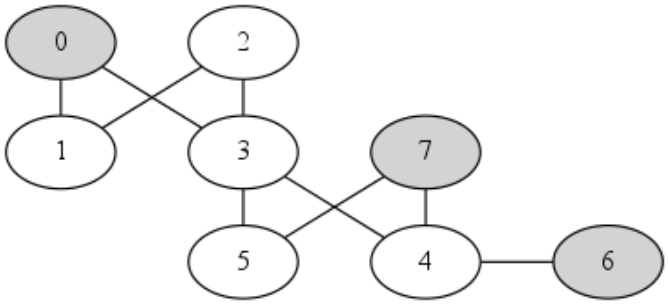
CPSC 320: NP-Completeness, or The Futility of Laying Pipe Solutions*

1 Steiner. . . Something-or-Others

UBC recently replaced its aging steam heating system with a new hot water system. A set of locations needs water delivered and there's another set of intermediate points through which we can deliver water. Some of these points can be connected—at varying costs—by laying new pipe, others cannot. You'd like to figure out the cheapest way to connect every delivery location to water.

We can model the problem as a graph, by representing locations and intermediate points as nodes. Possible connections are represented as weighted edges, where the weight of an edge is the cost of laying pipe between its two endpoints. For simplicity, moving forward, we'll assume that all costs are 1.

1. **Build intuition through examples.** Here's an instance, where shaded nodes are in S. Indicate a solution to this problem, that is, a way to connect up all of the shaded nodes. Draw your own small examples. What are trivial instances?



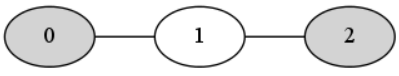
SOLUTION: We could use the edges $(0,3),(3,4),(4,7),(4,6)$ to connect up the shaded nodes. Note that the shaded nodes, plus the chosen edges, form a tree. However, this tree does not span *all* nodes in the graph, so it is **not** a spanning tree.

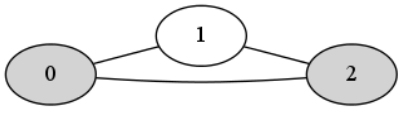
Some trivial examples include:

- The empty graph, whose solution is the empty set of edges.
- A graph with at most one shaded node, in which case the solution is also the empty set of edges.
- Less trivially, any graph where all vertices are shaded, in which case the solution is any spanning tree (which we can find efficiently with DFS or BFS).

We should probably assume that the graph is connected, since a disconnected graph where two or more connected components contain a shaded vertex has no solution.

And here are two small examples that don't fit in our trivial categories above yet remain simple. The first shows that we may end up with a spanning tree, while the second shows that we need to be ready **not** to (since its solution is the one edge connecting nodes 0 and 2).





2. Formalize the problem as an *optimization* problem, where the goal is to minimize or maximize something.

   Also, formalize the problem as a decision problem, where the answer is Yes or No. We'll call the decision problem the Steiner Tree (ST) problem. What is an instance of ST? A potential solution? A good solution?

   SOLUTION: An instance of the optimization problem is an undirected graph $G = (V, E)$ and a subset $S \subseteq V$ (the shaded nodes) to which we must deliver water. As usual we'll assume that $V = \{1, 2, \ldots, n\}$ and $|E| = m$. A (potential) solution to the instance is a subset $E' \subseteq E$ of the edges A solution E' is *optimal* if the edges of E' connect all vertices in S, perhaps via intermediate vertices in V, and in addition the size of E' is less than or equal to the size of any other solution for the instance.

   To make this a decision problem, the problem instance also includes a positive integer bound k, and we want to know whether or not there is a solution of size at most k. So in summary, an instance of the ST problem consists of $G = (V, E)$, $S \subseteq V$, and k, where $1 \leq k \leq n$. A solution E' is *good* if the edges of E' connect all vertices in S, perhaps via intermediate vertices in V, and in addition $|E'| \leq k$.

3. Think about similar problems by considering what an optimal solution to ST looks like. Is it a path? A cycle? A tree? A graph? Can we reduce ST to a similar problem we've seen before?

   SOLUTION: It's a tree. It is also, of course, a graph, since trees are graphs. It can be a path but need not necessarily be one, and it cannot be a cycle. That is, an optimal solution cannot be a cycle.

   There are problems that we've solved that are similar in various ways, mostly in the sense that they work with graphs (e.g., shortest path, or vertex cover, which is concerned with selecting particular nodes to cover the edges).

