Group Project Proposals

Alberto Santini

DMO — Fall Term 2024

1 Where are the hard participatory budget instances?

Recall the problem presented in class: we must select a subset of participatory budget proposals from the set $J = \{1, \dots, n\}$ of all proposals. Set J is partitioned into sets $J_1, \dots, J_\ell$; each set J_k contains the proposals relative to neighbourhood k. Each proposal $j \in J$ has received $p_j \in \mathbb{N}$ votes, costs $w_j \in \mathbb{N}$ to implement, and positively affects $h_j \in \mathbb{N}$ citizens. The objective of the problem is to select a subset of proposals such that:

- The number of votes received by the selected proposals is as large as possible.
- The overall budget limit $c \in \mathbb{N}$ is not exceeded.
- The selected proposals in each neighbourhood k affect at least $\underline{h}_k$ and at most $\bar{h}_k$ citizens.

An integer model for this problem uses variables $x_j \in \{0, 1\}$ for each $j \in J$; x_j takes value 1 if and only if proposal j is selected. The model reads as follows:

$$\begin{aligned}
\max \quad & \sum_{j \in J} p_j x_j && && (1)\\
\text{subject to} \quad & \sum_{j \in J} w_j x_j \leq c && && (2)\\
& \sum_{j \in J_k} h_j x_j \geq \underline{h}_k && \forall k \in \{1, \dots, \ell\} && (3)\\
& \sum_{j \in J_k} h_j x_j \leq \bar{h}_k && \forall k \in \{1, \dots, \ell\} && (4)\\
& x_j \in \{0, 1\} && \forall j \in J. && (5)
\end{aligned}$$

We would like to test the "limits" of the above formulation. If you implement this formulation and generate a few instances, you will realise that Gurobi finds the optimal solution in a few seconds in most cases. Can we generate more challenging instances? And, in particular, can we do so without taking the "easy path" of simply increasing n to a very large number? In other words, for a given fixed upper bound on n (say, $n \leq 500$) can we find a method to generate the other problem parameters (w_j, p_j, h_j, c, $\underline{h}_k$, $\bar{h}_k$) in a way that the resulting instances are, on average, hard for Gurobi to solve?

Your objective is to devise various instance generation methods, generate many instances with each method, and test the proposed formulation on each instance. You should report if any instance generation method produces harder instances, ensuring the differences are statistically significant.

Remark that the instance generation methods should have a stochastic component. For example, you could say, "We take the w_j's uniformly at random between 1 and 50," etc. This way, it will be possible to generate many instances by re-running the method with a different random seed. Your method can also use "hyperparameters". For example, you could say, "We take each $p_j = \lfloor \alpha \cdot w_j \rfloor$, where α is a parameter that varies in $\{0.75, 1, 1.25\}$". Then, you can show whether there is a relationship between the instance hardness and the values of the parameters you use for each instance generation method.

Also, remember that your instance generation procedure must not necessarily guarantee that the instances are feasible. In this case, expand your report by analysing how often your method produces infeasible instances and check whether this metric correlates with the difficulty metric.

2 The Travelling Salesman Problem for Courier Services

Consider an extension of the Travelling Salesman Problem for courier services. Courier services can be contracted to pick up a parcel at a given location and then deliver it to another location. The service uses a vehicle that starts and ends its tour at the depot. Along its tour, the vehicle will visit all the pickup and delivery locations, making sure that:

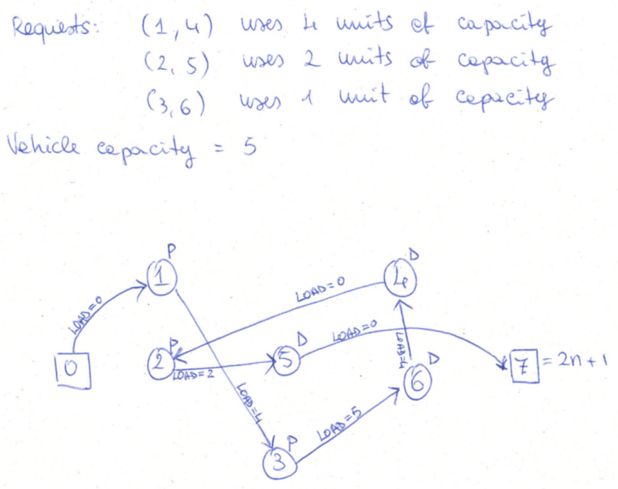
- Each pickup location is visited before the corresponding delivery location, i.e., you cannot deliver a parcel at its destination if you didn't first pick it up at its origin.
- The vehicle leaves and returns to the depot empty.
- The vehicle capacity is never exceeded.
- The tour cost is minimal.

