Resolution of Google *HashCode2014* by Integer Linear Programming

BY EO & XW

2015-02-02

1 Problem fomulation

1.1 Input data

- Graph: $G(V,\, E)$, $|E| = M$, $|V| = N$, in the statement, E is considered as not oriented, here we re-define E as the *oriented* edges ($|E| = 2M|$), for one-way roads, we just set the corresponding time of reverse direction edge to ∞;
- $\forall e = (u,v) \in E$, $\exists t_e, l_e \geqslant 0$: time and length of each road, ($t_e = \infty$ if this road is one-way);
- T: time limit for each car;
- v_{start}: Vertex that a car starts.
- n_{car}: Number of available cars

1.2 LP variables

- $\forall v \in V$, define $\boldsymbol{f_e} \in \{0,1\}$, indicating if v is the last vertex that the car stops;
- $\forall e \in E$, define $\boldsymbol{x_e} \in \{0,1\}$, indicating if this road is taken;
- $\forall e \in E$, define $\boldsymbol{y_e} \in \mathbb{N}_+$, indicating the number of times of passing this (oriented) edge.

1.3 LP formulation

So the LP problem *for one car* can be formulated as:

$$\underset{\substack{x_e \in \{0,1\} \\ y_e \in \mathbb{N}_+ \\ f_v \in \{0,1\}}}{\text{Maximize}} \sum_{e \in E} x_e l_e\,, \quad s.t. \begin{cases} \forall e = (u,v) \in E, \quad x_e \leqslant y_e & (C1) \\ \forall e = (u,v) \in E, \quad x_{(u,v)} + x_{(v,u)} \leqslant 1 & (C2) \\ \sum_{e \in E} y_e t_e \leqslant T & (C3) \\ \forall v_i \in V, v_i \neq v_{\text{start}}, \quad \sum_{\substack{e \in E, \\ e = (u,v_i)}} y_e = \sum_{\substack{e' \in E, \\ e' = (v_i,u)}} y_{e'} + f_{v_i} & (C4) \\ f_{v_{\text{start}}} + \sum_{\substack{e \in E, \\ e = (v_{\text{start}},v)}} y_e - \sum_{\substack{e' \in E, \\ e' = (v,v_{\text{start}})}} y_{e'} = 1 & (C4\text{bis}) \\ \sum_{v \in V} f_v = 1 & (C5) \end{cases}$$

Some comments on this formulation:

- There is no constraint to guarantee the *unicity of the connected component* for 2 reasons:

1. This constraint is very expensive.
2. If our path is very *dense* in the graph, the probability of having more than one connected component is very low.

- Explaination of the constraints:

– $(C1)$ says that our path has to cross a street at least once to take picture of this street.

– $(C2)$ says if you cross a two way street both direction it only counts once.

– $(C3)$ says that our path should be finished within the time limit T.

– $(C4)$ and $(C4\text{bis})$ assert that we are creating a path where no car teleports.

– $(C5)$ says that our path has a unique end.

- To get 8 paths for 8 cars, we run the above model by setting the time limit $T = 8\,T_{\text{given}}$. We can then split the solution into 8 parts to get one path for each car (see section 2.2 below). This trick also guarantees that the path should be very dense, thus there's no subtours in the solution.

1.4 LP resolution

The size of our model is very big, to solve this integer LP problem, we used the NEOS site (*http://www.neos-server.org/neos/*), and choose *Gubori* as the solver, this enables us to solve our mixed integer LP problem quite quickly (~ 3 minutes for our model input).

2 Solution processing

Once we get a solution from the formulation above (i.e. the variables x, y, f), we need to process this solution to get a *path* for each car.

2.1 Turn the y_e into a big path

We use a greedy algorithm to get a path from y_e :

Algorithm 1

i. Turn the y_e into a graph where the edge e is copied y_e times.

ii. Follow a random path from the start to the end, destroying every edge we use right after using it.

iii. If this path contains all the edges: return this path;
else: look for an intersection point v between our path and the remaining edges.

iv. Follow another path (loop) starting from this point v, and add this loop into the middle of the path, then go to *iii*.

The construction of the y_e guarantees that this algorithm finishes (as we predicted theres is only one connected component) and returns a proper path.

2.2 Cut the big path into n_{car} paths

The result of **Algorithm 1** will give us a big path (let's call this path $\boldsymbol{P}_0$) for one car, starting from v_{start} (Google Paris HQ) and going through the streets of Paris, with time limit equal to $8T$. To get 8 paths that starts from v_{start} and with time limit equal to T, we do the following:

- We run the first car (stars from v_{start}), the car follows $\boldsymbol{P}_0$ as long as the time limit T is not surpassed, until it stops at vertex v_{next}.
- The next car first finds a shortest(in terms of time cost) path from v_{start} to v_{next}, when it reaches v_{next}, again it follows $\boldsymbol{P}_0$, until it run out of time, and stops at another vertex, we set v_{next} to this vertex, and re-do the same procedure for all the cars left.

To get the shortest path, we can run the Dijkstra algorithm starting from v_{start}. Although going from v_{start} to v_{next} will cost some time, this time loss is quite small, and leads to a small loss of total length visited.

