# Example solution to assignment 3

David Hammer

May 20, 2020

This is intended to serve as an example solution for the assignment. It includes the details I would look for in a thorough answer. Note that this is by no means perfect: it could definitely be more concise and I have included comments that are not strictly necessary (only there to point out that there are more details one could consider).

## Problem 1

For the regular "Cave Tunnels" (CT) problem, an instance has the form $(G = (V, E), k)$ where $G$ is the graph corresponding to the given map of tunnels: $V$ is the set of rooms with $|V| = n$ and $\{u, v\} \in E$ if the rooms $u$ and $v$ are connected by a tunnel. The regular CT problem is to find the maximum number of tunnels which can be controlled by placing $k$ blockers on nodes of the given graph.

### a

To get a decision problem, we add a target value. An instance of "Decide Cave Tunnels" (DCT) consists of $(G, k, t)$ and the question is whether the $k$ blockers can be placed on rooms of $G$ such that at least $t$ tunnels are controlled.

We first show that DCT is in NP. Suppose a certificate is given which indicates which rooms to place blockers in; that is, some $S \subseteq V$ with $|S| = k$. It is then trivial to verify in polynomial time that there are at least $t$ tunnels present in $E$ connecting rooms in $S$—this could for example be done by querying $E$ for each of the $\binom{|S|}{2}$ potential edges.

We now show that DCT is complete in NP by providing a reduction from the known NP-complete problem CLIQUE (theorem 34.11 in Cormen). Given an instance $(G, k)$ of CLIQUE, construct an instance of DCT of the form $\left(G, k, t = \binom{k}{2}\right)$. That is, the same graph is used and we will argue that there exists a clique of size $k$ in $G$ if and only if $k$ blockers can be placed such that they control $t$ tunnels.

Suppose there is a clique of size $k$ in $G$. Placing all $k$ blockers on the vertices of this clique will yield a solution where all tunnels (edges) among rooms (vertices) of the clique are controlled. As a clique is by definition a complete (sub)graph, there will be exactly $\binom{k}{2} = t$ controlled tunnels.

Suppose now that here is a way to place $k$ blockers on a subset $S$ of $G$ with $|S| = k$ such that $t$ tunnels are controlled. As both endpoints of a tunnel must be blocked for a tunnel to be controlled, all the $t$ controlled tunnels are between rooms in $S$. The largest possible number of controlled tunnels must therefore be $\binom{|S|}{2} = t$, requiring $S$ to be a complete (sub)graph. This means that $S$ is indeed a clique of size $k$.

This concludes the proof of the correctness of the reduction. As the value of $\binom{k}{2}$ can be computed in polynomial time, this is a polynomial-time reduction.

### b

We will now show how a polynomial-time algorithm for DCT would allow us to construct a polynomial-time algorithm for the evaluation version, CT. The input is $(G, k)$ and the goal is to find $t^\star$, the maximum number of tunnels that can be controlled using $k$ blockers. The algorithm is simply:

- for $t = 1$ to $\binom{k}{2}$:
  - run the decider for DCT on $(G, k, t)$
  - if it accepts, continue
  - if it rejects, return $t - 1$

By construction, this algorithm will return the largest number $t^\star$ of tunnels which could be controlled by $k$ blockers. Assuming $k \leq n$,

$$\binom{k}{2} = \frac{k(k-1)}{2} \leq \frac{n(n-1)}{2} \in O(n^2)$$

means that the loop will have a polynomial number of iterations and thus a polynomial number of calls to the decider for DCT (note that even if $k > n$, the decider would reject at $t > \binom{n}{2}$ assuming $G$ is a simple graph).

### c

We now show that a polynomial-time decider for DCT can be used to find a concrete placement of blockers maximizing the number of controlled tunnels. Again, let the input be $(G, k)$. Below, $G' - e$ refers to the graph attained by removing $e$ from the edges of $G'$ (leaving the nodes unchanged).

First, use the algorithm from (b) to find $t^\star$. Then:

1. Set $G' = G$ and $S = \emptyset$.
2. for $e \in E$:
   a) run decider for DCT on $(G' - e, k, t^\star)$
   b) if it (still) accepts, let $G' \leftarrow G' - e$
   c) if it rejects, let $S \leftarrow S \cup e$ (where $e = \{u, v\}$)

3. return $S$

After running this algorithm, $G' = (V, E')$ will contain a minimal set of edges for which it holds that $k$ blockers could control $t^\star$ edges. As this set is minimal, there is an optimal solution where all of $E'$ is controlled and thus $|E'| = t^\star$. Additionally, the set of their endpoints of $E'$ is $S$ and must have cardinality $k$.

The algorithm uses a polynomial number of iterations and the input graphs given to DCT are of non-increasing size, so still clearly polynomial in the original input size.

Instead of finding the set of tunnels controlled in an optimal solution, the same approach could be applied to the vertices to iteratively find the set of rooms to place the blockers in. This would require at most $n$ calls to the DCT decider.