The input to this problem is a directed graph $G = (V, A)$. The vertex set is composed by a depot 0, a set of pickup points $\{1, \ldots, n\}$, and a set of delivery points $\{n+1, \ldots, 2n\}$. For modelling purposes, it is convenient to model the depot twice: once as a departing depot 0 and once as a returning depot $2n+1$. Therefore, the complete vertex set is $V = \{0, 1, \ldots n, n+1, \ldots, 2n, 2n+1\}$ and a tour that starts and ends at the depot is represented by a path starting at 0 and ending at $2n+1$. The two vertices 0 and $2n+1$ represent the same physical location and are distinct only for modelling convenience.

Set A contains three arc types:

- Arcs of type $(0, i)$ for $i \in \{1, \ldots n\}$. These are arcs the vehicle can use when leaving the depot. Because each parcel must be picked up before it can be delivered, the first vertex visited by the vehicle must be a pickup vertex.
- Arcs of type $(n+i, 2n+1)$ for $i \in \{1, \ldots, n\}$. These are arcs the vehicle can use when returning to the depot. Because the vehicle must return empty, the last visited vertex must be a delivery vertex.
- Arcs of type (i, j) for $i, j \in \{1, \ldots, 2n\}$ with $i \neq j$ (because we travel between distinct locations) and $i \neq n+j$ (because otherwise this arc would visit delivery location $n+j$ before pickup location j).

We call each pair of type $(i, n+i)$ a request. The following figure shows a small instance with three requests and a feasible solution. Pickup vertices are denoted with a small P, while delivery vertices have a small D. Arrows are annotated with the quantity onboard the vehicle along each leg of the tour. Remark that each delivery location is visited after the corresponding pickup location, but not necessarily *immediately* after.


Requests: (1, 4) uses 4 units of capacity
(2, 5) uses 2 units of capacity
(3, 6) uses 1 unit of capacity
Vehicle capacity = 5

Let $Q \in \mathbb{N}^+$ be the vehicle capacity. Each request $(i, n+i)$ consumes $d_i \in \mathbb{N}^+$ units of capacity. For convenience, we define parameter d for all vertices in the following way:

$$\begin{aligned} d_{n+i} &= -d_i \\ d_0 &= 0 \\ d_{2n+i} &= 0. \end{aligned}$$

The cost of traversing an arc $(i, j) \in A$ is $c_{ij} \geq 0$. As usual, we denote the set of out-neighbours and in-neighbours of $i \in V$ as

$$\begin{aligned} \delta^+(i) &= \{j \in V \ : \ (i,j) \in A\} \\ \delta^-(i) &= \{j \in V \ : \ (j,i) \in A\}. \end{aligned}$$

A possible model for this problem uses two sets of variables. Binary variables $x_{ij} \in \{0, 1\}$ for each arc $(i, j) \in A$ take the value one if and only if the vehicle uses this arc. Integer variables $y_{ij} \in \mathbb{N}$ for each arc $(i, j) \in A$ keep track of the capacity used onboard the vehicle when it traverses arc (i, j). They take the value zero if the vehicle does not use arc (i, j). The model reads as follows.

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} && && (6) \\
\text{subject to} \quad & \sum_{j \in \delta^+(i)} x_{ij} = 1 && \forall i \in \{0, \ldots, 2n\} && (7) \\
& \sum_{j \in \delta^-(i)} x_{ji} = 1 && \forall i \in \{1, \ldots, 2n+1\} && (8) \\
& y_{ij} \leq Q x_{ij} && \forall (i,j) \in A && (9) \\
& \sum_{j \in \delta^+(i)} y_{ij} - \sum_{j \in \delta^-(i)} y_{ji} = d_i && \forall i \in \{1, \ldots, 2n\} && (10) \\
& \sum_{j \in \delta^+(0)} y_{0j} = 0 && && (11) \\
& \sum_{\substack{j \in S \\ (i,j) \in A}} x_{ij} \geq 1 && \forall i \in \{1, \ldots, n\},\ \forall S \subsetneq V \ : \ i \notin S \text{ and } n+i \in S && (12) \\
& \sum_{\substack{j \in S \\ (n+i,j) \in A}} x_{n+i,j} \geq 1 && \forall i \in \{1, \ldots, n\},\ \forall S \subsetneq V \ : \ n+i \notin S \text{ and } 2n+1 \in S && (13) \\
& x_{ij} \in \{0,1\} && \forall (i,j) \in A && (14) \\
& y_{ij} \in \mathbb{N} && \forall (i,j) \in A. && (15)
\end{aligned}$$

