CS 170 HW 11

Due on 2018-11-11, at 9:59 pm

1 (★) Study Group

List the names and SIDs of the members in your study group.

2 (★★★) 2-SAT and Variants

In this problem we will explore the variant of 3-SAT called 2-SAT, where each clause contains at most 2 literals (hereby called a 2-clause).

(a) Show that 2-SAT is in P. (Hint: Note that the clause $x \vee y$ is equivalent to the implications $\neg x \Rightarrow y$ and $\neg y \Rightarrow x$. If a 2-SAT formula is unsatisfiable, what must be true about the set of all such implications?)

(b) The problem of Max-2-SAT is defined as follows. Let $C_1, \ldots, C_m$ be a collection of 2-clauses and k a non-negative integer. We want to determine if there is some assignment which satisfies at least k clauses.

The problem of Max-Cut is defined as follows. Let G be an undirected unweighted graph, and k a non-negative integer. We want to determine if there is some cut with at least k edges crossing it. Max-Cut is known to be NP-complete.

Show that Max-2-SAT is NP-complete by reducing from Max-Cut. Prove the correctness of your reduction.

Solution:

(a) Let φ be a formula acting on n literals $x_1, \ldots, x_n$. Construct a graph with $2n$ vertices representing the set of literals and their negations. For each clause $(a \vee b)$ of φ add the edges $\neg a \Rightarrow b$ and $\neg b \Rightarrow a$. Use the strongly connected components algorithm and for each i, check if there is a SCC containing both x_i and $\neg x_i$. If any such component is found, report unsatisfiable. Otherwise, report satisfiable.

In the case that the algorithm reports unsatisfiable, notice that the edges of the graph are necessary implications. Thus, if some x_i and $\neg x_i$ are in the same component, there is a chain of implications which is equivalent to $x_i \to \neg x_i$ and a different chain which is equivalent to $\neg x_i \to x_i$, i.e. there is a contradiction in the set of clauses.

In the case that the algorithm reports satisfiable, there is a simple satisfying assignment: Take any sink component, and assign variables so all the literals in this component are True. Because of how we define the graph, there is a corresponding source component which has the negations of all literals in this component. Remove this source/sink component pair, and repeat the process until the graph is empty. Since we set components to true in reverse topological order, there is no implication from a true literal to a false

literal. Since no literal and its negation are in the same SCC, we never try to set a variable to be both true and false. So this produces an assignment satisfying all clauses.

(Note: A common mistake is to report unsatisfiable if there is a path from x_i to $\neg x_i$ in this graph, even if there is no path from $\neg x_i$ to x_i. Even if there is a series of implications which combined give $x_i \to \neg x_i$, unless we also know $\neg x_i \to x_i$ we could set x_i to False and still possibly satisfy the clauses. For example, consider the 2-SAT formula $(\neg a \vee b) \wedge (\neg a \vee \neg b)$. These clauses are equivalent to $a \to b, b \to \neg a$, which implies $a \to \neg a$, but this 2-SAT formula is still easily satisfiable.)

(b) As suggested, we reduce unweighted `Max-Cut` to `Max-2-SAT`. Let a graph $G = (V, E)$ and an integer k be given. We want to find whether there is a cut in the graph of size k or more. We construct a boolean formula based on the graph. Every assignment of this formula will correspond to a cut in the graph.

For each vertex $u \in V$, we add a variable x_u, representing whether u is in S (true) or in $V \setminus S$ (false). For each edge $(u, v) \in E$, we add *two* clauses: $(x_u \vee x_v)$ and $(\overline{x_u} \vee \overline{x_v})$. Then, if (u, v) crosses the cut, *both* of these clauses will be satisfied (if $u \in S$ and $v \notin S$, then x_u and $\overline{x_v}$ are both true; if $u \notin S$ and $v \in S$, then $\overline{x_u}$ and x_v are both true). If (u, v) does not cross the cut, then exactly one of the clauses will be satisfied (the first if both u and v are in S, the second if neither is). Thus, any cut of size q in the graph corresponds to an assignment of values to variables that satisfies exactly $|E| + q$ clauses (1 for each of the $|E| - q$ pairs representing non-cut edges, 2 for each of the q pairs representing cut edges).