   However, the problem with what seems the greatest similarity is the minimum spanning tree problem. We can solve that in polynomial time (quite efficiently). Any spanning tree of the graph will connect **all** nodes in the graph and therefore is a solution to the ST instance, but it may not be a good solution. We could "trim off unshaded leaf nodes" repeatedly until there are none left, but there's still no guarantee that we'll get a good solution. So there's no obvious way to reduce ST to MST.

4. Prove that the ST (Steiner Tree) problem is in NP. Remember: it's in NP if it's "efficiently certifiable". The "certificate" is usually what we'd think of as a solution.

   SOLUTION: The ST problem is in NP if there is an efficient algorithm—called the certification algorithm—that takes as input an instance $I = ((V, E), S, k)$ of ST and a potential solution E', and correctly determines if E' is a good solution. The certification algorithm should run in polynomial time.

   In polynomial time, we can check that (1) the set of edges in E' has size at most k and (2) the edges connect all the shaded vertices. One way to handle (2) would be to DFS the subgraph of G formed by deleting all edges but the ones in the certificate, starting from any shaded node. During the DFS, we "check off" each shaded node as we reach it, including the starting point. If $|E'| \leq k$ and upon completion of the DFS, all shaded notes are checked off, then E' is a good solution and the output is Yes. Otherwise the output is No.

2 A Reduction from 3-SAT to ST

We've already shown that ST $\in$ NP. To show that it's NP-complete, we need to also show that it is NP-hard; that is, that it's at least as hard as every other problem in NP. We'll use 3-SAT to help us, since we already know that 3-SAT is at least as hard as every other problem in NP.

1. Which of these would show that ST is at least as hard as 3-SAT: reducing from ST to 3-SAT in polynomial time or reducing from 3-SAT to ST in polynomial time? (Hint: We **know** that any problem in NP can be reduced to 3-SAT in polynomial time. We want to **prove** that any problem in NP can be reduced to ST in polynomial time.)

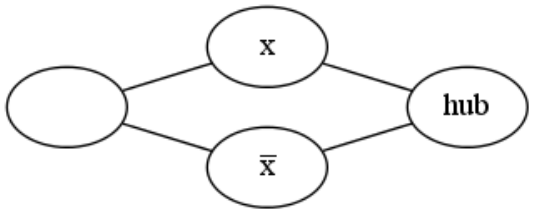
   SOLUTION: If we reduce from 3-SAT to ST in polynomial time, then we can "chain" together a reduction from any problem in NP to 3-SAT with the reduction from 3-SAT to ST and get a single "larger" reduction that still takes polynomial+polynomial=polynomial time from that problem in NP to ST. So, this is the direction we want.

   The other direction isn't interesting because it just tells us ST is yet another problem that can be reduced to 3-SAT in polynomial time.

   What does that tell us? We want to reduce **from** the problem we know is NP-complete **to** the problem we're unsure about.

   So, we reduce **from** 3-SAT **to** ST.

2. Here is a sketch of a "variable gadget" to help with our reduction. How can we "shade in" (put in S) some of these vertices and choose an appropriate k (maximum number of edges in the solution) to enforce 3-SAT's choice that x or $\overline{x}$ is true but not both?

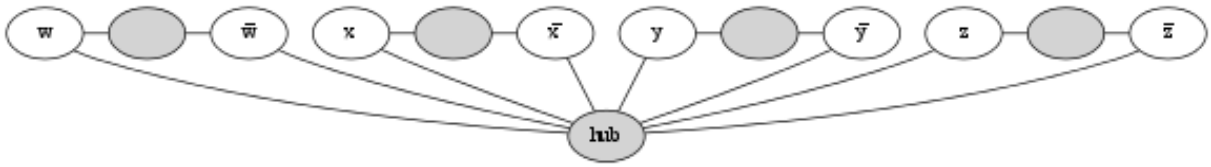


   SOLUTION: We want to use the rules of the ST "game" to force something to happen that we can intepret as "choosing x = true or choosing x = false but not both". If we shade either one of the x and $\overline{x}$ nodes, then they'll **have** to be included in our water distribution network, which seems an unlikely way to interpret the idea of "choosing" one of them. So, instead, let's shade the other two nodes.