The objective function (6) minimises the total tour cost. Constraints (7) and (8) are the same as in the TSP, and ensure that the vehicle arrives and leaves from each location once, except for the departing depot (the vehicle only leaves it) and the returning depot (the vehicle only arrives at it). Constraint (9) ensures that vehicle capacity is respected and that $y_{ij} = 0$ when arc (i, j) is not used. Constraint (10) sets variables y to the correct value every time they pick up or deliver some parcel. Constraint (11) ensures that the vehicle is empty when it leaves the depot. Constraint (12) makes sure that each delivery location is visited after the corresponding pickup location. Similarly, constraint (13) ensures that the returning depot is visited after all the delivery locations.

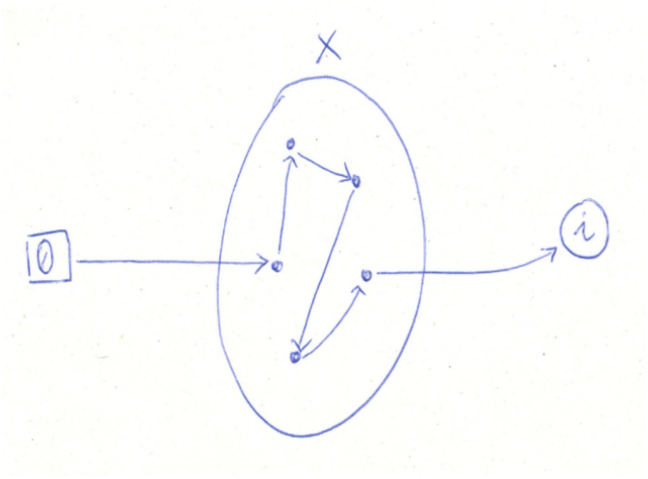
Constraints (7), (8), (12) and (13), taken together, also imply that no subtour are present in any feasible solution. Similar to the TSP's SECs, there is an exponential number of constraints (12) and (13).

Implement a branch-and-cut algorithm to solve the proposed formulation. You should initially remove all constraints of type (12) and (13). These constraints should be separated on the fly via Gurobi callbacks. You should separate them on integer and fractional solutions alike. Remark that both constraints can be separated by solving several max-flow problems; therefore, you can adapt the TSP separation procedure we have seen in class.

3 Dynamic Programming for the Travelling Salesman Problem

Consider an instance of the Travelling Salesman Problem defined on a directed graph $G = (V', A)$. Assume that costs $c_{ij} \geq 0$ for each $(i, j) \in A$ satisfy the triangle inequality. Let $0 \in V'$ be the depot and $V = \{1, \ldots, n\}$ be the customers. Consider any set $X \subset V$ and vertex $i \in V' \setminus X$. We denote with $\mathcal{H}(X, i)$ the cost of the shortest path that

- Starts at the depot $0 \in V'$;
- Visits all the vertices of X once;
- Ends at vertex i.



Remark that the cost of an optimal TSP solution can be obtained by computing $\mathcal{H}(V, 0)$. We also remark that function $\mathcal{H}$ can be trivially computed when $X = \emptyset$:

$$\mathcal{H}(\emptyset, i) = \begin{cases} c_{0i} & \text{if } i \neq 0 \\ 0 & \text{if } i = 0. \end{cases}$$

The above expression states that the least-cost way of going from 0 to i without visiting any other vertex is c_{0i} (which is true because of the triangle inequality). It also sets to 0 the cost of an "empty path" from the depot to the depot without visiting any other vertex.

Finally, we give the recursion relation that allows computing $\mathcal{H}(X, i)$ in terms of values of the function $\mathcal{H}$ already computed for *smaller* sets than X, i.e., for proper subsets of X:

$$\mathcal{H}(X, i) = \min_{j \in X} \big\{ \mathcal{H}(X \setminus \{j\}, j) + c_{ji} \big\}.$$

This relation states that $\mathcal{H}(X, i)$ can be found by identifying the optimal exit vertex j, i.e., the last vertex visited before leaving X bound to i.