### d

Graphs of degree at most two can only be composed of simple paths and cycles. We consider only graphs consisting of either a path or a cycle. We are allowed to assume that the input graph is connected.

Consider an instance $(G, k, t)$ of DCT where $G$ is a simple path of length $n$. As both endpoints of a tunnel must be blocked to control the tunnel, the optimal placement of blockers on a path must clearly be to block consecutive rooms; the first two blockers control one tunnel and every additional blocker will control another tunnel. The decision problem for a path can then be solved trivially for a path in polynomial time since exactly $k - 1$ (for $k \leq n$) tunnels can be controlled: simply answer whether $k - 1 \geq t$.

Consider now an instance $(G, k, t)$ of DCT where $G$ forms a simple cycle. For $k < n$, the argument for paths applies directly and at most $k - 1$ tunnels can be controlled. If $k = n$, the entire cycle can of course be blocked and all $n = k$ tunnels can be controlled. Thus, in this case, simply answer whether $k - 1 \geq t$ if $k < n$ or whether $k \geq t$ if $k \geq n$[1]

As it is trivial using graph traversal to determine which of the two cases applies, we now have an algorithm which decides DCT for graphs of maximal degree 2 in polynomial time. If DCT is still NP-complete for maximum degree 2, then this would imply that $P = NP$. Assuming $P \neq NP$ implies that DCT for maximum-degree 2 is no longer NP-complete.

In extending the algorithm to general (not necessarily connected) graphs of maximum degree two, one non-trivial issue is how to optimally cover some subset of cycles as covering a cycle entirely is always a better use of blockers than only partially covering cycles or entirely covering paths. This problem can be solved in polynomial time using dynamic programming, but as we are allowed to assume that the input graph is connected, we omit the details of this here.

## Problem 2

Out of $n$ people, $2k$ have a virus and we need to find $k$ of them.

[1] Note that if $k \geq n$ and $t > k \geq n$, then the instance could be seen to be a no-instance already by the fact that $t$ is strictly greater than the number of tunnels.

## a

To identify and test $k$ infected, consider the naive approach of testing one person at a time:

- $I = \emptyset$
- for each person in arbitrary order:
  - test if this person is infected; if they are, add them to $I$
  - if $|I| = k$, return $I$

As there will be exactly $n - 2k$ non-infected persons in the population, this algorithm will at most test $n - 2k + k = n - k$ persons before having identified a set $I$ of exactly $k$ infected.

To establish a matching lower bound, suppose an algorithm could identify $k$ infected persons using at most $n - k - 1$ tests. An adversary algorithm could answer that the first $n - 2k$ tests were all negative and the last (at most) $k - 1$ tests performed were positive. Thus, the algorithm has found and tested at most $k - 1$ infected people and is therefore not correct.

## b

We now allow algorithms to—based on knowing $n$ and $k$ ahead of time—deduce the set of infected people without necessarily testing them. The naive algorithm can then be changed to:

1. $I = \emptyset$, $S = \emptyset$
2. for each person in arbitrary order:
   a) test if this person is infected;
      i. if they are, add them to $I$
      ii. if they are not, add them to $S$
   b) if $|I| = k$, return $I$
   c) if $|S| = n - 2k$, return any subset of $k$ people in $\overline{S}$

The difference is that now identifying all the $n - 2k$ non-infected persons will let us immediately conclude that the remaining are infected in which case no more tests are needed. At most, this approach will test $n - 2k - 1$ non-infected persons, $k - 1$ infected persons and then one additional person (infected or non-infected) before terminating. This is in total up to

$$n - 2k - 1 + k - 1 + 1 = n - k - 1$$

tests.

This immediately gives the adversary algorithm proving a matching lower bound. Suppose an algorithm uses at most $n - k - 2$ tests. An adversary could answer that the first $n - k - 1$ people tested were non-infected and the remaining at most $k - 1$ tested were infected. There will be at least $k + 1$ untested people remaining and at least one of them is not infected. The algorithm has only decidedly identified $k - 1$ infected, so it must include at least one of the $k + 1$ untested as its output—the adversary can simply answer that this person is (among the) last non-infected people.

## Problem 3

For the information-theoretic lower bound, we ignore the people with the $k$ highest identification numbers (these are assigned before the algorithm runs and not related to infection status).

We model all algorithms as binary decision trees in which each node corresponds to testing one person at a time (branching on whether or not the test is positive). We know from the slides that a $t$-ary tree with $n$ leaves has height $\lceil \log_t(n) \rceil$. The height of the decision tree will correspond to the number of tests performed in the worst case to reach a leaf corresponding to a result.

We thus need to count how many leaves the decision tree must have. The population to consider has size $n - k$ and—depending on the input and which are ruled out by having the highest IDs—any subset of size $k$ could be the correct answer. This leads to $\binom{n-k}{k}$ possible outputs or leaves. The straightforward lower bound achieved this way will therefore be:

$$\left\lceil \log_2 \binom{n-k}{k} \right\rceil = \left\lceil \log_2 \frac{(n-k)!}{k!(n-2k)!} \right\rceil$$