500468777 COMP3027 A4

Problem 1

Part A

Let p be the original problem of finding the minimum value of b and d be the decision problem given some value b'. We run d for increasing values of b', starting at 1 until the decision problem returns true. Then the minimum value of b is the smallest value of b' for which d returns true.

Part B

b = n is a trivial solution for the decision problem, as assigning a base to every planet possible will ensure there is at least one base on a node at each edge. Thus, the minimum value of b ≤ n. Therefore, assuming the decision problem, d, is correct as b must be an integer, testing every possible value of b = b' for b' = 1, 2, ..., n ensures that firstly, a solution will be found (as b = n is a trivial solution) and secondly the minimum will be found as every possible solution is checked incrementally from the minimum.

This process involves O(n) calls to d. At each value of b' we run d without any modifications to the graph and then either return true or check the next value. Thus, if d has time complexity O(p(n)) then the reduction has time complexity O(np(n)).

Part C

Let CERT(G, B, b) be a certifier for the decision problem where G = (V, E) is a graph, B is the proposed set of vertices and b is the size limit of B.

CERT must check the following:

1. $\forall v \in B : v \in V$
2. $|B| \leq b$
3. $\forall v \in V \backslash B, \exists u \in B : (u, v) \in E$

The first can be done by a linear search through V for each vertex in B. There are at most n vertices in V and thus this check will run O(n^2) time.

The second can be done in O(n) time by counting the number of vertices in B and checking if it is less than b.

The last can be done by first finding the set S = V \ B. Then for each vertex u $\in$ B, check each of its edges e = (u, v). If V is in S then remove it and continue. If by the end S is empty, the third condition is met. Finding the set S can be done in O(n^2) time by a linear search through B for each vertex in V. Then, there are at most n^2 edges out of B (by the handshake lemma), at each we iterate through O(n) edges to check if the edge is in S. Thus it can be done in O(n^3) time.

Therefore, we can certify a solution in polynomial time and hence, the decision problem is in NP.

Part D

As the decision problem, d is in NP we now show that the set-cover problem, which is NP complete, reduces to it. That is, we prove set-cover ≤$_p$ d.

The set cover problem (SC) asks given:

- A set U of integers
- A set of sets S = {S$_1$, S$_2$, ..., S$_n$: S$_i$ $\subseteq$ U $\forall 1 \leq i \leq n$}
- An integer k
  Is there at most k sets in S whose union equals U?

To reduce to d, create a graph G with the following:

1. Create a node u$_i$ for each number in the set U (U layer of graph)

2. For each S$_i$ in S:
   a. create a node s$_i$ (S layer of graph)
   b. for each number c in S$_i$ create a forward edge from s$_i$ to u$_c$
3. Then connect every node in the s layer with an edge in either direction

Under this construction, the U layer of the graph represents the original set U and the S layer represents which numbers in U each subset contains. Further, the only forward edges available are those from a node in the S layer to one in the u layer or another node in the S layer. Thus, the only feasible additions to the set B in problem d are those in the S layer (adding a node in the U layer is equivalent to not taking any node as there are no edges out of that node). In doing so, the nodes added to the set B in d represent the collection of sets in S chosen in the set cover problem. Additionally, connecting all the nodes in the S layer ensures that any node that doesn't satisfy the condition must be in the U layer.

Proof SC = True $\implies$ d = True:
Assume there is a valid solution to the set cover problem. Then there exists at most k subsets whose union is equal to U. After reduction to and instance of d, each set in SC maps to a unique node in the S layer of the graph in d. Further, each of these sets are connected to a node in the U layer if the original subset contained that value. Therefore, there must be at most k nodes in the S layer where each node in the U layer is connected to at least one. Further, every node in the S layer is connected to every other layer. Thus, there is a valid solution to d.

Proof d = True $\implies$ SC = True:
Assume there is a valid solution to d. Then there are at most k nodes in the S layer such that every other node in the graph is connected to at least one. As the S layer forms a clique this is automatically satisfied for that subgraph. However, all of the nodes in the U layer have an edge into them from a node in the S layer as the requirements for d was satisfied. The edges out of a node in the S layer correspond to the values in a subset in SC. Thus as there are at most k nodes in the S layer that connect to every node in the U layer, there must be at most k subsets in SC whose union equals U.