Implement the above Dynamic Programming algorithm for the TSP. Generate a sufficient number of random TSP instances and perform a computational comparison between this algorithm and the branch-and-cut algorithms we developed in class. Report the results of your comparison.

4 The Shortest Path Problem

Consider a directed graph $G = (V, A)$. Given an arc $a = (i, j) \in A$, denote with $a^- = i$ the arc's origin and with $a^+ = j$ its destination. As usual, given a vertex $i \in V$, we use the following notation:

$$\begin{aligned}\delta^+(i) &= \{j \in V \ : \ (i,j) \in A\} \\ \delta^-(i) &= \{j \in V \ : \ (j,i) \in A\}.\end{aligned}$$

A path P in G is a sequence of arcs $(a_1, \ldots, a_\ell)$ (with $a_k \in A \ \forall k \in \{1, \ldots, \ell\}$) such that $a_k^+ = a_{k+1}^-$ for all $k \in \{1, \ldots, \ell - 1\}$. We call a_1^- the origin and a_ℓ^+ the destination of P. A path P is called a cycle if $a_\ell^+ = a_1^-$. Graph G is called acyclic if it does not allow any cycles.

We will focus our attention on directed acyclic graphs (DAGs). Let $G = (V, A)$ be a DAG and let $s \in V$ be a source vertex and $t \in V \setminus \{s\}$ a sink vertex. Assume that $\delta^-(s) = \delta^+(t) = \emptyset$. Let $c_{ij} \geq 0$ be a cost associated with each arc $(i, j) \in A$. Given a path P, its cost c_P is the sum of the costs of the arcs making up path P.

The Shortest Path Problem (SPP) asks to find a minimal-cost path from s to t. An integer programme for this problem uses binary variables $x_{ij} \in \{0, 1\}$ for each arc $(i, j) \in A$ with the following meaning:

$$x_{ij} = \begin{cases} 1 & \text{if arc } (i,j) \text{ is part of the minimal-cost path from } s \text{ to } t \\ 0 & \text{otherwise.} \end{cases}$$

The formulation reads as follows.

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in A} c_{ij} x_{ij} && && (16)\\
\text{subject to} \quad & \sum_{j \in \delta^+(s)} x_{sj} = 1 && && (17)\\
& \sum_{i \in \delta^-(t)} x_{it} = 1 && && (18)\\
& \sum_{j \in \delta^+(i)} x_{ij} = \sum_{j \in \delta^-(i)} x_{ji} && \forall i \in V \setminus \{s, t\} && (19)\\
& \sum_{j \in \delta^+(i)} x_{ij} \leq 1 && \forall i \in V && (20)\\
& x_{ij} \in \{0, 1\} && \forall (i,j) \in A. && (21)
\end{aligned}$$

Now consider a generalisation of the (SPP) in which the graph G can contain cycles, and the costs can be negative ($c_{ij} \in \mathbb{R}$ for all $(i, j) \in A$). Formulation (16)–(21) is no longer valid. Show an example of a graph in which the above formulation produces an optimal solution that is *not* a path from source to sink. (Hint: devise a graph containing a negative cost cycle.)

We might think to extend formulation (16)–(21) with subtour elimination constraints (SECs) similar to those we devise for the Travelling Salesman Problem (TSP):

$$\sum_{i \in S} \sum_{\substack{j \in V \setminus S \\ (i,j) \in A}} x_{ij} \geq 1 \quad \forall S \subset V \ : \ s \in S \text{ and } t \in V \setminus S. \tag{22}$$

Constraint (22) is an "obvious" adaptation of the TSP SEC, but it is wrong (as it often happens when something looks too obvious). Show an example of a graph in which constraint (22) does not prevent a subtour from forming. A correct SEC for the SPP is the following:

$$\sum_{i \in S} \sum_{\substack{j \in S \\ (i,j) \in A}} x_{ij} \leq |S| - 1 \quad \forall S \subset V \setminus \{s, t\} \text{ and } |S| \geq 2. \tag{23}$$

The above SECs use sets S that contain neither the source nor the sink and have a size of at least 2 (otherwise, there is no arc all internal to the set). Remark that (23) are similar to the alternative version of the TSP SECs.

Write a branch-and-cut algorithm to solve formulation (16)–(21), (23). You shall devise a separation procedure for (23) that works on integer and fractional solutions. Remark: Unlike the TSP, the SPP does not require all vertices of G to be "visited". Test your procedure on complete graphs (that, therefore, contain cycles) and include graphs with negative-cost cycles in your test set.