Pablo Acuna
CSCI 4020
Anshelevich

Computer Algorithms Homework 7

29. Some of your friends have recently graduated and started a small company, which they are currently running out of their parents' garages in Santa Clara. They're in the process of porting all their software from an old system to a new, revved-up system; and they're facing the following problem.

They have a collection of n software applications, 1, 2, ..., n, running on their old system; and they'd like to port some of these to the new system. If they move application i to the new system, they expect a net (monetary) benefit of $b_i \geq 0$. The different software applications interact with one another; if applications i and j have extensive interaction, then the company will incur an expense if they move one of i or j to the new system but not both; let's denote this expense by $x_{ij} \geq 0$.

So, if the situation were really this simple, your friends would just port all n applications, achieving a total benefit of b. Unfortunately, there's a problem...

Due to small but fundamental incompatibilities between the two systems, there's no way to port application 1 to the new system; it will have to remain on the old system. Nevertheless, it might still pay off to port some of the other applications, accruing the associated benefit and incurring the expense of the interaction between applications on different systems.

So this is the question they pose to you: Which of the remaining applications, if any, should be moved? Give a polynomial-time algorithm to find a set S $\subseteq$ {2, 3, ..., n} for which the sum of the benefits minus the expenses of moving the applications in S to the new system is maximized.

Set-up: To do this we will make graph G have a vertex for every software application, v_i. For v_1 we will create an outgoing edge to every other application with capacity x_1, j. Now for the other applications, create a bidirection edge, (v_i, v_j), between each application with capacity v_i, j. Now all thats left is to create a sink node, t. For all applications other than 1, create an an incoming edge to t, (v_i, t), with capacity b_i.

Solution: Using the min-cut problem, we say that v_1 - t min-cut will give us our answer.

Proof: Let S be a cut such that $v_1 \in$ S and t $\notin$ S.

$$Capacity of S = \sum_{i,j}^{v_i \in S; v_j \notin S} x_{i,j} + \sum_{j}^{v_j \notin S} b_j + \sum_{v} b_v$$

This is the same as expenses minus benefit of moving applications which are not in S. Thus finding a min-cut is the same as finding the set of applications where expenses minus benefits is minimized.

Pablo Acuna
CSCI 4020
Anshelevich

Computer Algorithms Homework 7

9. Consider the following problem. You are managing a communication network, modeled by a directed graph G=(V, E). There are c users who are interested in making use of this network. User i(for each i = 1, 2, ..., c) issues a request to reserve a specific path P_i in G on which to transmit data.

You are interested in accepting as many of these path requests as possible, subject to the following restriction: if you accept both P_i and P_j, then P_i and P_j cannot share any nodes.

Thus, the Path Selection Problem asks: Given a directed graph G = (V, E), a set of requests $P_1, P_2, ..., P_c$ — each of which must be a path in G and a number k, is it possible to select at least k of the paths so that no two of the selected paths share any nodes?

Prove that Path Selection is NP-complete.

NP: Given a set of solution paths that are $\subseteq P_1, P_2, ..., P_c$, we can check the paths, if there are any shared nodes, it is not a valid solution, else it is. This is a polynomial checker.

Independent Set $\leq_p$ Path Selection: Input: Graph G = (V, E), and k. With this input, we will abriturarily assign a direction to each edge in the graph. Then for all vertices in the graph, create a path that includes all outgoing edges. If there is no outgoing egde from a vertex, i, then just create $P_i = \{(i)\}$. Then with these set of paths and the original k given for independent set, we input that into Path Selection. If it returns yes, then there is a set of k vertices that are all not adjacent to each other.

Proof: Path Selection → Independent Set of size k

Each path includes all the outgoing adjacent nodes from a node, therefore path selection will only choose paths that do not share nodes. As a result, if there is k paths, then there is a set of k vertices that are all not adjacent to each other.

Independent Set of size k → Path Selection

If there is an independent set of size k, there are k vertices that are not adjacent to one another. Therefore any edge leaving a vertex in this set is a path that does not share a node.