Thus, to solve `Max-Cut`(G, k), we construct this formula Φ_G and return `Max-2-SAT`$(\Phi_G, |E| + k)$.

Lastly, we know that `Max-2-SAT` is in NP because given an instance and an assignment of variables, we can just iterate through the clauses and count how many are satisfied in polynomial time.

3 (★★★) Reduction to (3,3)-SAT

Consider the (3,3)-SAT problem, which is the same as 3-SAT except each literal or its negation appears *at most* 3 times across the entire formula. Notice that (3,3)-SAT is reducible to 3-SAT because every formula that satisfies the (3,3)-SAT constraints satisfies those for 3-SAT. We are interested in the other direction.

Show that 3-SAT is reducible to (3,3)-SAT. By doing so, we will have eliminated the notion that the "hardness" of 3-SAT was in the repetition of variables across the formula. Give a precise description of the reduction and prove its correctness.

Solution: Assume that the variable x_i or $\neg x_i$ appears $k > 3$ times. Let us replace the k occurances of x_i with $x_i^{(j)}$ for $j = 1, \ldots, k$. We now have used each literal $x_i^{(j)}$ exactly once. We additionally add the clauses $\neg x_i^{(j)} \vee x_i^{(j+1)}$ and $\neg x_i^{(k)} \vee x_i^{(1)}$ for each $j = 1, \ldots, k-1$.

Recall that $\neg x_i^{(j)} \vee x_i^{(j+1)}$ is equivalent to $x_i^{(j)} \Rightarrow x_i^{(j+1)}$ and $\neg x_i^{(j+1)} \Rightarrow \neg x_i^{(j)}$, so the collective chain enforces that $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(k)}$.

Make the replacement for each literal that occurs more than 3 times ensures that all literals in the new formula occur at most 3 times. This is only a polynomial increase in the size of the problem description.

A satisfying assignment $\{x_i = \alpha_i\}$ for the original formula can be translated to an assignment for the new formula by setting $x_i^{(j)} = \alpha_i$ by the previous argument. For the other direction, a satisfying assignment for the new formula must have $x_i^{(1)} = x_i^{(2)} = \ldots = x_i^{(k)} = \alpha_i$, so we can set $x_i = \alpha_i$ in the original formula.

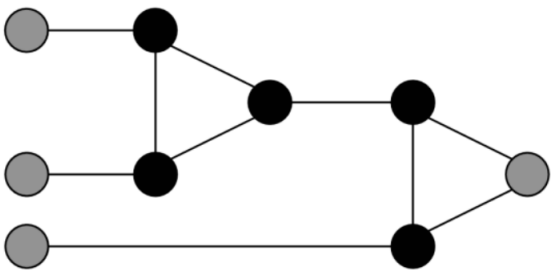
4 (★★★) Reduction to 3-Coloring

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, showing that 3-coloring is NP-hard.

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 labelled 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 other is assigned the same color as False?

(b) Consider the following graph, which we'll 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) to 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 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

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 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's 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 1 if x_i is True and x_i 0 if x_i is False (vice-versa for $\neg x_i$). Assign each C_j 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.

5 (★★) Coloring: Two's Company, Three's a Crowd

In the last problem, you proved that 3-coloring is NP-hard. For any $k \geq 2$, consider the k-coloring problem, which is the same as the 3-coloring problem except that there are k colors to choose from. In this problem, we'll see how the hardness of the problem is affected by the value k.

(a) Give an efficient algorithm for the 2-coloring problem. Just a brief description of the algorithm suffices.

(b) For any $k \geq 3$, give a polynomial-time reduction from the k-coloring problem to the $(k+1)$-coloring problem. Just a brief description of the reduction suffices. (By induction, this shows that for any constant $k \geq 3$, the k-coloring problem is NP-hard.)

