5: Duration = 25, Deadline = 40, Profit = 15

(i) Find the optimal sequence of igloos to polish **Solution:** Igloo 4, Igloo 1, Igloo 2

(ii) If all igloos had identical deadlines, equal to 100, would that change the optimal sequence? Why? **Solution:** Igloo 4, Igloo 5, Igloo 2, Igloo 3 (in any order)

(c) For this question **do not** use the profit decay function specified in the specification. Instead, assume that our profit decays according to the following function, $max(0, p_i - s_i)$, where p_i is the original profit and s_i is the number of minutes by which the task was delayed past the deadline.

(i) Consider a greedy solver, where the task with the **highest benefit** is chosen at every step and scheduled as soon as possible, until we run out of time or tasks to schedule. Construct an input, of exactly 2 igloo polishing tasks, such that the greedy solver chooses a sub-optimal list of igloos. Explain why the greedy algorithm fails. **Solution:**

- Igloo 1: Duration = 1; Deadline = 2; Profit = 2;
- Igloo 2: Duration = 1; Deadline = 1; Profit = 1;

(ii) Consider a greedy solver, where the task with the **next earliest deadline** is chosen at every step, until we run out of time or tasks to schedule. Construct an input, of exactly 2 igloo polishing tasks, such that the greedy solver chooses a sub-optimal list of igloos. Explain why the greedy algorithm fails. **Solution:**

- Igloo 1: Duration = 2; Deadline = 1; Profit = 1;
- Igloo 2: Duration = 2; Deadline = 2; Profit = 2;

5 (Extra Credit) Approximation Hardness of Independent Set

Recall the maximum independent set problem: Given an undirected graph G, we want to find the largest independent set, i.e. largest set of vertices S such that no two vertices in S are adjacent. An algorithm is a α-approximation algorithm for maximum independent set if given a graph G, it outputs an independent set of size at least OPT/α, where OPT is the size of the largest independent set.

Show that if there is a polynomial-time $2^{2^{100}}$-approximation algorithm for maximum independent set, there is a polynomial-time 2-approximation algorithm for maximum independent set.

Solution: Given a graph G, we use the following "graph squaring" procedure to construct a graph G', such that if the size of the largest independent set in G is OPT, the size of the largest independent set in G' is OPT^2. G' has a vertex labelled $\{u, v\}$ for every ordered pair of vertices in G (including repeat pairs, such as (v, v)). G' has an edge between $\{u_1, v_1\}$ to $\{u_2, v_2\}$ if at least one of (u_1, u_2) and (v_1, v_2) is an edge in G.

If an independent set S of size $|S|$ exists in G, one of size $|S|^2$ exists in G'. We simply include all vertices $\{u, v\}$ in G' for which $u, v \in S$. Since S was an independent set, the set of edges we included in G' ensures this is also an independent set.

If an independent set S of size $|S|$ exists in G', one of size at least $\sqrt{|S|}$ exists in G. Let S_1 be the set of all vertices in G that are "first coordinates" of vertices in S. e.g. if $\{u_1, v_1\}$ is in S, then u_1 will be in S_1. Define S_2 similarly, but for the second coordinate. We have $|S| \leq |S_1||S_2|$, i.e. one of S_1, S_2 is size at least $\sqrt{|S|}$. Furthermore, both S_1, S_2 are independent sets in G since S is an independent set in G', which means none of the first (or second) coordinates in S are adjacent in G.

Given a $2^{2^{100}}$-approximation algorithm for maximum independent set, we can design a 2-approximation algorithm as follows: Take G with maximum independent set of size OPT, and apply graph squaring to G 100 times. This gives a graph G' such that the largest independent set in G' has size $OPT^{2^{100}}$. Furthermore, G' has size $O((|V|+|E|)^{2^{100}})$, so we can construct it in polynomial-time. Now run the $2^{2^{100}}$-approximation algorithm for maximum independent set on G' to find an independent set of size $(OPT/2)^{2^{100}}$. Using the procedure from the previous paragraph, we can retrieve from this an independent set of size at least $OPT/2$ in G.

Comment: In essence, this shows that if it is NP-Hard to 2-approximate independent set, it is NP-Hard to $2^{2^{100}}$-approximate independent set or really, α-approximate independent set for any constant α. This is a very simple example of a technique known as gap amplification, which is fundamental in the area known as "hardness of approximation", which seeks to show that finding α-approximation algorithms for some problems is as hard as solving the problem exactly.