6.890: ALGORITHMIC LOWER BOUNDS Prof. Erik Demaine
Fall 2014

Problem Set 3

Due: Wednesday, October 22nd, 2014

Problem 1. A Tour of Hamiltonicity Variants
For each of the following variations on the Hamiltonian Cycle problem, either prove it is in P by giving a polynomial time algorithm, or prove it is NP-hard.

(a) Given a simple graph, is there a cycle that visits every node at least once and no more than twice?

Solution: We will prove this is NP-hard by a simple reduction from Hamiltonian Cycle. We take the original input graph and to every vertex we attach two extra nodes in a triangle. If there was a Hamiltonian Cycle in the original graph, we can solve this problem by traversing the original nodes in that order and traversing the triangle at every node, visiting the auxiliary nodes once and the original nodes twice. To visit the auxiliary nodes, we are forced to visit each node twice, once to enter the triangle and once to exit. Thus if there is not a Hamiltonian Cycle, we would either be forced to skip a triangle or visit an original node more than once.

Many of the given solutions add a single leaf to every node. This is also correct, and it is interesting to note that this reduction is parsimonious, whereas the triangle gadget is not, since each can be traversed in two different directions. □

(b) Given a simple directed graph, is there a path that visits every node at least once, but does not cross an edge more than once?

Solution: We will reduce from finding Hamiltonian paths in directed graphs. We replace each node with two nodes, one with all the incoming edges and one with all the outgoing ones, connected by an edge. Then, if we ever return to a node we cannot recross the edge to the outgoing node and thus cannot leave this node. □

(c) Given a simple graph, is there a path that visits at least half the nodes, but never visits a node more than once?

Solution: We will reduce from finding Hamiltonian paths in a graph. Create four copies of the graph and hook them up to a central node. You can only get to two graph copies and thus you must decide if one copy is Hamiltonian.

This problem was intended to be on a connected graph, but this was not specified. If the graph need not be connected, there is a far simpler solution of creating two disconnected copies of the original graph to be solved. Thus if there is a Hamiltonian Path in one, then one finds a Hamiltonian Path in exactly half of the nodes. □

(d) Given a simple dense graph ($|E| = \Theta(|V|^2)$), is there a Hamiltonian Cycle?

Solution: We reduce from the Hamiltonian Path Problem with a specified starting node s and specified ending node t. We construct a clique of even size $\Theta(n)$, then connect two different nodes in the clique to each of the two nodes s and t with a single edge. The clique trivially has a Hamiltonian Path from any one node to any other, and we've added $\Theta(|V|)^2$ edges to the graph. Since the clique is only connected to the rest of the graph at the nodes s and t, the new graph will only have a Hamiltonian Cycle if the original graph had a Hamiltonian Path from s to t. □

Problem 2. Quartet Ordering
Suppose that you are given a set of elements $U = \{u_1, \ldots, u_n\}$ and a set of ordered quartets $Q = \{\langle a_1, b_1, c_1, d_1\rangle, \ldots, \langle a_k, b_k, c_k, d_k\rangle\}$ containing elements of U. For each quartet $\langle a_i, b_i, c_i, d_i\rangle$, the four elements must be distinct. Prove that it is NP-complete to find an ordering for U such that for each $\langle a_i, b_i, c_i, d_i\rangle$, either a_i and b_i are the extremes of the quartet (minimum and maximum, not necessarily in that order), or c_i and d_i are the extremes of the quartet.

