Lab10-Reduction

Exercises for Algorithms by Xiaofeng Gao, 2018 Spring Semester.

∗ If there is any problem, please contact TA Jiaxi Liu (jsxzljx@163.com).

∗

1. Define that *Two-Cliques* is a pair $\langle G, k\rangle$ where G is an undirected graph that has two vertex-disjoint cliques of size k and $k+1$. Prove that *Two-Cliques* is NP-Complete. Specifically, you should

   (a) Prove that *Two-Cliques* $\in$ NP.

   (b) Prove that *Two-Cliques* is NP-Hard.

   Hint. You can use a polynomial-time reduction from *Clique* to *Two-Cliques*.

   Solution:
   Two- Cliques $= < G, k > | G$ is an unidirected graph that contains two disjoint cliques of size k
   First, We need to prove that 2-CLIQUES is an NP. A polynomial time verifier for 2-CLIQUES takes $< j, z >$ as inputs.
   if j $\neq$ Have the FORM $< G, k >$ then
   reject
   ELSE if (c = Encoding of two sets of nodes) then
   accept
   The sets are both of size k, then the two sets are disjoint. And both sets define CLIQUES in G (i.e for both sets, there is an edge connecting every pair of nodes in the set) otherwise it can reject it. This can all be done easily in polynomial time.
   Another solution to this problem is that we can show that 2-CLIQUES is in NP by describing a nondeterministic Turing machine that decides it and that runs in polynomial time.
   We need to show this in our next step that NP can be reduced in polynomial time to 2-CLiques in every language. By proving that NP-Complete can be reduced to 2-CLIQUES as NP-Complete is also know as a CLIQUE. So by doing that all languages in NP can be reduced to 2-CLIQUES using a reduction method over the CLIQUE. We could reduce the CLIQUE to 2-CLIQUE in several ways.
   Two prove the two conditions asked in the question, I have some methods that I will like to explain.
   (1) The reduction maps $< G, K >$ to $< G^*, K >$, where G^* is the graph consisting of all the nodes and edges of G, along with another K nodes that are connected to each other, but not to any of the nodes from G. This is easy to do in polynomial time.
   (2) $< G, K >$ is in CLIQUE if and only if $< G^*, K >$ is in 2-CLIQUES. For the forward direction.
   if ($< G, K >$ is in CLIQUE) THEN
   subset of k nodes of G is a CLIQUE.
   In this case it states that subset of G^* is also a CLIQUE and the nodes in G^* that are not a member of G also form a CLIQUE.
   Hence stating that there will be two disjoints cliques of size k, and $< G^*, K >$ is in 2-CLIQUES.
   On the other hand,
   IF ($< G^*, K >$ is in 2-CLIQUES) THEN
   G^* will have two two disjoints cliques of size k.
   These two cliques can not both be in the k nodes of G^* that are not in G. So there will be a clique of size k in G

Hence $< G, K >$ is in CLIQUE AS (G can also not be partly in G and partly out since there are no edges between these parts).
Reference
http://www.cs.utoronto.ca/~radford/csc363.S10/
http://www.cs.utoronto.ca/~radford/csc363.S10/test2-sol.pdf
https://www.geeksforgeeks.org/two-clique-problem-check-graph-can-divided-two-cliques
https://math.stackexchange.com/questions/310092/the-two-clique-problem-is-in-p-or-np

2. A subset S of vertices in a graph G is triad free if for all triples of distinct vertices x, y and z in S, at least one of the edges (x, y), (x, z) or (y, z) is not present in G. For example in Fig. 1:

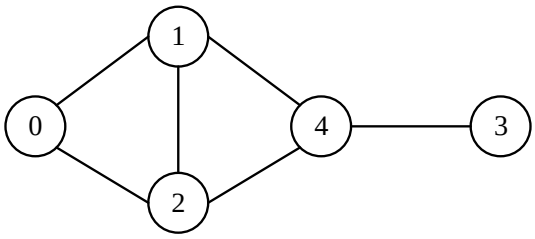

Figure 1: An example for question 2.

the set $\{0, 1, 3, 4\}$ is triad free, but the set $\{0, 1, 2, 4\}$ is not triad free. Given a graph G and an integer k, Show that the problem of determining whether G has a triad free set S of size k is NP-hard using the fact the the independent set problem is NP-Complete. Recall that the independent set problem is to determine whether a graph has a independent set of a particular size, where an independent set is a collection of mutually nonadjacent vertices.