   If we shade in the leftmost and rightmost nodes and choose $k = 2$, we know the Steiner tree may include at most three nodes (since any tree has one more node than edges). It **must** include the leftmost and rightmost nodes, which means it can include at most one of the x and $\overline{x}$ nodes. Furthermore, solutions exist containing either of these nodes. So, the gadget does what we'd hoped (where "picking a truth value" means "including the corresponding node in the Steiner Tree").

3. Draw a graph with four variable gadgets (w, x, y, and z), all sharing a single "hub" node. Make sure your layout still enforces choosing either true or false but not both for each of the four variables, yet allows all 16 possible combinations of their truth values.

   SOLUTION: Our layout isn't incredibly beautiful, but it gets the point across.

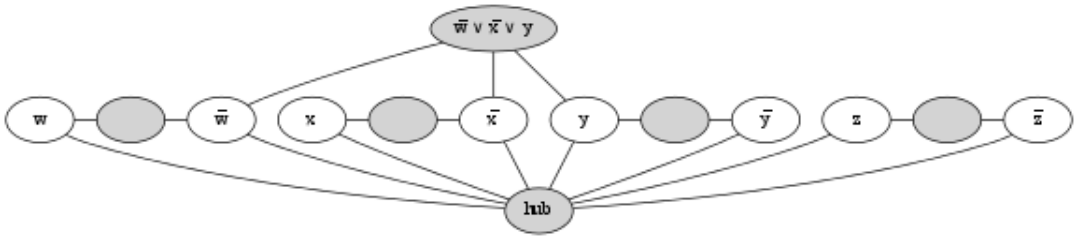


   Note that we can choose independently for each variable whether to go "left" to make it true or "right" to make it false.

   We'll make $k = 8$ so that there's just enough edges to enforce choosing one or the other direction (but not both) for each variable.

4. Now, find a way to add one or more nodes and edges to your graph and choose a k in order to represent the clause $(\overline{w} \vee \overline{x} \vee y)$ and enforce that: at least one of $\overline{w}$, $\overline{x}$, and y is true and also (still) each variable is either true or false but not both. (z isn't in this clause, which is fine. In most 3-SAT problems, not all variables are in all clauses.)

   SOLUTION:



   How many nodes do we need to connect? For each variable, there's the shaded node between its positive and negative literal and there's the literal we chose for its truth value. That's two nodes per variable. There's one node for the single clause. Finally, there's one node for the hub. We need one fewer edges than nodes.

   So, we use $k = 2 * 4 + 1 + 1 - 1 = 9$ or, more generally $k = 2 * n + c + 1 - 1 = 2 * n + c$, with n variables, c clauses, and one hub node, and subtracting one since i nodes need only $i - 1$ edges in a tree. This allows just the right number of nodes so that the solution must include exactly one of each of the negated/non-negated variable nodes. "Wiring up" the clause node requires that one of the clause's literals be true.

5. Give a complete reduction from 3-SAT to ST such that the answer to the ST instance you produce is `YES` if and only if the answer to the original 3-SAT instance is `YES`.

   SOLUTION: Our algorithm to transform an instance I or 3-SAT to an instance $I' = (G, S, k)$ of ST is as follows.

   Nodes: We first create a single hub node. Then, for each variable x_i of instance I, we create a node labeled x_i and one labeled $\bar{x}_i$, and one additional "pin" node p_i (the unlabeled node). Then for each clause $c_l = (l_1 \vee l_2 \vee l_3)$, we create a node labeled c. (Recall that each l_j is either some variable x_i or its negation $\bar{x}_i$.) So, with n variables and c clauses, that's one hub, n pins, n variable nodes, n negated variable nodes, and c clause nodes, for a total of $3n + c + 1$ nodes.

   Edges: We put edges between the hub and each of x_i and $\bar{x}_i$, as well as edges between p_i and each of x_i and $\bar{x}_i$. Also, we connect each clause node c_l to each of its literals l_1, l_2, and l_3. The total number of edges is $4n + 3c$. These nodes and edges comprise our graph G.

   We let the set of "shaded" node S be the hub, the n pin nodes and the c clause nodes. So, there are $n + c + 1$ shaded nodes in total.

   Finally, we let $k = 2n + c$.

6. Analyze the runtime of your reduction (to show that it takes polynomial time).

   SOLUTION:

   We tried to be careful accounting for the number of times we were performing operations above. In terms of $n + c$, each step clearly does take polynomial time.

