Bram Schuijff
CSCI 387, §S01/02
Homework 7
April 8th, 2025 by 10:30am

1. In order to formulate this problem as a language, we can phrase it in the following way:

$$L = \{\langle G, K\rangle \mid \text{For } G = (V, E) \text{ a graph with } K \subseteq V,$$
$$\text{there exists a set of vertices } M \subseteq V \backslash K \text{ such that for each } k \in K,$$
$$k \text{ is adjacent to exactly } n(k) \text{ vertices in } M \text{ where } n(k) \text{ is the label of } k\}$$

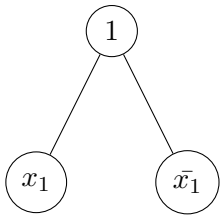
To demonstrate that L is $\mathbb{NP}$-complete, we must first show that L is in $\mathbb{NP}$. To do so, we can construct a polynomial-time verifier D for L. When given an input $\langle G, K, w\rangle$, D can check first that $K, M \subseteq V$ and furthermore that $K \cap M = \varnothing$. Traversing the vertex set of M and K is certainly polynomial time and then comparing them against each other is $O(|V|^2)$ time. Therefore, this step can be done in polynomial time. To verify that an assignment is valid, we can simply iterate through each $k \in K$, check its adjacency list, and then see if there are exactly $n(k)$ vertices adjacent to k that are in M. If there are not, reject. If we have not rejected by the time we have iterated through each k in K, accept. This can be done in $O(|V|^2)$ time because that is the most amount of time it can take to check all of the adjacent vertices of all vertices in G. Therefore, D is polynomial time in the size of $\langle G, K\rangle$.

To check that D verifies L, we see first that any witness that D will accept necessarily has M and K being disjoint. Any valid witness will have this property because if there exists an $e \in M \cap K$, then a mine would have been revealed and the game would be over. Then, we simply check that every revealed node has exactly as many mines adjacent to it as its value says it does. If it does not, then the graph is necessarily inconsistent because there exists a revealed node whose value is more or less than the number of mines it is adjacent to and we reject. If all revealed nodes have precisely the number of mines adjacent to them as their s value indicates, then the graph is consistent by definition. Therefore, we accept. Thus, D decides L and therefore L is in $\mathbb{NP}$.

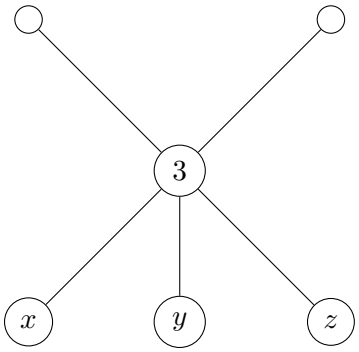
To demonstrate that L is $\mathbb{NP}$-complete, we can perform a reduction from 3SAT. Given a 3-cnf boolean expression X, we are attempting to produce a graph and revealed vertex set G, K such that finding whether or not G, K is in L would prove the satisfiability of X.

We can start by recognizing what our truth value is supposed to be. For my construction, I am going to assign each literal to exactly one unrevealed vertex in G. Then, a variable being marked as True is associated with having a mine marker placed on that variable's associated vertex.

We can start by constructing a variable gadget. We want it to be that for a variable x, either x or $\bar{x}$ is True. Therefore, only x or $\bar{x}$ can have a mine placed on top of it. Therefore, we create a setup whereby x and $\bar{x}$ are adjacent to each other and they are both adjcaent to a revealed vertex whose value is 1. Thus, if there is a satisfying assignment, it cannot be that both x and $\bar{x}$'s vertices have mines. A visual demonstration follows:



Then, to construct a clause gadget, we want it to be that for each clause, at least one of the variable's associated vertices has a mine. To do so, we can construct a gadget whereby if 3 variables are in the same clause, they are all connected to a vertex labelled 3. That vertex labeled 3 is additionally connected to 2 other unrevealed vertices not associated with any variable. For instance, in the clause $x \vee y \vee z$, we would produce the following component:



This guarantees that at least one of x, y, z contains a mine because 3 has already been revealed and therefore cannot contain a mine itself. In order for this graph to be consistent, then because there are only 2 other unrevealed nodes connected to 3, one of x, y, z has to contain a mine. If a clause only has two variables (for instance $x \vee x \vee y$) then we simply get rid of the z and connect the vertex labelled 3 to x and y. Meanwhile, if a clause consists of only one variable like with $x \vee x \vee x$, then we simply connect the 3 to x and the two other unlabelled, unrevealed vertices.