Solution: We solve this problem by reducing from Betweenness. To simulate the constraint forcing C to lie between A and B, create two new elements x and y, and use the quartets (A, B, C, x), (A, B, C, y), (A, B, x, y). If C is not between A and B, then both x and y must lie on the opposite side of A and B from C. This contradicts the constraints of the third quartet, so our assumption must be false, and C must lie between A and B. Hence, any solution to the Quartet Ordering problem will result in a solution to the original Betweenness Problem. Furthermore, as long as C lies between A and B, x and y can be placed wherever inside the $[A, B]$ range, so any Betweenness solution can be converted to a corresponding Quartet Ordering solution by placing the newly created elements for each quartet adjacent to the middle element C. □

Problem 3. *n* Days Later
Suppose that you are given a grid graph, representing a network of cities connected by roads. Several of the cities are marked as infected. One city is marked as the player's starting position. The player has two possible actions: place the city that they're in under quarantine (which keeps that city from becoming infected in the future), or move to an adjacent city. Each day, two things happen in sequence: the player performs a single action, and then the infection status of each city is updated according to the following rules:

R1. If the city was infected at the start of the day, it remains infected.

R2. If the city is not under quarantine and is adjacent to another city that was infected at the start of the day, it becomes infected.

R3. All other cities remain uninfected.

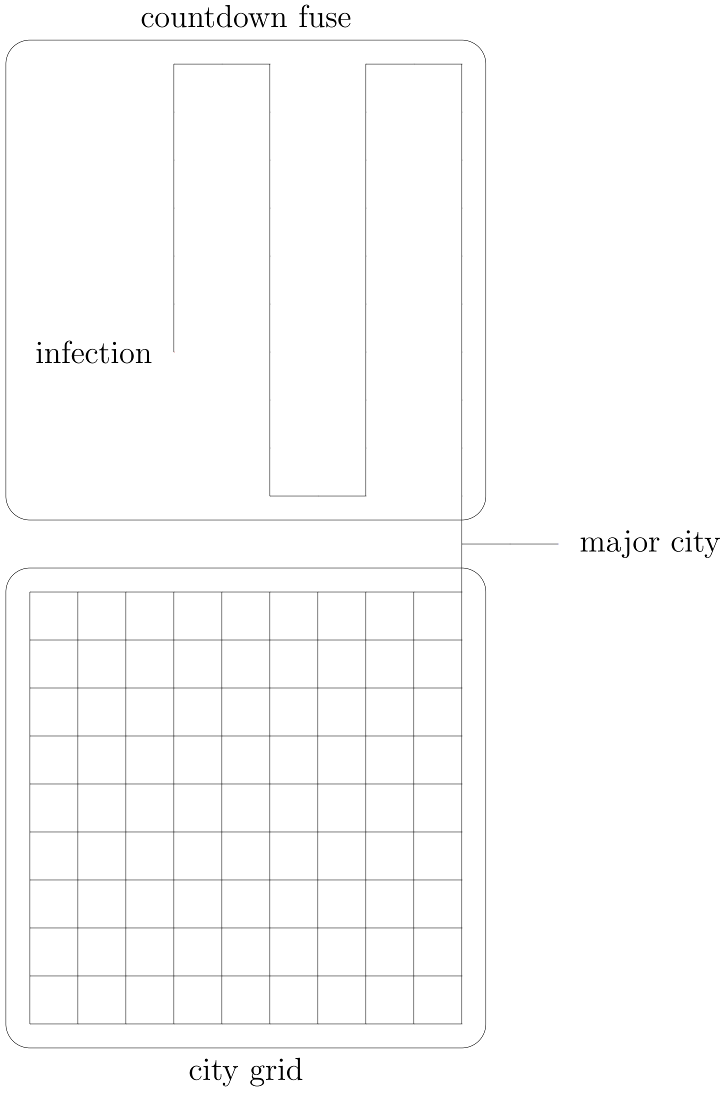
The scenario is over when the infection cannot spread any further. Prove that it is NP-hard for the player to maximize the number of cities left uninfected.

Solution: Reduction from the Hamiltonian Path problem in a grid graph with maximum degree 3. Let n be the number of points in the graph, and let m be the number of edges. Let k be a very large number, such as $100n^{100} + 100$. We start by scaling up our grid graph, placing a point (x, y) at position (kx, ky). Call these the "major cities." For each edge connecting two nodes in the original graph, place a path of $(k - 2)$ nodes between the corresponding nodes in the scaled graph.