Part E

We reduce the decision problem d to the original problem P.

Firstly, we run P which returns the minimum integer b for which there exists a subset of b nodes in the graph such that every node in the graph is at most one edge away from a node in b.

Now let b' be the input to the decision problem d, then the decision problem is true if b ≤ b' ≤ |V|.

Proof b' $\notin [b,n] \implies d = False$
We know if b' > |V| then there is no possible set of b' nodes as there are not enough nodes in the graph. Further, if b' < b then there are also no possible subsets of size b' as b is the minimum size that satisfies the requirements. Thus if b' not in the given range we return false.

Proof b' $\in [b,n] \implies d = True$
As P returned b we know a satisfying subset of b nodes exists. Adding any extra node in the graph to this subset found will not increase the distance of any node to the subset as all edges that existed from the subset to a node outside still exist. Thus we can add any number of nodes in V \ B to that set and still have a satisfying subset. When B = ϕ, |V \ B| = |V| = n. Thus there exists a satisfying subset of any size between b and n.

Thus d ≤$_p$ P and therefore the original problem is NP hard as D is NP complete.
Further, as only decision problems can be NP complete the original problem cannot be NP complete and is only NP hard.

Problem 2

Part A

There are n planets that need to be stored. Each planet has:

1. Quadrant number between 1 and k $\implies$ log k bits
2. Cost value ≤ D to be relevant (otherwise can ignore planet) $\implies$ log D bits
3. Total distance ≤ Δ $\implies$ log Δ bits

Thus log k + log D + log Δ = log(kDΔ) bits for each planet of the n planets plus the input of n which takes log n bits. So total is upper bounded by O(n log(kDΔ) + log(n)) = O(n log(kDΔ))

Part B

Is there a choice of exactly one planet l_i from each quadrant Q_i such that $\Sigma_{1=i}^{k} c_{l_i} \leq D \; and \; \Sigma_{i=0}^{k} \delta_{l_i} \leq M$ for some non-negative numbers M and D.

Part C

For set of quadrants Q, set of chosen planets P, max cost D and max distance M define Cert(Q, P, D, M) which must check the following:

1. For each quadrant in Q there is exactly 1 planet in P
2. The sum of costs of planets in P is less than D
3. The sum of distances of planets in P is less than M

The first can be done in $O(k^2)$ time by first iterating through the quadrants in Q for each planet in P and checking each planet is assigned to a quadrant in Q. Then the same is for each quadrant in Q checking there is an assigned planet in P. If each planet in P has a quadrant in Q and each quadrant Q has a planet in P then there is exactly 1 planet chosen from each quadrant as |Q| = |P| = k.

The last two can be done in linear time by summing the cost and distances of each planet in P and checking if they are less than D and M respectively.

Thus, there exists a polynomial time certifier, and thus the decision problem is in NP.

Part D

Consider the subset sum problem (SSS):
Given a set of positive integers S, and an integer t, is there a subset of numbers S' in S that sum to exactly t?

We reduce from SSS(S, t) to the quadrant decision problem from part B, QDP(Q, D, M). To begin, we create an instance of QDP from an instance of SSS. First find s$_{max}$, the largest value in S. Then, for each number s in S, create a quadrant in Q containing two planets. The first with δ = s and c = s$_{max}$ - s and the second with δ = 0 and c = s$_{max}$. Then we run the QDP using the constructed quadrants, D = |S|s$_{max}$ - t and M = t which returns true if and only if there is a valid subset sum.

