NP Completeness: an attempt at an unsolved problem

Connor

This week, the plan was to quickly find a problem that hadn't been proven to be NP complete and demonstrate that it was. Unfortunately, our efforts were stifled when we realized how hard it was to find problems that seemed to have the potential for NP-Completeness, but hadn't yet been proven to be. That's not to say this was all a big waste of time. In understanding the ideas behind the proofs we looked at, we've gained insight into the techniques in proving NP-Completeness, and an idea for the types of problems that are potentially NP Complete. What follows will be a brief overview of a handful of the more interesting problems we looked into.

Zelda! The decision problem of whether or not a goal location in a dungeon containing blocks is reachable from a starting location in the original Legend of Zelda is NP-Hard. In the original game, there exist blocks that can be pushed at most 1 time. This proof is essentially the same is the proof that "pushpush" games are np complete. In the proof for "pushpush", variables are stored by way of branching paths, 2 for each variable. The player "assigns" the variables by taking one of the 2 paths. These variables connect to clauses to form circuits. Further, a variety of other gadgets are used to form wires and other essential pieces. Beyond the variable and forking gadgets required to lock the player to a variable choice, there are one-way gadgets, which allow travel in one direction, clause gadgets, which only allow passage if one of the literals it contains is true, lock & key gadgets, which cannot be traversed until a key block is pushed, and crossover gadgets, which let 2 "wires" to physically cross without data from one contaminating the other.

The combinations of these gadgets allows for complex systems of wires, variables and clauses. If a algorithm was able to tell you whether or not the end goal was reachable from the starting point in a dungeon, it would be inadvertently telling you whether the 3SAT problem encoded into the dungeon was satisfiable. Because 3SAT is NP-Hard, and 3SAT reduces to Zelda, Zelda is NP-Hard!

All previous NP-Hard proofs we had read reduced 3SAT to various other problems to prove NP-Completeness. We began to discover there are other problems that are used in reduction proofs, albeit less frequently. One such problem is the "Vertex Covering" problem. Consider an arbitrary graph G, the vertex covering problem asks if it is possible to select a set S of at most k verticies from G such that every edge in G is connected to an element of S. 3SAT can be reduced to vertex covering to show that vertex covering is NP-Complete.

Take any 3SAT formula F. For each variable in an arbitrary 3SAT problem, create a pair of verticies corresponding to the variable and its negation. Note that only one of the verticies in this gadget needs to be covered. For each clause in F, create 3 connected verticies in a triangle corresponding to the literals in the clause. Note that at minimum two of these verticies have to be covered. Then, connect each literal in each clause to its corresponding variable or negation. Set $k = n + 2m$ where n is the number of variable gadgets, and m is the number of clause gadgets. Thus, we can let S contain all the verticies in variable gadgets corresponding to true literals, and all the verticies in clause gadgets corresponding to false literals. This demonstrates that if F is satisfiable, then G is coverable with at most k verticies. For the other direction, let all the variables in F corresponding to verticies in S be true. Because $k = n + 2m$, every variable is assigned either true or false, and every clause has at least one true literal. Thus, we've proven the reduction

Another game we considered was MasterMind. For the uninitiated, Mastermind is a two person game, in which one player selects 4 colored pegs and arranges them in an order of their choosing. The second player then makes guesses at the first player's choice of color and arrangement, receiving feedback after each guess. If the second player receives a black peg as feedback, it signifies that the guess contained a peg of the same color and position as one in the first player's combination. A white peg signifies that the guess contained a peg of the correct color, but not the correct location.

The proof we read concerns a variant called the "Mastermind Satisfiability Problem". In this variant, the second player makes all their guesses simultaneously, then after reviewing their feedback, attempts to find a unique combination of colored pegs that satisfies all of the first players feedback. Phrased as a decision problem, is there a combination of colored pegs S such that after the second player has made all their guesses and received feedback of the form (b, w) where b and w are the number of white and black pegs they receive for the guess, S satisfies all feedback received? Note that in this version of the game, the number of pegs, l, in a guess is not constrained to 4, and the number of colors, k, is not constrained to 6.

Vertex covering with at most n verticies can be reduced to MSP. Let $G = (V, E)$ be a graph to map to a Mastermind Satisfiability Problem. For this MSP, let $k = \#V + \#E + 2$ and $l = 3 + 2\#V + \#E$, where $\#V$ and $\#E$ are the number of verticies and edges in G respectively. The colors in this game are $K = \{v_1, v_2 ... v_{\#V}, e_1, e_2 ..., e_{\#E}, Y, N\}$.

Player 2's guesses in this game of MSP can be described as follows.

1. Player 2's first guess will be all pegs of color N, or in other words $(N, N, ...N)$, with feedback $(0, 0)$. This prevents the color N from appearing anywhere in the solution permutation.

2. Player 2's second guess will be $(Y, Y, Y, ...N)$. The feedback for this guess will be $(3, 0)$. This combined with the first step forces the first 3 elements to be Y.

3. Thirdly,for each edge in G, create a guess. For ith edge that connects verticies a, b, let the guess for that edge be $(e_i, a, b, N, N, ..., N)$. This guess should recieve feedback $(0, 2)$. Because the first 3 positions are Y, Y, Y, we know that none of the elements in $e_i, a, b, N, N, ..., N)$ have the correct positions.

4. For Player 2's last guess, put down $(Y, Y, Y, v_1, v_2, v_3 ... v_{\#V}, N, N ... N)$. This will receive feedback of $(3, n)$, because we know the three Y are correct. This also tells

us that the n correct colors in V are accounted for, but their positions are incorrect.

First, show that if MSP has a solution, then G is coverable with n verticies. We know that n verticies $W = \{w_1, w_2 ... w_n\}$ from V are in the solution to MSP. Each edge has a corresponding guess from 3 above, either a or b is in W in order for 3 give the feedback that it did. Therefore we can say that W covers G using n verticies.

For the other direction,
If W can cover G with n verticies, then the solution to the MSP that G encodes is $(Y, Y, Y, Y, Y...Y, w_1, w_2, w_3...w_n, Y, Y, Y, Y..., Y, e_{i1}, e_{i2}, ...e_{it}, Y, Y, Y)$. Where the e_{ij} are edges not connected by pairs of verticies in W. In order for the $(0, 2)$ condition above to be valid, the MSP must have a solution.

References:

http://arxiv.org/pdf/1203.1895v2.pdf http://arxiv.org/pdf/cs/0007021v2.pdf http://arxiv.org/pdf/cs/0512049.pdf http://en.wikipedia.org/wiki/Vertex_cover http://www.csc.kth.se/utbildning/kth/kurser/DD1352/adk12/alvie/vertexCoverNPCompleteness.p
http://www.csc.kth.se/utbildning/kth/kurser/DD1352/adk12/alvie/vertexCoverNPCompleteness
pdf