Solution:

(a) Use DFS from an arbitrary vertex r to find a spanning tree. All vertices which are an even distance from r in this tree are given one color, and all vertices which are an odd distance from r in this tree are given the other color. If this causes the coloring constraints to be violated, then no coloring exists.

(For completeness, here's the proof of correctness: Suppose we find a coloring such that some u, v are adjacent and the same color. The path from u to v in the spanning tree must be even length, which means the cycle formed by (u, v) and this path is odd length. But an odd-length cycle cannot be 2-colored, so the graph is not colorable.)

(b) To reduce the problem of k-coloring G to the problem of $(k+1)$-coloring some graph G', we just set G' to be a copy of G, with a new vertex s and edges from s to every vertex in G added to the graph. Since no vertex besides s in G' can be the same color as s, in any $(k+1)$-coloring of G', at most k colors are used to color the vertices in G, giving a k-coloring of G. Similarly, any k-coloring of G is easily extended to a $(k+1)$-coloring of G'.

6 (★★★) Vertex Cover Approximation

In the textbook, we saw that finding a maximal matching and outputting all endpoints of edges in the matching is a 2-approximation for the vertex cover problem, i.e. the size of the solution output by this algorithm is at most twice the size of the best solution. In this problem, we'll give another 2-approximation. For a graph $G(V, E)$, consider the following linear program for the vertex cover problem:

$$\begin{aligned} \min \sum_{v \in V} x_v & \\ \text{s.t. } \forall (u,v) \in E: \quad x_u + x_v & \geq 1 \\ \forall v \in V: \quad x_v & \geq 0 \end{aligned}$$

(In an integer solution, we would set $x_v = 1$ if we include v in our vertex cover and $x_v = 0$ otherwise, and we have the constraint that for every edge we must include one of its endpoints.)

(a) How does the optimal value of this linear program compare to the size of the minimum vertex cover? (Note that the optimal solution to the linear program does not necessarily set every x_v to 0 or 1.)

(b) Describe a procedure which given only the optimal solution to the linear program, outputs a vertex cover of size at most twice the size of the minimum vertex cover. Prove both that your procedure finds a feasible vertex cover and the size guarantee. No runtime analysis needed.

(Hint: Your procedure should output a solution similar to the LP solution, i.e. it should only include vertices for which x_v is sufficiently large)

Solution:

(a) Consider any vertex cover C. If we set $x_v = 1$ for each $v \in C$ and $x_v = 0$ for each $v \notin C$, we get a feasible solution to the LP. In particular, if we apply this to the minimum vertex cover C^*, we get an LP solution where $\sum_{v \in V} x_v = |C^*|$. Any feasible LP solution gives an upper bound on the optimal LP solution's value, so the optimal value of the LP is at most $|C^*|$.

(b) The procedure is to output the set S of all v such that $x_v \geq .5$.

To show feasibility: For any edge (u, v), we know $x_u + x_v \geq 1$. So one of x_u, x_v is at least .5, and thus one of u, v appears in S. So S is a feasible vertex cover.

To show the size of this vertex cover is at most twice the size of the minimum vertex cover, by the previous part it suffices to show the size of this vertex cover is at most twice the optimal value of the LP. We can think of S as a solution x^* to the LP where $x^*_v = 1$ if $v \in S$ and $x^*_v = 0$ otherwise, in which case for all v, $x^*_v \leq 2x_v$. Then $|S| = \sum_{v \in V} x^*_v \leq 2 \sum_{v \in V} x_v$.

(Note: One can actually show that any basic feasible solution to the LP sets every x_v to a value in $\{0, .5, 1\}$, which means solutions like "include every vertex where x_v is positive" or "include every vertex where $x_v > .01$" are *technically* correct since they're identical to the above solution. However, showing this is not particularly straightforward, and would be necessary to get most of the credit for such a solution.)