Call these the "connecting cities." The resulting graph will have $n + m(k-2)$ cities in total. Note that a Hamiltonian path in the original graph, consisting of n nodes and $n-1$ edges, corresponds to a path of $k(n-1)$ edges and $k(n-1)+1$ vertices in this scaled graph.

Because the graph that we started with has maximum degree 3, we know that for each city (x, y) in the original graph, either $(x-1, y)$, $(x+1, y)$, $(x, y-1)$, or $(x, y+1)$ doesn't have a city placed there. Suppose that the city $(x-1, y)$ doesn't exist. Then we can use that space to the west of the major city (kx, ky) in the scaled graph to construct a gadget.

The gadget that we use consists of three sections: the city grid, the countdown fuse, and the connector. The grid is just that: a grid of $(k/2-4) \times (k/2-4)$ cities that the player has to save. The countdown fuse is a path consisting of $\ell = k(n-1)+1+5n$ nodes, arranged within a square of dimensions $(k/2-4) \times (k/2-4)$ by zig-zagging back and forth. (Note that by using alternating columns, connected at the appropriate end, we are guaranteed to get a path containing at least $(k/2-4)^2/2 = k^2/8 - 2k + 8$ nodes, which is more than enough to fit a fuse of that length.) One end of the countdown fuse starts out infected; the other end is attached to the city grid using one connector node. Finally, the other connector node is used to join the gadget to one of the major cities. The following figure shows an example of what this gadget looks like:



Note that for every city in the grid, there exists a path consisting of $\leq \ell + 2 + 2(k/2 - 4 - 1) = \ell + k - 8$ edges to the initially infected city at the other end of the countdown fuse. That path only uses cities inside the gadget. Hence, if the player doesn't quarantine at least one of the cities inside the gadget by day $\ell + k - 8$, the entire city grid will be overrun.

We construct one of these gadgets for each major city, rotated to fit into the space that is guaranteed to exist near that city (as argued above). Our goal will be to allow at most $n\ell$ of those cities to be sacrificed. By construction, this means that we sacrifice at most:

$$\begin{aligned} n\ell &= n(k(n-1) + 1 + 5n) \\ &= n^2k - nk + n + 5n^2 \\ &< n^2k - nk + k + k < n^2k \\ &< (k/4 - 4)k = k^2/4 - 4k \\ &< (k/2 - 4)^2 \end{aligned}$$

So we cannot allow any single city grid to become fully infected. This means that in any valid solution to the problem we have constructed, we must quarantine at least one node in each gadget by day $\ell + k - 8$. Each gadget is connected to the rest of the graph only through a major city, so this means that our solution must visit every major city in the first $\ell + k - 8$ turns. Furthermore, for each gadget, the player must spend at least 2 turns walking from the major city into the gadget, at least 1 turn quarantining a city, and at least 2 turns walking out again. This requires a total of at least $5n$ turns, leaving the player with only $\ell + k - 8 - 5n = k(n-1) + 1 + 5n + k - 8 - 5n = kn - 7$ turns to navigate between major cities. The distance (number of edges) between any pair of major cities is always a multiple of k, so in those $kn - 7$ turns, the player can make at most $\lfloor (kn-7)/k \rfloor = n - 1$ trips between cities, and each of those trips must have length k (and therefore must correspond to an edge in the original graph). Hence, the sequence of cities visited gives us a Hamiltonian Path in the original graph.

