Mästarprov 2 2017

Max Bergmark
931211-3674 - maxbergm@kth.se

April 2017

1 Super connectors

From wikipedia:

> In the k-clique problem, the input is an undirected graph and a number k. The output is a clique with k vertices, if one exists, or a special value indicating that there is no k-clique otherwise. In some variations of this problem, the output should list all cliques of size k.

Thus, our input is an undirected graph G and a number k. We note that in any clique of size k, all vertices in the clique has at least $k-1$ edges. Thus, they are super connectors if $C \leq k-1$. Also note that if $C \geq k-1$, then any group of super connectors of size k is also a clique.

Thus, in the proposed reduction, the graph G stays the same, k stays the same, and $C = k-1$. This reduction is clearly polynomial in time, and if the Super connector problem is solvable in polynomial time, then the Clique problem is also solvable in polynomial time. Thus, the Super connector problem is at least NP-hard.

To prove that it is NP-complete, we must also show that it is in NP. That means that if we have a proposed solution, we must be able to check whether it is a true solution in polynomial time.

To check the solution, loop through all vertices in the solution, and check whether they are super connectors. Also check that they are connected to all other vertices in the solution set. This takes $\mathcal{O}(n^2)$ time, and thus the problem is in NP.

Since it is both NP-hard and in NP, it is NP-complete.

2 Different types of matchings

2.1 a)

To reduce Tripartite matching to Fourpartite matching, we must be able to transform input from Tripartite matching to input for Fourpartite matching in polynomial time.

The easiest way to make such a reduction is to create the set of cats, and make every cat get along with every man, woman and dog. That is, for every feasible three-group in the Tripartite input, insert every cat into each group. This will yield n times as many feasible four-groups for the input of Fourpartite matching.

The reduction must be doable in polynomial time. It is easy to see that the sets of men, women and dogs are reducible in linear time (they are just copied to Fourpartite). The maximum number of four-groups possible is n^4, which is also clearly polynomial. Thus, the maximum input size to Fourpartite is polynomial in size, and thus the reduction exists.

Note that since every cat gets along with every three-group in Tripartite matching, then a solution of Fourpartite matching will also be a solution of Tripartite matching, if all cats are removed from the solution.

2.2 b)

We know that Tripartite matching is NP-complete. Thus, no algorithm exists which can solve it in polynomial time (if we assume that $P \neq NP$). A reduction from Tripartite matching to Bipartite matching exists if input from Tripartite matching can be transformed to input for Bipartite matching in polynomial time. This input can then be solved by Bipartite matching, and transformed back to become output for Tripartite matching.

If such a reduction existed, the fastest time it would take to solve Tripartite matching would be $T_R + T_{BM} + T_{TR} \in \mathcal{O}(p(n))$, where T_R is the time for reduction, T_{BM} is the time to solve Bipartite matching, and T_{TR} is the time to transform the output back. All of these times are polynomial in complexity, which means that their sum is also polynomial.

Thus, if a reduction exists, then Tripartite matching (a problem known to be NP-complete) would have a polynomial time solution, which would violate $N \neq$

NP.

3 The Plus Minus Game

Deciding if A has a winning strategy is equivalent to checking if A can win, given the cards in the game. A can win if B is not able to create the sum 0 regardless of A's moves.

To see if this problem is NP-hard, we will reduce it from a known NP-complete problem. The problem chosen is the Partition problem.

The Partition problem takes as input a multiset S, and the goal is to figure out if there is a partition of S into S_1 and S_2 such that $sum(S_1) = sum(S_2)$. By simple algebra, we see that this implies that $sum(S_1) - sum(S_2) = 0$.

Thus, the reduction proposed is to take Partition problem input S and give it as input to B. A is given all zeros as input. Thus, A is only able to win the game if B can not find a partition of its card into a plus set and a minus set, which results in $sum(B_+) - sum(B_-) = 0$.

If there existed a polynomial time algorithm which could find out if A could win the game, then since there exists a reduction from Partition problem to the Plus Minus game, Partition problem would be solvable in polynomial time. It is trivial to see that the proposed reduction takes $\mathcal{O}(n)$.

Thus, the Plus Minus game must be NP-hard, and is at least as hard as Partition problem to solve.

4 The intricate amusement park

The first step is to find the length of the maximum path. The maximum value possible is $|V|$, and a binary search could give the correct answer in $log(|V|)$. Call the maximal path length L_{max}.

Once the longest path is found, we must find all edges which belong to the longest path. There could be several paths with the same longest length, but we only need to find one of them.

If we remove one edge from the graph, one of three things can happen. Either that edge is not part of any path of maximum length, in which case it will not affect any interesting properties of the graph. Removing it will result in a graph with $|E| - 1$ edges which still has the same maximal path length L_{max}. Call this case 1.

Or it could be part of one longest path, but after removing it there still exists another path of the same maximum length. In this case, removing it also yields a graph with $|E| - 1$ edges and the same maximal path length L_{max}. Call this case 2.

The last case is if the edge is part of the maximal path, and removing it decreases the maximal length of the path. In this case, removal yields a graph with $|E| - 1$ edges and maximal length $< L_{max}$. Call this case 3.

Thus, the algorithm iteratively removes edges from the graph, and after each removal check which one of the three cases was caused by removing the edge. If removing an edge $\{u, v\}$ still gives "Yes" when calling $PathFinder(G \backslash \{u, v\}, L_{max})$, then either case 1 or case 2 happened. Thus, the edge removed was not important, and the algorithm continues with that edge permanently removed.

If removing an edge $\{u, v\}$ gives "No" when calling $PathFinder(G \backslash \{u, v\}, L_{max})$, then that edge is part of the maximal path that we wish to find. Thus, we put the edge back, and save it to a list of edges present in the maximal path.

This algorithm iterates through all edges once, and once it is complete it has a list of all edges present in the maximal path. Thus, the problem is solved!

Time complexity is $\mathcal{O}((log|V| + |E|) * T(|V|, |E|))$, which is clearly polynomial in both $|V|$ and $|E|$.