COMP3821 Assignment 4

April 2021

1.1 **Solution:** This is a linear programming problem. This is because the objective function as well as the equalities/inequalties are linear. An example is the amount of flavour being dispensed doesn't have to take an integer value, rendering the question to be a linear programming problem.

1.2 **Solution:** The variables are as follows:
B: maximum cost of frozen yoghurt he can buy
u_i: satisfaction score for a flavour i
y_i: amount of yoghurt flavour i Andrew buys, per litre n: number of flavours
p_i: price of flavour i (in $\$/l$)
M: maximum amount of frozen yoghurt, in litres

1.3 **Solution:** The constraints are as follows:

$\sum_{i=1}^{n}\mathrm{y}_i * p_i <= \mathrm{B}$.

$\sum_{i=1}^{n}\mathrm{y}_i <= \mathrm{M}$.

1.4 **Solution:** The objective function is as follows:

Maximize $\sum_{i=1}^{n}\mathrm{y}_i * u_i$.

1.4 **Solution:** Yes, a solution can be derived in polynomial time, since Linear Programming (LP) is in P.

2.1 **Solution:** This problem can be formulated with a integer linear programming (ILP) solution. This is because the solution output is required to be an integer (each flavour is now sold in 1-litre tubs), justifying the ILP choice.

2.2 **Solution:** The changes are as follows:
y_i: 1-litre tubs of yoghurt flavour i Andrew buys
The objective function and constraints remain unchanged.

2.3 **Solution:** ILP is NP-Hard. Thus, a solution for the ILP problem cannot be derived in polynomial time.

3.1 **Solution:** For the Subset Sum Dynamic programming solution:

Base Cases:
- If n = 0, return true.
- If a sum of A is odd return false. This is because if the sum of the entire array is odd it cannot be partioned into 2 sub-arrays of equal sums.

Sub-problem: Find 2 subsets of K[i,..,n] which don't have any common indices, and both their sums are $\sum_A$[K$_i$/2].

Recurrence Relation:
Let SubsetSum(K, n, sum/2) be the function that returns true if there are 2 subsets that meet the constraints of the sub-problem as described above.

SubsetSum (K, n, sum/2) = 1) SubsetSum (K, n-1, sum/2 - K[n-1])
or 2) SubsetSum (K, n-1, sum/2)

1) of the recurrence relation represents the sub-problem taking into account the last element of the array.

2) of the recurrence relation represents the sub-problem without taking into account the last element of the array.

3.2 **Solution:** The time complexity is $O(sum * n)$. The dynamic programming solution can be made with a 2D-array subset[x][y]. The value stored will be true if a subset of A[0], A[1],...A[n-1] has sum equal to i, otherwise false.

The bottom-up solution will have a 2 loops, with the inner loop checking the recurrence relation is satisfied for each index of the array, and sum/2 being the maximum possible sum value.

3.3 **Solution:** The SubsetSum problem can be reduced to the subset sum problem which is NP-Complete (which means it is both NP and NP-Hard). Thus, the existence of the algorithm does not prove that SubsetSum is in class P.

4.1 **Proof:** To prove 3SAT is in class NP we will show that SAT is in class NP.

SAT can be expressed as a test for existence: (let K be an instance of SAT) K is satisfiable <=> there exists an assignment X of truth values to the variables in K so that the assignment X makes K true.

A non-deterministic algorithm for SAT:
An assignment of truth values to the variables that occur in K, which is accepted if the assignment makes K true.

3-SAT is just a restriction of SAT where each clause is required to have exactly 3 literals. A non-deterministic algorithm for SAT will also work for 3SAT. The algorithm is not affected by the restriction to 3 literals per clause for 3SAT. Therefore, since SAT is in NP and can be reduced to 3SAT, 3SAT is also in NP.

4.2 **Proof:** To prove G3C is in NP, the certificate of G3C can be verified in polynomial time in this way: for each edge x, y in graph G, verify that the color c(x) != c(y).

Hence, the assignment can be checked for correctness in the polynomial-time of the graph with respect to its edges in O(V+E). Since the certificate is verified in polynomial time G3C is in NP.

4.3 **Solution:** By the Cook–Levin theorem, we know we can reduce G3C to 3SAT.

We are given a graph G = (V, E) and the constant set of colours C = 1, 2, 3. For every vertex m, let the boolean variable m_1 represent the fact the m-th vertex is of colour 1. We can similarly do the same for m_2 and m_3.

Suppose two vertices m and n were connected by an edge e. Consider the clause $(\neg m_1 \vee \neg n_1)$. If we demand the clause is true, it means that the vertices cannot both be colour 1 at the same time. The bigger clause considering all 3 colours would be:

$(\neg m_1 \vee \neg n_1) \wedge (\neg m_2 \vee \neg n_2) \wedge (\neg m_3 \vee \neg n_3)$.

The above 2SAT clause states that vertices m and n are not of the same colour. Each vertex can either be colour 1, 2 or 3 thus:

$(m_1 \vee m_2 \vee m_3)$ for every vertex m in the graph.

Adding this clause to the above 2SAT clause we get:
$(m_1 \vee m_2 \vee m_3) \wedge (\neg m_1 \vee \neg n_1) \wedge (\neg m_2 \vee \neg n_2) \wedge (\neg m_3 \vee \neg n_3)$.

Converting this 2SAT to a 3SAT clause k_e:
$(m_1 \vee m_2 \vee m_3) \wedge (\neg m_1 \vee \neg n_1 \vee \neg n_1) \wedge (\neg m_2 \vee \neg n_2 \vee \neg n_2) \wedge (\neg m_3 \vee \neg n_3 \vee \neg n_3)$

Thus for every edge in the graph, we can have a conjunction of k_e as follows:

$k = \wedge_e \in Ek_e$
Thus we have a reduction from G3C to 3SAT.

We can clearly see that a Yes instance of G3C results in a Yes instance of 3SAT and a No instance of G3C results in a No instance of 3SAT by the reduction (by plugging in 1 when an edge is valid in k_e will result in output of 1; 0 when an edge has two of of the same colourn will have an output 0). Thus proven.

4.4 **Solution :** [B] If G3C is NP-complete, then 3SAT is NP-complete.

END.