To complete the proof, we wish to show that if we have a Hamiltonian Path in the original graph, we can convert it to a sequence of moves for the player that will result in at most $n\ell$ cities becoming infected. The player's sequence of moves is constructed as follows. First, take the Hamiltonian Path, and convert it into a path navigating from major city to major city, using the connecting cities to travel between them. This requires $k(n-1)$ moves. Modify this sequence of moves so that whenever the player first arrives at a major city (and at the beginning, in the starting city), they should take two moves out into the gadget, then quarantine the connector city, then move back to the major city again. This increases the number of moves to $k(n-1) + 5n$. Note, however, that because the last two actions of this sequence are movement, the player's last quarantine occurs on day $k(n-1) + 5n - 2$. The length of each countdown fuse is $\ell = k(n-1) + 1 + 5n$, so the infection will not make it out of the countdown fuse before the player has finished quarantining all of the necessary cities. Hence, this plan will allow the player to save every city except for the $n\ell$ cities used for the countdown fuse. □

Problem 4. Cheating on Puzzles

Sometimes puzzles are too hard to solve exactly. What if we're satisfied with a solution that's "almost" correct?

Recall that a *Light Up* puzzle[1] consists of a board with black squares and white squares. Some of the black squares are labeled with a number. For this problem, we also specify the number of

[1]Refer to Lecture 6 and http://www.mountainvistasoft.com/docs/lightup-is-np-complete.pdf

lights to be placed. Two white squares on this board can *see* each other if they lie in the same row or column, and they do not have a black square between them. In a solution, lights are placed in some of the white squares, subject to the following constraints:

C1. Every white square can see a light.

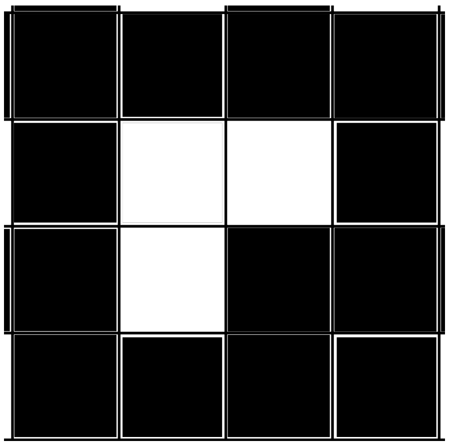
C2. No two lights can see each other.

C3. Every black square labeled with value k has exactly k lights adjacent to it.

Here we consider what happens when a solution is allowed to violate these constraints. Define the *cost* of a solution to be the number of violated constraints, i.e., the number of white squares that cannot see a light, plus the number of pairs of lights that can see each other, plus the number of black squares with incorrect labels.

(a) Given a *Light Up* puzzle, prove that it is NP-hard to decide whether there is a solution that violates at most an ε fraction of the constraints, for any constant $\varepsilon > 0$.

Solution: The general strategy here will be to make a copy of our original Light Up puzzle and then introduce constraints that must be violated. First, we will deal with the fact that now we have exactly k lights, rather than any number as in the original Light Up puzzle. To deal with this we introduce the L gadget, simply three white squares arranged in an L shape with no numbered black squares near them. This can be fully satisfied by 1 or 2 lights, and thus allows more flexibility in potential solutions with respect to k. Capping the number of lights at k was done because Constraint 2 is quadratic in the number of lights placed. This causes significant complications if one assumes a numbered black square must be next to a white square.

