Theory of Computational Complexity – First Assignment

CHEN PENG
Department of Social Informatics
M1 in Yoshikawa & Ma Lab
Email: chenpeng.acmer@yahoo.com
StuNo.6930302948

Problem 1

1. What are NP-complete problems?

Firstly, NP is the set of problems that can be verified by deterministic computations in polynomial time, or they are solvable in polynomial time by a non-deterministic Turing machine (multiple actions for any given situations).

A problem M is NP-complete if:

(1) it is NP

(2) every NP problem can be reduced into M in polynomial time.

If one NP-complete problem could be solved in polynomial time on a deterministic Turing machine, then every problem in NP could.

2. Is it likely that the problem of finding a perfect matching in a given graph is NP-complete?

Yes.

3. How does the complexity of solving one NP-complete problem affect the complexity of solving any problem in NP?

If one NP-complete problem could be solved in polynomial time on a deterministic Turing machine, then every problem in NP could.

4. List 10 NP-complete problems?

(1) Subset sum problem
(2) Knapsack problem
(3) Travelling salesman problem
(4) Boolean satisfiability problem
(5) Vertex cover problem
(6) Minimum k-cut problem (graph theory)
(7) Independent set problem
(8) Hamiltonian path problem
(9) Dominating set problem
(10) Graph homomorphism problem

Problem 2(NP-completeness)

Show *Graph-3-Colorability* is NP-complete.

I try to give a reduction from 3-SAT to graph-3-colorability problem in polynomial time.
(1) First, suppose φ is a 3-SAT instance, which is defined over n variables {x1, x2 ⋯ xn} and has clauses C1, C2 ⋯ Cm.

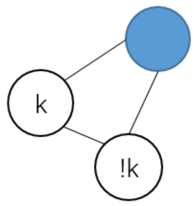
(2) Construct n isolated nodes, representing n variables, e.g. node1 for x1.

(3) Use 3 color to indicate the assignment of n nodes(variables), green for True, red for False, blue for Special State (only used to distinguish colors).

Now we should ensure these n nodes could only be colored green or red.

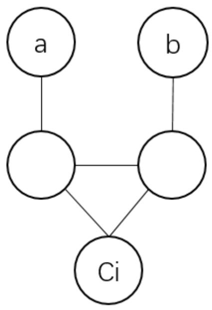
For each node, we build another node which has opposite assignment of it. Add another auxiliary node that is colored blue. Then connect them to each other.

e.g. build such a sub-graph for node k, and !k has opposite assignment of k.



After build a sub-graph for each node, we can guarantee every can only be colored green(True) or red(False).

(4) For a clause Ci = (a V b), we can express it in the following structure:

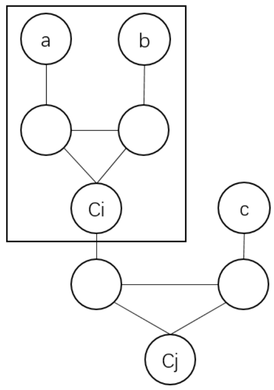


We can find that the node named Ci is able to express the result of (a V b).

If a and b are both red(False), then nodes connecting to a and b must be colored green(True) or blue and they should be different, too. Thus the Ci must be red(False).

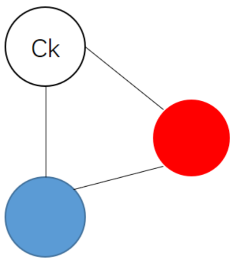
If one of a and b is green(True), we can find a valid 3-coloring that Ci is green(True).

(5) For a clause Cj = (a V b V c), it equals to ((a V b) V c). Let Ci = (a V b), then the Cj = (Ci V c). It can be expressed:



As we can see, 3-variables clause can be decomposed into two 2-variables clauses. So, a clause Cj = (a V b V c) can also be expressed by this structure in 3-coloring background.

(6) In 3-SAT, we have a series of clause like: C1 ^ C2 ^ C3 ··· ^ Cn. We need to capture the satisfiability of the whole equation. So, for each clause Ck, we build a structure to capture its satisfiability:



Now, if φ = C1^ C2 ^ C3 ··· ^ Cn is satisfiable, every clause Ck should be colored green.

(7) Proved.