Analysis of Algorithms

Homework 7 – *NP*

Abraham Murciano & Elad Harizy

January 25, 2021

1 Subgraph Isomorphism

We are given two graphs, $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$. The subgraph isomorphism problem (SGI) is to determine if there exists a subgraph of G_2 which is isomorphic to G_1.

Part A

To prove that SGI $\in$ *NP* we must show that there is a way to verify a solution to the problem in polynomial time.

If we are given a solution to an SGI problem in the form of $G_3 = (V_3, E_3)$, which is a subgraph of G_2; and a bijection $f : V_1 \to V_3$,which is a map from the vertices of G_1 to the vertices of G_3, then we can verify that they are in fact isomorphic as follows.

```
function ISOMORPHIC($G_1, G_3, f$)
    for $(u, v) \in E_1$ do
        if $(f(u), f(v)) \notin E_3$ then return false
    return true
```

This algorithm is clearly $O(E)$, so is polynomial. Therefore $SCI \in NP$.

Part B

To prove that SGI is *NP*-complete, we will show that the Clique problem (which is known to be *NP*-complete) is reducible to SGI.

The Clique problem is to determine, given a graph $G = (V, E)$ and a constant $c \leq |V|$, whether or not K_c (a complete graph with c vertices) is a subgraph of G.

To reduce the Clique problem to SGI, we must convert every instance of the Clique problem into one of SGI, such that the solution will be the same for both.

Consider some inputs to the Clique problem; a graph G, and a constant c. We can feed K_c and G as the inputs G_1 and G_2 of the SGI problem respectively. Then any SGI algorithm will tell us whether or not $G = G_2$ has a subgraph isomorphic to $K_c = G_1$, which is precisely the definition of the Clique problem.

2 Simple Cycle Problem

This problem is to find whether or not a given graph has a simple cycle (a cycle with no repeating vertices, other than the first and last) of k distinct vertices.

Part A

This problem is in *NP*. This can be easily shown since if we are given a cycle, we can walk the cycle and verify that there are precisely k vertices in linear time, and we can verify that they are all distinct in linear time with the use of a hash-set, or certainly in quadratic time by comparing all vertices to all previously seen ones. Therefore the problem is in *NP*.

Part B

We will now show that our problem is *NP*-complete by showing that the Hamiltonian Cycle problem, which is known to be *NP*-complete reduces to it.

The Hamiltonian Cycle problem goes as follows. Given a graph, is there a path that traverses all the vertices without repeating any, then ends back at the start?

For any instance of the Hamiltonian Cycle problem, supposing the input is the graph $G = (V, E)$, we can construct an equivalent Simple Cycle problem by asking is there a simple cycle in G of $k = |V|$ distinct vertices. If there is one, that is a hamiltonian cycle. Otherwise, there is no hamiltonian cycle. Thus the Simple Cycle problem is also *NP*-complete.

Part C

We are presented with another similar problem. Given a graph, does it contain *any* cycles? This problem is different to the Simple Cycle problem, in that we do not care about the size of the cycle, nor do we care if there are smaller cycles within it.

This problem can be solved in polynomial time by performing a breadth-first search on the tree, and if back-edges are found, then there must be a cycle. Otherwise there cannot be one.

3 Cliques and 3-CNF

We are given the following boolean formula in 3-CNF.

$$\begin{aligned}\varphi &= (a \lor b \lor \neg c) \land (a \lor b \lor \neg c) \land (\neg a \lor \neg b \lor c) \land (a \lor \neg b \lor \neg c) \\ &= (a \lor b \lor \neg c) \land (\neg a \lor \neg b \lor c) \land (a \lor \neg b \lor \neg c)\end{aligned}$$

This boolean formula is solvable if and only if the graph in figure 1 contains a clique of size three.