Solution:
Create a new Graph G^*
by Replace each edge $e = (x, y)$ in G with v_e
New Edges for (x,y) will be:
(x,v_e) and (y,v_e) and (y,x).
The Vertex v_e essentially forces the edge e to be covered.
Call the Procedure for dominating set to get a minimum cardinality dominating set D.
IF ($v_e \in$ D) THEN
REPLACE (v_e,x) OR (v_e,y)
Call this new Set D^*.
Note that D^* is still donimating in G^*. Then D is a minimm cardninality vertex cover in G.
This reduction only handles the case when G is connected graph and has atleast one edge. If G is not connected then this reduction can run on each non-trivial connected component.
For Traids we can say that, independent set is like triad-freedom but looking at pairs of nodes instead of triples. So for the reduction, so we should use an operation that turns edges into triads and independent nodes into incomplete triads.
Reference:
https://people.cs.pitt.edu/~kirk/cs1510/
https://people.cs.pitt.edu/~kirk/cs1510/reductionhwsolns.pdf (16)
https://people.cs.pitt.edu/~kirk/cs1510/homework/reductionhw.pdf (17)

3. Suppose you have one machine and a set of n tasks a_1, a_2,$\cdots$, a_n. Each task a_j has a processing time t_j, a profit p_j, and a deadline d_j. The machine can process only one task at a time, and task a_j must run uninterruptedly for t_j consecutive time units. If you complete task a_j by its deadline d_j, you receive a profit p_j, but if you complete it after its deadline, you receive no

profit. As an optimization problem, you are given the processing times, profits, and deadlines for a set of n tasks, and you wish to find a schedule that completes all the tasks and returns the greatest amount of profit.

(a) State this problem as a decision problem.

(b) Show that the decision problem is NP-complete.
Hint. Reduction from Knapsack problem.

Solution:
(1): A single machine and with set of n task a_1,a_2,..,a_n. Each task a_j has a processing time t_j, A profit p_j and a deadline d_j. The machine can process only one task at a time and task a_j must run an interruptedly for t_3 consecutive time units, if task a_j by its deadline d_j. Then only it receives a profit p_j Here the discussion problem is can we get a total profit of at least K **(2):** So now we show that the subset sum reduces to this problem. by letting w_1,...w_n and W be a subset sum instance. We include the tasks a_1...a_n and set p_i=t_i=w_i for i=1,2,...n and P=d_i=W.
Then for any schedule the profit it returns equal $\sigma_{i\in s}p_i = \sigma_{i\in s}t_i = \sigma_{i\in s}w_i$ where S is the set of task scheduled to finish before common deadline.
If there is a set of S of numbers w_i that adding up W, Schedule the corresponding tasks before the deadline. and the remaining tasks afterwards.
By the observation above this schedule returns profit P=W.
If there is a schedule that returns profit at least P then this profit return is equal to P=W, Conversely.
So if we select numbers corresponding to tasks scheduled before the deadline they add up exactly P=W.

Another explanation for this is that we introduce enforcer , The enforcer ensures that if a solution exists then enforcer will always get scheduled in its enforced time slot, if it does not get scheduled in its enforced time slot that would imply that a feasible solution does not exist (or the claim of a feasible schedule is farce). Once the enforcer is in its place, the sum of processing times of the jobs before and after it will have equal total time. Also because of condition four of the RDS problem, these two sets of jobs will be non overlapping thus satisfying the PART problem requirements.

Note: For more details , please check this PDF file link that I am attaching.
https://cs.wmich.edu/gupta/teaching/cs6320/cs6320sp16web/lectureNotes6320/Lecture6_CS6320_SP16_MuaazGulAwan.pdf
Check the Article 6.2.2

Reference:
http://www.chegg.com/homework-help/suppose-one-machine-set-n-tasks-a1-a2-task-aj-pro
p-problem-4p-solution-9780070131514-exc
https://www.coursehero.com/file/p4l5mcd/As-an-optimization-problem-you-are-given-the
https://cs.wmich.edu/gupta/teaching/cs6320/cs6320sp16web/lectureNotes6320/Lecture6_CS6320_SP16_MuaazGulAwan.pdf(6.2.2)