Proof SSS = True $\implies$ QDP = True:
Assume there is a valid subset of S that sums to t. After the reduction, each integer s$_i$ in S is mapped to exactly one quadrant Q$_i$ in Q with:

$$Q_i = \{p_1^i = (\delta = s_i, c = s_{max} - s_i), p_0^i = (\delta = 0, c = s_{max})\}$$

Under this construction, selecting p$_1^i$ is equivalent to adding s$_i$ to the subset in the SSS while choosing p$_0^i$ is equivalent to not selecting it. As there is a valid SSS, there must be a valid subset P$_1$ = $\{p_1^i \mid 1 \leq i \leq |S|\}$ such that:

$$\begin{aligned} \sum_{p \in P_1} \delta_p &= t \quad and \\ \sum_{p \in P_1} c_p &= |P_1| s_{max} - \sum_{p \in P_1} \delta_p \\ &= |P_1| s_{max} - t \end{aligned}$$

As these are in separate quadrants, choosing one will not restrict the ability to choose another. Set P$_0$ = $\{p_0^i \mid p_1^i \notin P_1\}$ such that:

$$\begin{aligned} \sum_{p \in P_0} \delta_p &= 0 \quad and \\ \sum_{p \in P_0} c_p &= |P_0| s_{max} - \sum_{p \in P_1} \delta_p \\ &= |P_0| s_{max} \end{aligned}$$

Then for $P = P_1 \cup P_0$:

$$\begin{aligned} \sum_{p \in P} \delta_p &= \sum_{p \in P_1} \delta_p + \sum_{p \in P_0} \delta_p \\ &= t + 0 \\ &= t \\ &\leq M \end{aligned}$$

And:

$$\begin{aligned}\sum_{p\in P} c_p &= \sum_{p\in P_1} c_p + \sum_{p\in P_0} c_p \\ &= |P_1|s_{max} - M + |P_0|s_{max} \\ &= |P_1 \cup P_0|s_{max} - t \\ &= |S|s_{max} - t \text{ as } |P_1 \cup P_0| = |Q| = |S| \\ &\le D\end{aligned}$$

Therefore, if there exists a valid solution to SSS then there will exist a valid choice of exactly one planet from each quadrant such that $\sum_{p\in P} c_p \le D$ and $\sum_{p\in P} \delta_p \le M$ and hence a valid solution to QDP.

Proof QDP = True $\implies$ SSS = True:
If the quadrant decision problem returns True, then there exists a set P, with |P| = |S| containing exactly one planet from each quadrant whose costs and distances sum to less than D and M respectively. That is:

$$\sum_{p\in P} \delta_p \le t \qquad and \qquad \sum_{p\in P} c_p \le |S|s_{max} - t$$

Let $P_1 = \{p \in P \mid \delta_p \neq 0\}$ and $P_0 = \{p \in P \mid \delta_p = 0\}$ such that $P = P_1 \cup P_0$. Then we have:

$$\begin{aligned} t &\ge \sum_{p\in P_1} \delta_p + \sum_{p\in P_0} \delta_p \\ &= \sum_{p\in P_1} \delta_p \end{aligned}$$

And:

$$\begin{aligned} |S|s_{max} - t &\ge \sum_{p\in P_1} c_p + \sum_{p\in P_0} c_p \\ &= |P_1|s_{max} - \sum_{p\in P_1} \delta_p + |P_0|s_{max} - \sum_{p\in P_0} \delta_p \\ &= (|P_1| + |P_0|)s_{max} - \sum_{p\in P_1} \delta_p \\ &= |P|s_{max} - \sum_{p\in P_1} \delta_p \\ &= |S|s_{max} - \sum_{p\in P_1} \delta_p \\ \implies -t &\ge -\sum_{p\in P_1} \delta_p \\ \implies t &\le \sum_{p\in P_1} \delta_p \end{aligned}$$

Thus $\sum_{p\in P_1} \delta_p \le t \le \sum_{p\in P_1} \delta_p \implies t = \sum_{p\in P_1} \delta_p$. A planet in P_1 corresponds to adding it to S' in SSS and a planet in P_0 means it was not chosen. By construction, each planet maps to exactly one quadrant and there is a valid solution of QDP with distances summing to exactly t. As the distances in P_1 are exactly the values of the numbers in S there exists a subset of S that sums to exactly t.

Thus, SSS = True iff QDP = True $\implies$ SSS $\le_p$ QDP. As SSS is NP complete and QDP is in NP it too is NP complete.