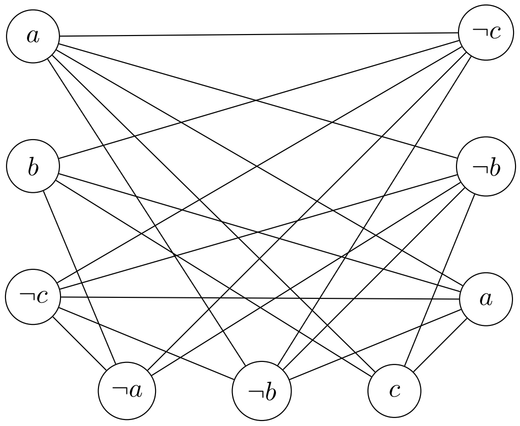

Figure 1: A graph which contains a clique of size three iff φ is satisfiable.

There are many cliques of size three in this graph, which means that there are many different ways to satisfy φ. For example, figure 2 shows a few of the different cliques contained in the graph. If we take the red clique, that corresponds to the assignments $a := \top, b := \bot$ in the context of the boolean formula φ. Clearly, if we substitute these assignments into φ, that would make $\varphi = \top$, so it is obviously satisfiable.

4 Tasks and Machines

We are given a set of n tasks with durations $T = \{t_1, t_2, \ldots, t_n\}$, a number of machines m, and a time limit l. All machines start at the same time, and continually process tasks until the time limit l is reached. The problem is to determine if all the tasks can be finished before the time limit.

Part A – An Exponential Algorithm

Algorithm 1 which checks every possible way to perform the tasks, and returns true if it finds one that finishes on time. The complexity is $O(n!)$, since there are $n!$ permutations of T.

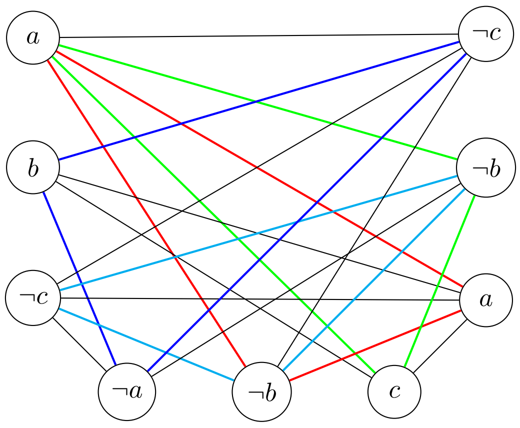

Figure 2: A graph which contains a clique of size three iff φ is satisfiable.

Algorithm 1

```
function TasksAndMachines($(T, m, l)$)
    for each permutation $P$ of $T$ do
        for $i$ from 1 to m do          ▷ Initialise working times of machines to 0
            $w_i := 0$
        $i := 0$
        for $t \in P$ do
            if $w_i + t < l$ then          ▷ Check if $t$ fits on current machine's time
                $w_i := w_i + t$          ▷ Reserve $t$ time units on machine $i$
            else
                $i := i + 1$          ▷ Move onto next machine
                if $i > m$ then
                    Continue          ▷ No more machines. Try next permutation
                $w_i := t$          ▷ Reserve $t$ time units
        return True          ▷ This permutation fits in the time limit
    return False          ▷ No permutations were found to fit
```

Part B

This problem is in NP, since if we are given a partitioning of T into at most m partitions, we can verify that the sum of each partition is less than l, and we would know that indeed, each partition can be assigned to one machine, and they would all finish before the time limit. This can be checked in polynomial time.

Part C – Reduction from the Bin Packing Problem

In the bin packing problem, n items of different volumes $V = \{v_1, v_2, \ldots, v_n\}$ must be packed into k bins or containers, each of a fixed given volume b. It is known that the bin packing problem is NP-complete.

Now to reduce the bin packing problem to our problem, we first create a task for each item whose duration is the same as the volume of the corresponding item; formally, $\forall 1 \leq i \leq n, (t_i = v_i)$. We then set the time limit l to be equal to the volume of each bin b. Finally, we set the number of machines to the number of bins available.

Now we are able to partition the tasks into partitions for each machine, if and only if we are able to partition the items into bins.