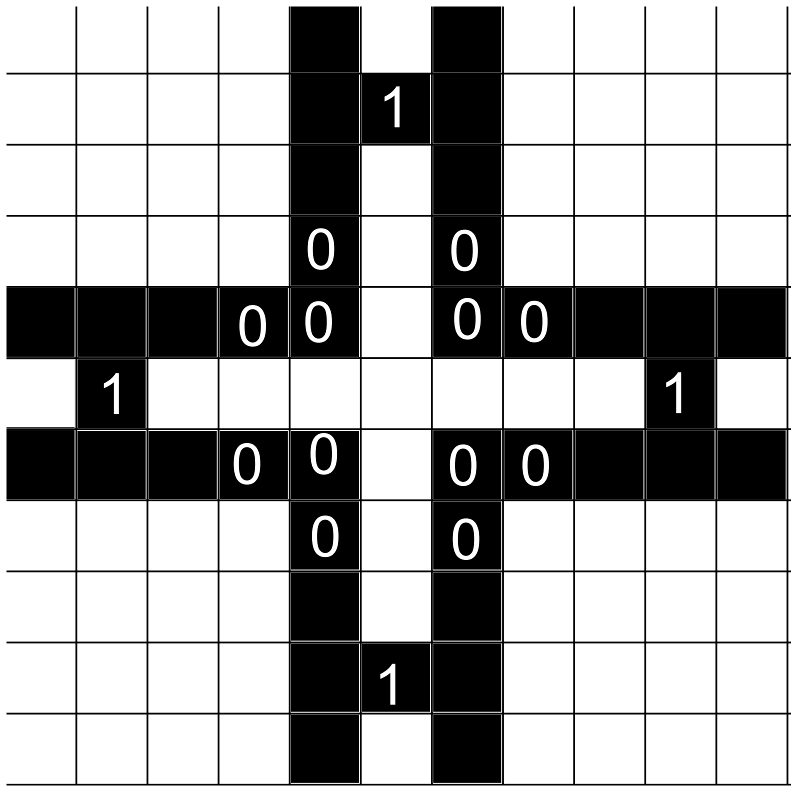


Now we need gadgets to adjust the number of constraints that must be violated. Most important, we introduce a gadget that forces a violation of a constraint. The simplest way to do this is have a positive numbered black square surrounded only by black squares. There is no possible way to satisfy that numbered square and thus we will incur one violation for each copy. Other examples include isolated white squares (since k is fixed, we simply cannot place enough lights to satisfy all of them), white squares surrounded by zeros, numbered black squares where the number is greater than the number of white squares adjacent to it, and white squares adjacent to black squares with different numbers. These different gadgets may introduce different numbers or ratios of violated to unviolated constraints, so some care will need to be taken in the next step of the analysis depending on which is chosen. Second, we need to be able to add extra satisfied constraints. We do this by creating a long row of b white squares surrounded by non-numbered black squares. This allows a single light to satisfy b constraints. Let the number of constraints that are satisfied by our original Light Up problem be C. We now construct a new problem with a copies of the unsatisfiable gadget, a length b satisfiable gadget, and a number of lights $k' = k + 1$. Now the ratio of satisfied to unsatisfied constraints needed to solve the problem is $\frac{a}{C+a+b}$. It is easy to see this is continuous on the open interval $(0, 1)$ and thus we can achieve any fraction ϵ of violated constraints. □

(b) Given a *Light Up* puzzle, prove that it is APX-hard to minimize the cost of a solution. (Hint: first convert the Circuit SAT reduction from class to a reduction from 3SAT.)

Solution: Several students pointed out that it is NP-hard to distinguish between 0 violations and 1 violation, and thus the problem is trivially APX-hard. This is similar to why we see Max 3SAT but not Min 3SAT in approximation lower bounds. However, we now present an example of a proof in the intended manner.

First we adapt the proof of *Light Up* by using two *OR* gadgets to form clauses and terminating all of the output wires with True. This gives a sequence of clauses, each with three literals, all of which must be satisfied. It is now a satisfiability problem in 3-CNF. We can lay out the problem in a grid, with variables along one axis and clauses along the other. Each variable is split into three wires, moves to the appropriate column for the clause, and heads down.



We will prove this is APX-hard by an L-reduction from Max E3SAT-3. First, we look at the gadgets in *Light Up*. The Parity, Wire, and Split gadget can only shift the value along that one wire. Violating the constraints in a wire cannot impact more than a single variable. Violating the constraints in a clause will only resolve up to one clause. Crossover gadgets, which are required to complete the proof, can be easily constructed using two intersecting hallways. Violating the constraints in a crossover gadget affects the values of at most two variables. So any violation lets us change at most one variable — which, because we are reducing from Max E3SAT-3, means that any violation allows us to (incorrectly) satisfy at most six clauses. Since it isn't useful to alter the wires more than once, OPT will not increase with the wire length. Thus we have a linear relationship between OPT for *Light Up* and OPT for Max E3SAT-3. The cost also only changes by a constant factor, so this will enable an L-Reduction. □