3 Reduction Correctness

Now, we'll prove that our reduction is correct. To do this, we need to show that instance I is a Yes-instance of 3-SAT if and only if instance $I' = (G, S, k)$ is a Yes-instance of ST. So, our proof should proceed in two directions, one for the "if" and one for the "only if".

3.1 If I is a Yes-instance of 3-SAT then I' is a Yes-instance of ST

1. We usually want to go further than the assumption that I is a Yes-instance, and describe what is a good solution (i.e., working certificate) for I.

   Consider what a solution for 3-SAT looks like and use it to finish the statement: "Since I is a Yes-instance of 3-SAT, there must be..."

   SOLUTION: "...some assignment of truth values to the variables that makes at least one literal true in every clause."

2. To prove that I' is a Yes-instance of ST, we similarly try to show the existence of a good solution (working certificate) to that instance, and conclude that therefore the answer to the instance is `YES`. What does a good solution for ST look like?

   SOLUTION: The "solution" to an ST problem is a set of at most k edges that connect all the shaded nodes.

3. Given a truth assignment that satisfies I, how would you select the edges of the good solution of I'?

   SOLUTION: We designed our reduction so that one "variable gadget" represents the truth (or falsehood) of each variable. Thus, in each "variable gadget", we'll use two edges to connect the "pin"

to the hub via the variable node x_i if x_i is set to true, or via $\bar{x}_i$ if x_i is set to false. This is possible using $2n$ edges in total.

Then, pick the first literal that is true in each clause c_j, and choose the edge connecting c_j to that literal. Note that this literal node must already be connected to the variable's hub and pin, since we ensured we always connect in the node corresponding to the variable's truth value. We use c additional edges here.

4. Finish the proof to show that the ST certificate actually works, i.e., is a solution to the instance.

   SOLUTION: Above we built a candidate solution using exactly $2n + c$ edges. Does it actually connect all shaded nodes? Every variable's pin node is connected to the hub, as discussed above. Further, every clause is connected to a true literal, which must in turn be connected to the hub given the way we chose the variable gadgets' edges. Thus, our ST "solution" is a valid certificate showing the answer to the ST instance is `YES`.

3.2 If I' is a Yes-instance of ST then I is a Yes-instance of 3-SAT

1. First, an apparently unrelated side-track: **How many edges must there be in a tree with n nodes?** So... **if you use at most k edges to form a connected graph, how many nodes can you connect?**

   SOLUTION: A tree with n nodes has $n - 1$ edges. One way to figure that out is that each edge connects exactly one node to its parent, and every node but the root has a parent.

   Similarly, with k edges to form a connected graph, we can include at most $k + 1$ nodes. (One way to prove that would be by induction on the number of edges.)

2. Now complete the proof. *Hints:* Many pieces of this proof will be very similar to the previous one, but you'll also need to show that any solution (working certificate) for the ST instance has its edges where you intended them to be when you designed your reduction. It may help to think about the number of nodes you can possibly connect, given a maximum of k edges.

   SOLUTION: Since I' is a Yes-instance of ST, there is a set of at most $k = 2n+c$ edges that connects up all shaded nodes of G. By part 1 above, this set of edges can connect exactly $2n + c + 1$ nodes into a tree.

   There are $n + c + 1$ shaded nodes that **must** be connected; so, n of the unshaded nodes can also be connected. For every variable gadget, at least one of its "variable" nodes (positive or negated) **must** be connected (because there's no other way "out" of the variable's pin, which is why we called it a "pin"). Since this consumes all remaining available nodes, we must choose exactly one of each positive/negated variable pair. Call these connected variable nodes the "chosen" nodes. Furthermore, each clause c_l must be connected to at least one of its literals, and moreover, should be connected to a "chosen" node.

   (Note that it can be possible to connect all $2n+c+1$ of the shaded and chosen nodes with $k = 2n+c$ edges, where some clause node has **more than one** incident edge; this does not pose a problem in our reasoning.)

   Now consider the truth assignment corresponding to the "chosen" nodes, that is, set x_i to be true if x_i is a chosen node, and set x_i to be false if $\bar{x}_i$ is a chosen node. Since every clause node has an edge to at least one chosen node, all clauses in our 3SAT instance are satisfied by this truth assignment, and so the instance's answer is `Yes`.

   You've now shown that ST is in NP and is NP-hard (because 3-SAT—which is known to be NP-hard—is polynomial-time-reducible to ST). Therefore, ST is NP-complete!