CSC373 A4

Simon Berest, berestsi

March 20, 2019

1 ...

For each tool set variable n_i to 1 if we will use the tool and 0 if we will not. The cost of buying the tools is the $\sum_{i=0}^{n} n_i c_i$ where n is the number of tools. The cost of the dependencies is $\sum_{j=0}^{m} \sum_{i=0}^{n} (1 - n_i) d_{ji}$ where m is the number of tasks and n is the number of tools.
Equation to minimize: $Cost = \sum_{i=0}^{n} n_i c_i + \sum_{j=0}^{m} \sum_{i=0}^{n} (1 - n_j) d_{ji}$
Constraints: For each tool i:
For each incompatibility with tool [0:k]: $\sum_{v=0}^{k} n_v + n_i = 1$

2 ...

Assumption: A) Does every cycle in G that starts at vertex s have no more than B? Means that each cycle has individual weight less than or equal to B rather than the combined weight of each cycle. Similar for b)
To find all cycles in a graph we run a DFS on the graph starting from the vertex s. Adding up the weight of each edge we traverse. If we hit the vertex s again we have found a cycle starting at s. At this point we do the checks on the weight described below. When we step back during the DFS we subtract the weight of the edge from the weight we are tracking. This is all done using a slightly modified DFS so it can be done in $O(|V| + |E|)$.

Each decision problem can be solved using the previous algorithm. When we find the vertex s after running the DFS we have found the weight of a cycle starting from s. For each cycle we encounter we do a check, when the DFS finishes and we haven't returned something we can then return something else.

- For A) we return false if the weight is $> B$ when we find a cycle, otherwise return true.
- For B) we return false if the weight is $< B$ when we find a cycle, otherwise return true.
- For C) return true if the weight is $>= B$ when we find a cycle, otherwise return false.

- For D) return true if the weight is $<= B$ when we find a cycle, otherwise return true.

As each decision question can be done in linear time ($O(|V| + |E|)$) all the decision problems belong to P.

3 ...

- A) By lecture 8, $D_1 - >_p D_2$ means that the inputs of D_1 can be transformed into inputs of D_2 in polynomial time to output the same answer. Similarly for $D_2 - >_p D_3$. We want to prove $D_1 - >_p D_3$ given the previous two statements. As inputs of D_1 can be transformed into inputs of D_2 in polynomial time, and inputs of D_2 can be transformed into inputs of D_3 in polynomial time, by running the inputs of D_1 through the transformation to inputs of D_2 and then running those inputs through the transformation into inputs of D_3 we have $D_1 - >_p D_3$. As running two polynomial operations consecutively can never take more than polynomial time.

- B) We know that D is also NP-complete.
  1) Claim D belongs to NP.
  $D' - >_p D$ means that there are some changes to the inputs of D' x that would allow the outputs of $D(f(x)) = D'(x)$. As the outputs of D' can be verified in polytime then so can the outputs of D. As you can just check the inverse of f(x) which is x.
  2) Claim D belongs to NP Hard.
  As D' is NP-complete it is also NP hard. This means that for all $D*$, $D*$ is NP, $D * - >_p D'$. By 3A), as $D * - >_p D'$ and $D' - >_p D$ then $D * - >_p D$ which means that D is NP hard.

- C) Claim NP = coNP.
  If D is NP-Complete this implies that D is in NP. If D is in NP this implies that for any D' in NP, $D' - >_p D$. This means that any verifier for a problem in NP can be transformed into a verifier for a problem in coNP.