Part E

To show the search problem QSP is NP hard, we show QDP $\le_p$ QSP.

QDP asks if there is a valid selection of exactly one planet from each quadrant with summed distance and summed cost less than M and D respectively.

We run QSP for value D which returns the set of exactly one planet from each quadrant that minimizes the summed total distance possible with summed cost at most D. Let this set of planets = P_s and M_s be the sum of these distances. If M ≥ M_s then there is a valid selection of planets with summed total cost less than or equal to D and summed distance less than or equal to M as QSP minimizes the value of M_s. Therefore, no selection of planets exist with M < M_s and QDP is True iff M ≥ M_s.

Thus, QDP reduces to QSP. As QDP is NP complete, QSP is NP hard.

Part F

Define OPT(q, d) to be the set of exactly 1 planet from the first q quadrants that minimizes the sum of distances while keeping the total cost less than or equal to d.

Then

$$OPT(q,d) = \min_{p \in Q_q} \begin{Bmatrix} OPT(q-1, d-c_p) + \delta_p & if\ c_p \le d \\ \infty & otherwise \end{Bmatrix}$$

We also define a utility lookup array, A, such that A$_{q,d}$ stores the following:

1. The value calculated by OPT(q, d)
2. The planet in q identified

The base case occurs when q is 0 meaning there are no more quadrants to search. In this case, return 0. That is $\forall d : OPT(0, d) = 0$

Then the minimum summed total distance for cost D across the Q quadrants is achieved by OPT(|Q|, D) and the exact set of these planets, P (initialized to be empty) can be found by searching through array A starting at $A_{|Q|,D}$ and at each A$_{q,d}$ we add the planet p to P and then search A$_{q-1,d-c_p}$ until q = 0 then return P.

Part G

Proof by induction is as follows.

For the base case (when q = 0) OPT will correctly return 0. Further, for q = 1, the algorithm will correctly choose the base with cost less than d that minimizes $OPT(0, d - c_p) + \delta_p = \delta_p$. That is, it will choose the base with the smallest total distance that doesn't exceed the maximum cost. If none are possible, infinity is returned.

Now assume that the algorithm correctly calculates $OPT(q', d')\ \forall 0 \le q' < q\ and\ \forall 0 \le d' \le d.$

Then, OPT(q, d) will find $OPT(q-1, d-c_p) + \delta_p\ \forall p \in Q$ which is defined to be ∞ if $c_p > d$. By the inductive hypothesis, the calculation of OPT for smaller q's and d's is done correctly. Thus, if no allocation is possible, all values will be infinity. If one or more possible allocations exists, their value will always be less than ∞ and the one that minimizes the total summed distance will correctly be chosen.

Further, at each step we either recurse for q decremented by 1 and d decremented by the cost of the chosen planet or return ∞. Thus, either OPT will return infinity, removing that allocation as a viable option or the base case will be reached. Therefore, if OPT correctly calculates the minimum value for all q' less than q and d' less than d, it will correctly calculate OPT for q and d. As for q = 1 and for all d the correct value is returned, by induction, OPT will correctly calculate OPT for all q and d. Hence, OPT(|Q|, D) will find the minimum allocation for each quadrant in Q and cost limit D.

Part H

Each iteration of OPT(q, d) makes $|Q_q|$ calls to OPT, and this can be done for all values of cost up to d. Thus, there are O(d|Q*q|) calls to OPT at each step of the recursion. In total, this is done for all values of q' up to Q totaling $O(d\sum*{i=1}^q |Q*i|)$ calls to OPT. If q = |Q|, $\sum*{i=1}^{|Q|} Q_i = n$, meaning there are O(dn) calls to OPT. Finally, once the base case is reached and values are returned, each call to OPT can be evaluated in constant time. Thus, the total time of the OPT algorithm for OPT(|Q|, D) is O(nD). Finally, the exact set of planets P can be found in linear time, as it takes exactly O(|Q|) < O(n) lookups in the utility array. Thus the overall running time of the algorithm is O(nD)