Graph optimization
Lab Project

Ewan Decima (11109703)
Lukas Hjelmstrand (11107015)
Marco Meinardi (10783486)

May / June 2025

Contents

Question 1

Model

Let's write down the model. First we have the following sets and parameters:

- N set of nodes in the network.
- E set of edges in the network.
- c_i cost of installing a device at node $i \in N$.
- $t_{i,j}$ propagation delay on edge $(i, j) \in E$.
- $d_{i,j}$ shortest path distance from node i to j.
- $M = (M_{ij})_{i,j \in N}$ a reachability matrix, such that M_{ij} is equal to one if $d_{i,j} \leq T$, zero otherwise.

and those decision variables:

- $y_i \in \{0, 1\}$, a binary variable equal to one if a device is installed at node $i \in N$, zero otherwise.

The objective function is then

$$\min \sum_{i \in N} c_i y_i \tag{1}$$

and is subject to the following constraints:

- Coverage constraint: Each node must be served by at least one device:

$$\sum_{i \in N} M_{ij} y_i \geq 1 \qquad \forall j \in N$$

Result

We obtain the following

Instance	Optimal value	Selected nodes	CPU time (s)
1	2	2; 5	0.268
2	25	13; 17; 19	0.081
3	129	1; 7; 13; 15; 21; 22; 28; 29; 34	0.127
4	256	11; 12; 14; 16; 17; 18; 26; 27; 36; 43; 46; 51; 53; 54; 57; 60; 66; 71; 75; 76	0.456

Table 1: Results for each instance

Question 2

The heuristic we designed takes an adaptive greedy approach to find a feasible solution to the problem. As in the previous question we premcompute the reachability matrix M_{ij} using the Floyd-Warshall algorithm. The algorithm starts with an empty solution and iteratively assigns a device to a node until a feasible solution is found where all nodes are served by a device. It keeps track of the nodes that have been searched $N^{Unsearched}$ and the nodes that are not yet served by a device in the solution $N^{Unserved}$. At each iteration, the algorithm selects the node i that minimizes the ratio where c_i is the cost of installing a device at node i, and the denominator is the number of new nodes that would be served by installing a device at i. In other words, we select the node where the cost per newly served node is minimized. Formally the node to be considered is determined:

$$\text{select i} \in N^{Unsearched} | min(\frac{c_i}{1 + \sum k \in N^{Unserved} M_{ij}})$$

These are the steps taken by the algorithm at each iteration:

- If solution is feasible end iteration.
- Consider the cheapest node using our defined heuristic.
- If installing a device on that node can serve 1 or more unserved nodes then add it to the solution.

A feasible solution will always be found as in worst case the algorithm will install a device on all nodes and this case is always a feasible solution.

Results

Instance	Solution Cost	Selected nodes	CPU time (s)
1	2	2; 5	0.002
2	36	1; 5; 8; 10; 11	0.013
3	147	7; 8; 10; 14; 15; 16; 25; 26; 29; 34; 40	0.083
4	287	1; 3; 10; 11; 12; 18; 23; 24; 25; 27; 29; 33; 53; 56; 57; 59; 62; 64; 71; 79; 80	0.599

Table 2: Results for each instance

Question 3

This problem can be represented as a *set covering problem.*

To convert our problem into a *set covering problem* we precompute the shortest distance between each pair of nodes in the graph, which can be easily done using the Floyd-Warshall algorithm in $\mathcal{O}(n^3)$ time, where n is the number of nodes in the graph. Then we construct the reachability matrix by checking the distance between each pair of nodes.

$$M \in \mathcal{M}_{n\times n}(\{0,1\})$$
$$M_{ij} = \begin{cases} 1 & \text{if } d_{ij} \le T \\ 0 & \text{otherwise} \end{cases}$$

We can solve the *set covering problem* using the columns of the reachability matrix as the possible subsets to obtain the solution to the original problem.

Question 4

Parameters

- $G = (N, A)$: the directed graph with nodes N and arcs A.
- d_k: data to send from node $k \in N$.
- c_i: cost of installing devices at node $i \in N$.
- cap_i: capacity of devices installed at node $i \in N$.
- T: maximum allowed propagation time for data.
- t_{ij}: propagation delay on arc $(i, j) \in A$.
- g_{ij}: cost of installing channels on arc $(i, j) \in A$.
- u: capacity of each channel installed on arcs $(i, j) \in A$.
- M_{ki}: one if node $k \in N$ is reachable from node $i \in N$ through the shortest path, zero otherwise. If $d_k = 0$, then $M_{ki} = 0 \; \forall i \in N$.

Decision Variables

- x_{ki} binary: one if data is sent from node k to node i, zero otherwise.
- y_i integer: number of devices installed at node i.
- z_{ij} integer: number of channels installed on arc (i, j).
- w_{ij}^k binary: one if data from node k is sent on arc (i, j), zero otherwise.