To show that this construction is actually a reduction, we find that the construction is inherently consistent—either x or $\bar{x}$ contains a mine—and that each clause must contain at least one mine. Therefore, if we let our mines represent True, we have that demonstrating G, K is consistent is equivalent to showing that X has a satisfying assignment.

In this construction, we find that because each variable x has a vertex in K associated with its variable gadget and each clause has 3 vertices associated with its clause gadget, we get $|K| = n \cdot m$ if m is the number of clauses and n is the number of variables. Then, $|V| = 2n + |K|$. These are both polynomial in the length of our boolean expression. Because $|E| \in O(|V|^2)$, we then have that we can also construct all of our edges in polynomial time. Therefore, this reduction is polynomial time.

Therefore, because $L \in \mathbb{NP}$ and there exists a polynomial-time reduction from 3SAT to L and 3SAT is $\mathbb{NP}$-complete, we have that L is $\mathbb{NP}$-complete.

2. Because PSPACE = NPSPACE by Savitch's Theorem, if we can demonstrate that there exists a nondeterministic polynomial space Turing machine which decides GOMOKU, we are done. Let D be the following nondeterministic Turing machine:

   1. On input $\langle C \rangle$:

   2. If C contains 5 contiguous red pieces, accept.

3. If C contains 5 contiguous blue pieces, reject.
4. If every square but 1 on C is covered:
5. If it is the blue player's turn, reject.
6. If there are not 4 red pieces adjacent to the empty square, reject.
7. If it is not the red player's turn, jump to step 9.
8. Nondeterministically select an unmarked space and place a red token there.
9. For each available space $c_{i,j}$ on the board:
10. Place a blue piece at $c_{i,j}$ and call this configuration $C[i, j]$.
11. Spawn a child process $D(\langle C[i, j]\rangle)$.
12. If $D(\langle C[i, j]\rangle)$ rejects, reject. Otherwise, continue.
13. If we have not yet rejected, accept.

We know that D decides GOMOKU because given a configuration C, we can nondeterministically select the best move for red and then attempt every single response that blue could produce. If blue has a move that allows them to win or force a tie as we check for in steps 3 to 6, we reject because red cannot force a win from the configuration C. However, if blue attempts every single response and still cannot win or force a tie, then we accept because red can force a win no matter what blue attempts. We have the base case in step 2 where if a configuration has 5 contiguous red pieces, then red wins from that configuration.

We know that $D \in$ PSPACE because each child process occupies space in $O(n^2)$ where n is the length of one side of the board. Furthermore, we have a recursive depth in $O(n^2)$ because we fork at most to the depth that it takes to fill the board; the number of child processes spawned at any point is irrelevant. Therefore, D occupies space in $O(n^4)$ which is polynomial. Therefore, GOMOKU $\in$ PSPACE.

3. To demonstrate that PSPACE is closed under the star operation, let L be a language in PSPACE. Now, we want to show that $L^* \in$ PSPACE. Because $L \in$ PSPACE, there exists a polynomial space decider D for L. We can now construct a polynomial space decider D^* for L^* as follows:

1. On input $\langle w\rangle$:
2. If $w = \epsilon$, accept.
3. Run $D(\langle w\rangle)$. If it accepts, accept.
4. Let $p = 0$.
5. Loop:
6. Nondeterministically set d such that $p < d \leqslant |w|$.
7. Run $D(\langle w[p, d]\rangle)$ with Pythonic string slicing. If it rejects, reject.
8. Set $p = d$.
9. If $d = |w|$, accept.

This is a decider for L^* because for any string $w \in L^*$, we have that $w = w_1w_2w_3...w_n$ where $w_i \in L$ for all i. Therefore, if we can iterate through and find the division points

between the slices that make up w, we are done because we can just run the decider for L on each slice and D^* will accept w. Meanwhile, if no such division exists, then there is exists at least one substring w_j in w such that $w_j \notin L$, at which point D will reject w_j and therefore D^* will reject w. Thus, D^* decides L^*.

To show that D^* is in PSPACE, we will first show that D^* is in NPSPACE. Steps 2 and 3 can clearly be done in polynomial space because L is in PSPACE. Meanwhile, steps 4, 6 and 8 are space in proportion to $\langle w \rangle$. Then, step 7 is polynomial space because L is in PSPACE. Finally, our end condition occupies constant space. Therefore, D^* occupies polynomial space and $L^* \in$ NPSPACE. By Savitch's theorem, we also have that $L^* \in$ PSPACE and therefore PSPACE is closed under the star operation.