CS 4102 Written HW6 - Graphs and NP

Eric Xie

December 2, 2016

1.
We need to prove that if every node in graph G has a degree of at least $n/2$, then the graph must be connected. Any node must connect to at least n/2 other nodes, creating a connected graph of at least n/2+1 nodes. Thus, there are n-(n/2+1) nodes not in that connected graph, forming a set which we can call G'. A node n in G' must connect to at least $n/2$ other nodes, but since G' is has less than $n/2$ nodes, n must connect to at least one node in the connected graph. This is true for all nodes in G', so all nodes in G' must eventually be connected to the connected graph. Since the connected graph and G' are complements of each other, every node in G must eventually be in a connected graph of size $|G|$

2.
We need to prove that the Bi-Partite Matching algorithm finds the optimal matching between nodes in the bi-partite graph. By way of contradiction, assume that the max flow algorithm has not found every edge in the optimal bi-partite matching. That means there exists some augmenting path through which no flow was sent. This results in 3 possible cases, involving some nodes u and v in set X and nodes u' and v' in set Y. Let u be connected to u' and v be connected to v' in the optimal solution:
Case 1: $\{u, u'\}$
In this case, a forward augmenting path from the source to v, v to v', and v' to the sink would be found via Ford-Fulkerson.
Case 2: $\{u, v'\}$
In this case, u flows forward to v', even though u should flow to u' and v to v' in the optimal solution. Ford-Fulkerson would have found an augmenting path from source to v, v to v', a backflow from v' to u, and u to u'. In this case the backflow from v' to u "negates" the suboptimal path from u to v'.
Case 3: $\{v, u'\}$
In this case, v flows forward to u', even though u should flow to u' and v to

v' in the optimal solution. Ford-Fulkerson would have found an augmenting path from source to u, u to u', a backflow from u' to v, and v to v'. In this case the backflow from u' to v "negates" the suboptimal path from v to u'.

Thus it has been shown that in all three cases in which an augmenting path exists after Ford-Fulkerson terminates, Ford-Fulkerson is able to find the augmenting path and increase the flow in the optimal edges.

3.
To solve this problem, set up a circulation problem. Let each flight be represented by two nodes.
- There is a source of planes with demand of -k that connects to the first node of every flight, representing that k planes must be used.
-The first node of each flight connects to the second node with a forward edge of at least one unit of flow, representing the condition that each flight must be serviced.
-The second node of each flight connects to a sink of demand k, representing that each plane eventually stops servicing other flights.

Furthermore, the second node of flight A must be connected to the first node of flight B if a plane has time to service flight B after servicing flight A.
- If flight B departs from the same airport that flight A arrives in, the second node of A connects to the first node of B if the departure time of B is at least m minutes after the arrival time of A.
-If flight B departs from a different airport than the one that flight A arrives at, the second node of A connects to the first node of B if the departure time of B is at least m+L minutes after the arrival time of A.

Finally, to account for the situation that less than k planes can be used to service every flight, there is an edge of capacity k between the source and the sink to allow any excess planes to be used up.

This graph forms a circulation problem, which can be reduced to a max flow problem by adding an infinite source connecting to the existing source with an edge of capacity k and an infinite sink draining the exisitng sink with an edge of capacity k. Both the previous source and sink are set to demand 0. If the resulting max flow problem results in k flow, then the flight problem is feasible.

4.
First loop through each node n in the graph. For each other node, m, that is

not adjacent, create a vertex contain the tuple (n,m). These represent all the pairs of locations that the two robots can be in simultaneously, where robot 1 is at node n and robot 2 is at node m. Since one robot moves on every step, we can show the transition between location tuple nodes by linking vertices such that (a,b) is linked to (a,c) if b is adjacent to c (robot 2 moves) and (a,b) is linked to (c,b) if a is adjacent to c (robot 1 moves). The resulting graph represents each of the valid states for the two robots to be in, and the legal moves that connect them.
Since we are given the two starting locations and destinations, this problem can be solved by running Djikstra's algorithm on the graph of valid states and steps in order to find the shortest set of legal moves that allow the robots to move to their destinations.

Given a graph of V vertices and r robots, there are V^r possible combinations of positions for the robots, which forms the input for Djisktra's algorithm. Since the runtime of Djikstras is $O(V^2)$, the overall runtime is $O((V^r)^2)$. If there are additional robots while V is held constant, the number of valid location states decreases, but the number of tuples increases as well, so the runtime stays the same.

5.

To show that HC reduces to HP:
Given a graph G, we can create a modified graph by taking an edge e = $\{u, v\}$ in G and connecting a node u' to u and a node v' to v. This is the same as designating nodes u' and v' as the start and end nodes of a possible HP. If an HP exists from u' to v', then an HP exists from u to v since u' and v' are connected to no other nodes. Then we can simply add edge e to the HP to convert it to an HC. Consequently, since the start and end nodes of the HP can potentially be any two nodes in G, we must run the HP algorithm on a modified graph with every possible edge as e. If any of the graphs return true for HP, then G must contain an HC. But if all of the graphs return false for HP, then G has no HC.

To show that HP reduces to HC:
Given a graph G, we can create a modified graph by creating a node v and connecting it to every node in the graph. This essentially designates v as the start and end node of a potential HC. If there exists an HC in the modified graph, then we can simply remove v and any edges connected to v to yield an HP.

6.

If the Lecture Planning problem can be solved in polynomial time by a nondeterministic turing machine, it can be verified in polynomial time by a de-

terministic turing machine. The verification algorithm must use brute force to check that each project can be fulfilled by at least one of the required lecturers. In addition, we must verify that each scheduled lecturer is available to present on the day that they are scheduled, and that the overall scheduling has no conflicts, aka multiple or no lecturers lecturing on the same week.

To do this, instantiate an array of size L, called schedule. Iterate through the projects, instantiating a boolean satisfied = false for each project. For each lecturer in a project, check to see if the lecture is in the schedule. If the lecturer is not scheduled, move to the next lecture. Else if the lecturer is scheduled, find its index, i. If schedule[i] is null or equal to the lecturer, set schedule[i] equal to the lecturer. If another lecturer is already in schedule[i], return false, as there is a scheduling conflict. If the lecturer is both scheduled and does not have any conflicts, set satisfied to done. After iterating through all the lecturers in a project, if done is true, continue to the next project. Else, return false as at least one project cannot be sastified by the provided schedule. If we successfullly loop to the end of the last project and sastified is true, then return true.

Runtime is R*l, where R is the total number of possible requirements for all projects, and l is the number of weeks of scheduled lectures.

7.

To reduce 3-SAT to LP, we can map each of the clauses in the 3-SAT equation to the projects in an LP problem. The literals in each of the clauses map to the 3 lecturers that can fulfill that particular project. For any literal x, x and $\neg x$ represent the subset of n lecturers that are available during a unique week. Since only one lecturer is scheduled per week, this allows the possibility that x and $\neg x$ are both in the 3-SAT equation, but cannot both be true. The 3 literals are joined with OR, representing that any of the 3 lectures required for a project can occur in order to sastify it. Meanwhile, all the of the clauses are joined with AND, equivalent to the condition that all projects must be fulfiled in order to return true for the LP problem. Thus any 3-SAT problem can be reduced to an equivalent LP problem in which the total number of clauses is p, the n literals are unique lecturers, and the number of unique literals (with x and $\neg x$ counting as one unique literal) is l. By solving the LP problem, the list l represents the list of literals that can be set to true to fulfill the 3-SAT equation.