Forced Variables

- $d_k = 0 \implies x_{kk} = 1 \quad \forall k \in N$
- $M_{ki} = 0 \implies x_{ki} = 0 \quad \forall k \in N, i \in N \mid k \neq i$
- $M_{ki} = 0 \vee M_{kj} = 0 \implies w_{ij}^k = 0 \quad \forall k \in N, (i, j) \in A$

Constraints

1. Each node must send its data to exactly one other node.
2. For each node, enough devices must be installed to handle the data sent to it.
3. For each arc, enough channels must be installed to handle the data sent on that arc.
4. The propagation time must not exceed the maximum allowed time.

5. Flow constraints: ensure that one and only one path from node k to node i exists if $x_{ki} = 1$; ensure that no data is sent on the network if a node sends data to itself ($x_{kk} = 1$).

Model

$$\min \quad \sum_{i \in N} c_i y_i + \sum_{(i,j) \in A} g_{ij} z_{ij}$$

s.t.

1. $$\sum_{i \in N \mid M_{ki}=1} x_{ki} = 1 \qquad \forall k \in N \mid d_k > 0$$
2. $$\sum_{k \in N \mid M_{ki}=1} d_k \; x_{ki} \leq cap_i \; y_i \qquad \forall i \in N$$
3. $$\sum_{k \in N \mid M_{ki}=1 \wedge M_{kj}=1} d_k \; w_{ij}^k \leq u \; z_{ij} \qquad \forall (i,j) \in A$$
4. $$\sum_{(i,j) \in A \mid M_{ki}=1 \wedge M_{kj}=1} t_{ij} \; w_{ij}^k \leq T \qquad \forall k \in N \mid d_k > 0$$
5. $$\sum_{(i,j) \in A \mid M_{kj}=1} w_{ij}^k - \sum_{(j,i) \in A \mid M_{kj}=1} w_{ji}^k = \begin{cases} (1 - x_{kk}) & \text{if } i = k \\ -x_{ki} & \text{if } i \neq k \end{cases} \qquad \forall k \in N, \; i \in N \mid M_{ki} = 1$$

$$x_{ki} \in \{0,1\} \quad \forall k \in N, i \in N$$
$$y_{ij} \in \mathbb{Z}^+ \quad \forall (i,j) \in A$$
$$z_i \in \mathbb{Z}^+ \quad \forall i \in N$$
$$w_{ij}^k \in \{0,1\} \quad \forall k \in N, (i,j) \in A$$

Results

Instance	**Optimal value**	**CPU time (s)**
1	85	0.095
2	328	0.079
3	376	0.039
4	149	3.225
5	154	0.543
6	424	13.843
7	420	10.754

Question 5

New constraints

The problem is equivalent to a multicommodity flow problem where we don't know a priori the demands' targets. We recall the single node cut set inequalities to be:

$$\sum_{(k,i)\in A} u\ z_{ki} \geq d_k \quad \forall k \in N \mid d_k > 0$$

Which can be rewritten to reduce the feasible region of the continuous relaxation, without changing the integer feasible region:

$$\sum_{(k,i)\in A} z_{ki} \geq \left\lceil \frac{d_k}{u} \right\rceil \quad \forall k \in N \mid d_k > 0$$

This requires to be adjusted to account self serving nodes. It can be done by nullifying the constraint in the case a node is serving itself:

$$\sum_{(k,i)\in A} u\ z_{ki} \geq (1 - x_{kk})\ d_k \quad \forall k \in N \mid d_k > 0$$

Which will be rewritten as:

$$\sum_{(k,i)\in A} z_{ki} \geq (1 - x_{kk}) \left\lceil \frac{d_k}{u} \right\rceil \quad \forall k \in N \mid d_k > 0$$

There is still one problem: for the provided instances and the way we formulated the model in question 4, the optimal solution of the continuous relaxation is just to let every node to serve itself, installing a fraction of a device to satisfy its demand. This way there is no flow in the arcs and the cut set inequalities would be useless.

To fix this issue, we introduce an additional helper constraint:

$$x_{ki} \leq y_i \quad \forall k \in N, i \in N \mid M_{ki} = 1$$

This way, if a node wants to completely serve another node or itself, it must install a whole device. Since x_{ki} is binary and y_i integer, it is trivial to see that no integer solution is removed by this constraint.

Results

Linear relaxation (equivalent to linear relaxation + single node cuts):

Instance	**Optimal value**	**o.f. − integer o.f.**	**Solver CPU time (s)**
1	28.669	56.33 (66.27 %)	0.022
2	157.299	170.70 (52.04 %)	0.020
3	157.299	218.70 (58.17 %)	0.013
4	106.416	42.58 (28.58 %)	0.020
5	106.416	47.58 (30.90 %)	0.028
6	99.705	324.30 (76.48 %)	0.065
7	99.705	320.30 (76.26 %)	0.079

Linear relaxation + helper constraints:

