# CPSC 320 Sample Solution: The Futility of Laying Pipe, Part 2

November 23, 2016

We've already shown that SP $\in$ NP. To show 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 SP is at least as hard as 3-SAT: reducing from SP to 3-SAT in polynomial time or reducing from 3-SAT to SP in polynomial time? (Hint: Checking whether a list of $n$ numbers is in sorted order is a simple problem that can be solved in polynomial time, which we'll call SORTED. There is a (silly, trivial) polynomial-time reduction one direction or the other between SORTED and 3-SAT. Whichever direction works easily for that is the **wrong** direction, since SORTED is only NP-complete if P=NP.)

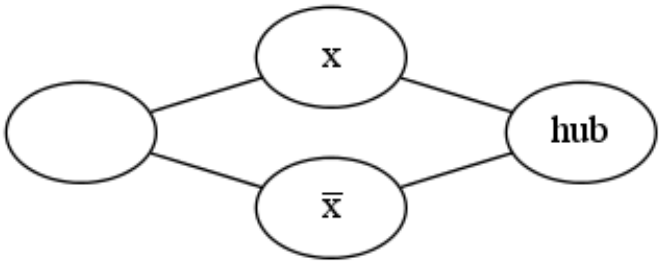
   **SOLUTION:** With respect to the hint, we can trivially reduce from SORTED to 3-SAT. Here goes: first, solve the SORTED problem (by scanning the list to check if neighbours are out-of-order in linear time). If it is sorted, generate the trivial 3-SAT instance with no clauses, for which the answer is defined to be `YES`. Otherwise, generate the trivial 3-SAT instance with a single empty clause, for which the answer is defined to be `NO`. (If you don't like trivial instances, use $(x_1 \vee x_1 \vee x_1)$ for the `YES` intsance and $(x_1 \vee x_1 \vee x_1) \wedge (\overline{x_1} \vee \overline{x_1} \vee \overline{x_1})$ for the `NO` instance.) Now, the answer to the 3-SAT instance is also the answer to the SORTED instance. The reduction takes polynomial time because I can **solve** SORTED in polynomial time.

   That reduction certainly doesn't prove that SORTED is NP-complete! (Technically: unless P = NP, in which case SORTED is trivially NP-complete anyway.)

   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** SP.

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?



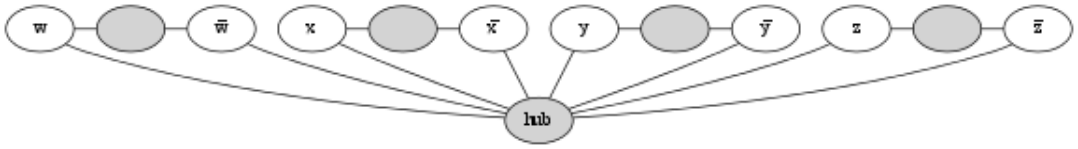

   **SOLUTION:** We want to use the rules of the SP "game" to force something to happen that we can intepret as "choosing $x$ = true or choosing $x$ = false but not both". If we shade either one of the $x$ and $\overline{x}$ nodes, then they'll **have** to be included in our water distribution network,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

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
For license purposes, the author is the University of British Columbia.

If we shade in the leftmost and rightmost nodes and choose $k = 2$, we know the Steiner Tree may include at most 3 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 ($x_1$, $x_2$, $x_3$, and $x_4$), all sharing a single "hub" node. (Be 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 used variables $w$, $x$, $y$, and $z$ rather than $x_1$ through $x_4$.

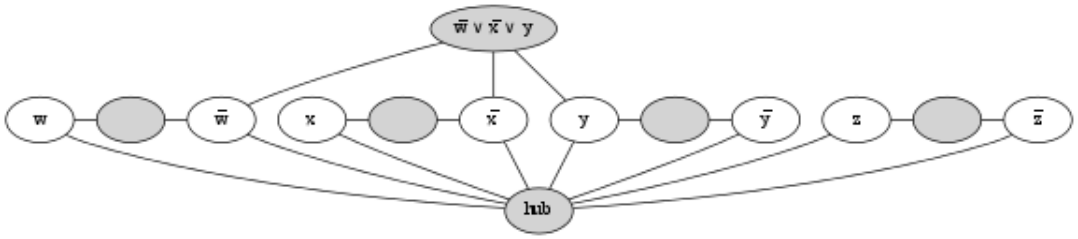



   Note that we can choose independently for each variable whether to go "left" to make it true or "right" to make it false.

   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 $(\overline{x_1} \vee \overline{x_2} \vee x_3)$ and enforce that: at least one of $\overline{x_1}$, $\overline{x_2}$, and $x_3$ is true and also (still) each variable is either true or false but not both. ($x_4$ isn't in this clause, which is fine. In most 3-SAT problems, not all variables are in all clauses.)

   **SOLUTION:** Again, we used $(\overline{w} \vee \overline{x} \vee y)$ rather than $(\overline{x_1} \vee \overline{x_2} \vee x_3)$:




   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 SP such that the answer to the SP instance you produce is `YES` if and only if the answer to the original 3-SAT instance is `YES`.

   **SOLUTION:** Based on our 3-SAT instance, we lay out a shaded hub node and then: for each variable, lay out a shaded "pin" (the unlabeled node), a positive node, and a negated node, connected in the "diamond" pattern pictured above; for each clause, lay out a shaded node and connect it to its three literals' nodes. So, with $n$ variables and $c$ clauses, that's one hub, $n$ pins, $n$ positive variable nodes, $n$ negated variable nodes, and $c$ clause nodes. Finally, let $k = 2n + c$ as described above.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
For license purposes, the author is the University of British Columbia.

Whatever the answer to our SP instance, we report it as the answer to our 3-SAT instance.

6. Analyse 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.

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.
For license purposes, the author is the University of British Columbia.