CS 630 – Fall 2024 – Lab 5
Oct 9, 2024

1. P, NP, and NP-completeness

For each of the following statements, indicate whether it is true, false, or "unknown", where "unknown" indicates that scientists have not conclusively determined whether the statement is true or false. For example, the correct answer for the statement "$P = NP$" is "unknown". But the correct answer for "the best algorithm for the maximum flow problem in n-vertex graphs takes time at least 2^n" is "false".

Give a short justification for your answer (i.e. explain why the statement is true, or false, or not known to be either true or false).

1. If X is in NP, then there does not exist a polynomial-time algorithm for X.

   False. Polynomial problems are by definition in NP.

2. If X is NP-complete, then there is no polynomial-time algorithm for X.

   Unknown. We don't know whether there exist polynomial time algorithms for any of the NP-C problems.

3. If X is NP-complete, Y is in NP, and X reduces to Y (thus $X \leq_p Y$), then Y is NP-complete.

   True, we have seen this in class. In summary; since X is NP-C we know by definition that any problem in NP reduces to X. By transitivity this yields a reduction from any problem in NP to Y, hence satisfying the definition of NP-C.

4. Shortest Paths is in P.

   True. We know multiple polynomial algorithms for it.

5. Shortest Paths is in NP.

   True. a valid certificate would be the path itself.

6. Shortest Paths is NP-complete.

   Unknown. Because Shortest Paths is in P, it can only be NP-complete if P=NP.

7. Longest Paths is in NP.

   The longest path itself is a polynomial time certificate.

8. Longest Paths is NP-complete.

   True. We saw in class that Hamiltonian Path $\leq$ Longest Paths and Hamiltonian Path is NP-complete.

9. Longest Paths is in P.

   We don't know, only if P=NP

10. Every problem in P is in NP.

    True. The polynomial solution to each problem works with an empty certificate.

11. Every NP-complete problem is in NP.

    true by definition of NP-C.

12. Some NP-complete problems have polynomial time algorithms, but some others do not.

    False. While we do not know if any NP-complete problems have polynomial time algorithms, if any of them do, then all of them have polynomial time algorithms. It cannot be the case that some but not all NP-complete problems have polynomial time algorithms.

13. Suppose that X is polynomial and $X \leq_P Y$, then Y is a polynomial problem.

    False. Reducing a polynomial problem doesn't say anything about Y as a trivial reduction is to 1. solve X using their polynomial algorithm 2. call the solver to Y 0 times. a.k.a. ignore problem Y.

14. Suppose that X is exponential, and $X \leq_P Y$, then Y is at least exponential.

    True. If Y was less than exponential, then we would have a non-exponential algorithm for X through the reduction.

15. Suppose that X is in NP, and $X \leq_P Y$, then Y is NP-C.

    False. The NP-complete definition requires that all NP problems be reducible to Y and that Y also be in NP. Here, only X is reduced to Y and Y is not known to be in NP. For example, Y could be double exponential time complete.

16. Suppose that Y is polynomial, and $X \leq_P Y$, then X is polynomial.

    True. We have a polynomial algorithm for X through the reduction to Y.

2. Approximation Algorithm

You are thinking of starting a project and want to form the largest possible group from a pool of students. Each student is enrolled in multiple courses. Since this project require extensive collaboration, we want to ensure that each students are familiar with each other and have overlapping schedules, which can be seen by them sharing a course. Write an algorithm that takes the list of students and their enrollment as an input and gives a largest subset of students such that every student in this subset are familiar with each other.

1. Design an approximation algorithm for this problem.
2. Clearly state and prove the approximation ratio.

Solution:
First, construct a graph where each student is represented a vertex, and there is an edge between two students' vertices if are familiar with each other (i.e. they share a course). This graph can be constructed in time polynomial time depending on the numbers of students and their courses with the details depending on how those are provided. Any set of k students who are all familiar with each other will form a k-clique in the graph by the graph's definition. To get the largest subset of students who are all familiar with each other, we can find the largest clique in the graph and map back to the corresponding students. We did not discuss an approximation algorithm for Max Clique but we can reduce to Max Independent Set by flipping the presence of each edge. This reduction preserves the sets of vertices in solutions, and the sizes of solution sets are unchanged. This means that the size of the maximum independent set in the flipped graph is the same as the largest subset of students who are all familiar with each other, and the same mapping from vertices back to students from the first graph construction still applies. Now we can apply the greedy independent set algorithm from class to yield a $D+1$ approximation where D in the second graph maps back to the maximum number of students any student is unfamiliar with (because of the edge flipping).

3. Max Flow

At a certain point in the season, each team i in the American League has won a certain number w_i of games, and there remain $q_{i,j}$ games to be played between teams i and j ($q_{i,j} = q_{j,i}$). Assuming no ties or canceled games, develop an efficient, flow-based algorithm for deciding if the Red Sox can still win the league pennant, i.e. whether they can still win more games than any other team after all games have been played.

Solution:
Suppose Red Sox is team 0 in the League, and they have won w_0 games. Without loss of generality, we assume that the Red Sox will win all of their remaining games. (If there exists a Red Sox pennant win without winning all their games, then there also exists a Red Sox pennant win where they win all their games which we can construct by changing all their losses to win.) The key question here is whether there exists a combination of wins and losses by the other teams such that none of the other teams has more wins than the Red Sox. Or to put it another way, can each team get enough losses simultaneously so that all of them have fewer wins than the Red Sox? We can model these simultaneous losses as a maximum flow problem.

Let $W_i = w_i + \sum_{j \neq i} q_{i,j}$. W_i is the maximum number of wins that team i can win. For each team $i > 0$, calculate $L_i = W_i - W_0 + 1$. Each team i besides the Red Sox must lose at least L_i games for the Red Sox to win the pennant.

We create the max flow graph as follows.

1. Create the source node.
2. For each pair of teams i and j where $0 < i < j$ and $q_{i,j} > 0$, add a node $S_{i,j}$ representing the series of games between teams i and j. Add an edge from the source to $S_{i,j}$ with capacity $q_{i,j}$. The capacity represents games to play between these teams, and flow represents losses to be had by one team or the other.
3. For each team $i > 0$, add a node T_i representing the team. Each one unit of flow through this node represents the team losing one game. Connect an edge from each $S_{i,j}$ and $S_{j,i}$ to T_i with infinite capacity. These edges represent the team accumulating losses from each series of games with another team. Crucially, flow conservation blocks each pair of teams from claiming more losses against the other team than there are games in their series.
4. Add the sink node. Connect an edge from each node T_i to the sink with capacity L_i. If the edge from T_i is saturated, then team i lost enough to lose to the Red Sox.

Run your favorite maximum flow algorithm on this graph. If the maximum flow is $\sum_{i>0} L_i$, then all the other teams simultaneously lost enough for the Red Sox to win.

If the maximum flow algorithm does not return integral flows, then this does not directly translate into win/loss records for each series, but by the Integral Path Theorem, we know that an integral flow exists and an algorithm based on Ford-Fulkerson will return one..