Instance	**Optimal value**	**o.f. − integer o.f.**	**Solver CPU time (s)**
1	79.061	5.94 (6.99 %)	0.017
2	303.310	24.69 (7.53 %)	0.014
3	349.518	26.48 (7.04 %)	0.014
4	115.983	33.02 (22.16 %)	0.023
5	107.492	46.51 (30.20 %)	0.033
6	160.606	263.39 (62.12 %)	0.085
7	160.606	259.39 (61.76 %)	0.082

Linear relaxation + helper constraints + single node cuts:

Instance	**Optimal value**	**o.f. − integer o.f.**	**Solver CPU time (s)**
1	83.018	1.98 (2.33 %)	0.018
2	310.035	17.96 (5.48 %)	0.021
3	360.366	15.63 (4.16 %)	0.014
4	138.967	10.03 (6.73 %)	0.028
5	140.057	13.94 (9.05 %)	0.027
6	406.845	17.15 (4.05 %)	0.094
7	406.764	13.24 (3.15 %)	0.092

As we can see from these results, the mix of the helper constraint and the single node cut set inequalities is incredibly effective, bringing us really close to the value of the integer solution.

Question 6

New constraint

We have two knapsack constraints in question 4, the second for the devices capacity and the third one for the channels capacity. We focus on the devices only, since they are way more effective and considering the channels would lead to a huge number of additional constraints.

To be able to generate our cover constraints, for each node i we assume the items to be the demands of the nodes k that can reach i and the knapsack capacity to be the one of one single device cap_i. This works well for our instances, since we never have to install more than one device (both in the continuous relaxation and in the integer oprimal solution). We can generate a cover with these parameters, but we need to keep in mind that they might be wrong, since I can have no device or multiple devices installed on one node. Thus the cover constraint for a cover C will be:

$$\sum_{k \in \varepsilon(C)} x_{ki} \leq \max(0, |C| - 1 + B \cdot (y_i - 1)) \quad \forall i \in N$$

- The max function is not linear, but can be replaced using an additional variable, which must be greater than both arguments of the function.
- $\varepsilon(C)$ is the extended cover of C.
- C and $\varepsilon(C)$ never contains nodes which have zero demand.
- B is a big constant.

Let's consider the integer solutions and check that none is eliminated by this constraint.

- If y_i is zero, no device is installed and no node can be assigned to i and, indeed, we have zero on the right hand side, since $B \gg |C| - 1$.
- If y_i is one, this is the original extended cover inequality.
- If y_i is greater than one, the constraint is always satisfied, again, because $B \gg |C| - 1 < \varepsilon(C)$.

By looking at the continuous relaxation, we can see that not only we have the benefits of the standard cover inequality, but we are also forcing y_i to be greater or equal than one, eliminating the possibility of a fractional number of devices between zero and one, which is our biggest concern for the provided instances. This is because if we try to install a number of devices which is slightly less than one, the whole right hand side gets dominated by the $B \cdot (y_i - 1)$ term, bringing everything to zero and preventing the assignment of any node in $\varepsilon(C)$ to i.

Cover heuristic

We use the following procedure to generate the cover inequalities:

1. Solve the continuous relaxation.
2. If the new objective function increased by less than a relative threshold, then stop.
3. For each node $i \in N$:
    - 3.1. Sort the nodes that can reach i by $x^*_{ki} \cdot d_k$ in descending order (break ties with indices).
    - 3.2. Get elements from the sorted list until the sum of their demands is greater than cap_i.
    - 3.3. If we cannot exceed cap_i, do nothing and continue to the next node.
    - 3.4. Extend the cover by adding all the nodes which demand is greater than the biggest demand of the nodes inside the minimum cover.
    - 3.5. Add the cover constraint using the extended cover and the cardinality of the minimum cover.
4. Go to step 1.

Results

Linear relaxation:

Instance	Optimal value	o.f. − integer o.f.	CPU time (s)
1	28.669	56.33 (66.27 %)	0.022
2	157.299	170.70 (52.04 %)	0.020
3	157.299	218.70 (58.17 %)	0.013
4	106.416	42.58 (28.58 %)	0.020
5	106.416	47.58 (30.90 %)	0.028
6	99.705	324.30 (76.48 %)	0.065
7	99.705	320.30 (76.26 %)	0.079

Linear relaxation + cover inequalities ($B = 10 \cdot |N|$, $threshold = 1\%$):

Instance	Optimal value	Covers	Iterations	o.f. − integer o.f.	CPU time (s)
1	78.350	36	4	6.65 (7.82 %)	0.315
2	280.032	30	4	47.97 (14.62 %)	0.298
3	188.458	6	3	187.54 (49.88 %)	0.055
4	113.583	24	3	35.42 (23.77 %)	0.375
5	106.746	7	2	47.25 (30.68 %)	0.072
6	161.551	135	6	262.45 (61.90 %)	15.141
7	162.889	162	7	257.11 (61.22 %)	31.358

Unfortunately, the proposed solution is neither efficient nor effective in most cases. The most likely cause is the fact that the capacities we are dealing with are not constant and handling this greatly reduces the effectiveness of the cover inequalities.