3 Improvements

3.1 Optimizing time usage in LP model

As the path from v_{start} to v_{next} will lead to some time loss, we want to optimize the time usage for the big path $\boldsymbol{P}_0$, so that we get sufficient time left for paths from v_{start} to v_{next}.

This can be done by introducing the time usage in the objecticve function of our model, instead of maximizing only length of visited streets ($\sum\limits_{e\in E} x_e l_e$), we can *add the lefted time as objective*:

$$\underset{\substack{x_e\in\{0,1\}\\ y_e\in\mathbb{N}_+\\ f_v\in\{0,1\}}}{\text{Maximize}} \sum_{e\in E} x_e l_e + \varepsilon\left(T - \sum_{e\in E} y_e t_e\right)$$

Here ε is a very small number (for example, $\varepsilon = 0.00000001$), to guarantee that the LP will optimize the length in the first place.

With this trick, the time left for big path $\boldsymbol{P}_0$ is 6411, which is largely sufficient for our cars to go from v_{start} to v_{next}.

After this improvement, our score is *1967437*, which is only 7 meters from the total street length in Paris, whereas there are still 2229 seconds left for the last car. Our solution from LP did not include this last street of 7 meters because Gurobi will not stop *exactly* at optimum, but at some point very close to optimum.

So we can manually find the last unvisited street (which is from vertex 10872 to vertex 2962), and perform a shortest path from the stopping point of last car to vertex 10872, then visited this last street. Thus we can visited *all* the streets in Paris with our 8 cars !

3.2 Chang objective: minimize time used

Now that we are sure that with $T = 8T_{\text{given}}$, we *can* visite all Paris, so we can change the model: just make the origional objective (street length) *as a constraint*, and this time minimize the *time* used.

And as we are sure that all street can be visited, we no longer need the variables x_e (which is a indicater of whether the street e is taken). So the new LP formulation is:

$$\underset{\substack{y_e \in \mathbb{N}_+ \\ f_v \in \{0,1\}}}{\text{Minimize}} \sum_{e \in E} y_e l_e \,, \quad s.t. \begin{cases} \forall e = (u,v) \in E, \quad y_{(u,v)} + y_{(v,u)} \geqslant 1 & (C1) \\ \sum_{e \in E} y_e t_e \leqslant T & (C2) \\ \forall v_i \in V, v_i \neq v_{\text{start}}, \quad \sum_{\substack{e \in E, \\ e = (u, v_i)}} y_e = \sum_{\substack{e' \in E, \\ e' = (v_i, u)}} y_{e'} + f_{v_i} & (C3) \\ f_{v_{\text{start}}} + \sum_{\substack{e \in E, \\ e = (v_{\text{start}}, v)}} y_e - \sum_{\substack{e' \in E, \\ e' = (v, v_{\text{start}})}} y_{e'} = 1 & (C3\text{bis}) \\ \sum_{v \in V} f_v = 1 & (C4) \end{cases}$$

Here we changed the objective, and the constraint (C1) means we impose that all streets of Paris should be visited.

The solution of this LP problem is 418589 (the total time is $8T_{\text{given}} = 432000$), which means that we not only have visited all Paris, but also saved *a lot* of time. With this improvement, the last car has *9596* seconds' time left !

The result of our solution is shown in figure 1.

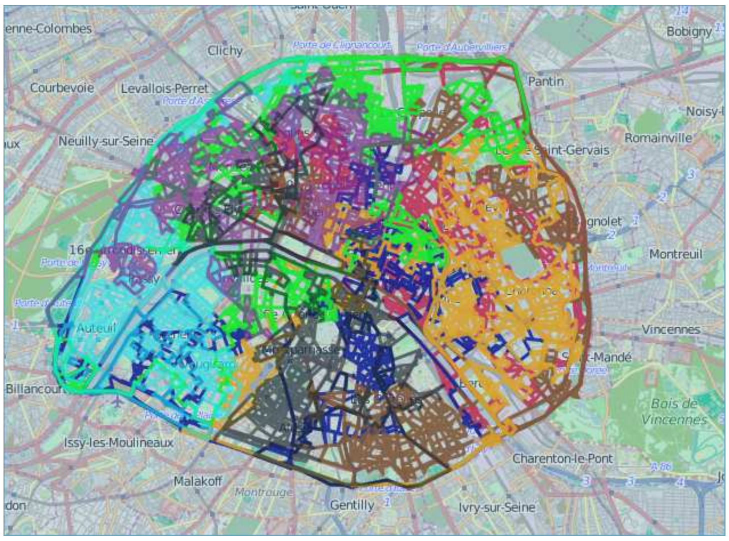

Figure 1. The paths of 8 cars in our solution

4 File descriptions

Here is a bref description of the files:

- The root folder

The root folder contains *final_ solution.gpx* & *final_ solution.txt*: this is our final solution (after the second improvement).

- The *LP_ model* folder

This folder contains the code for the LP formulation.

- *PL.mod*: this is our first model as described at the beginning.
- *PL_ time_ reg.mod*: this is the model for the first improvement.
- *PL_ min_ time.mod*: this is the model for the second improvement.
- subfolder *gen_ mod*: contains code for generating the parameters (l_e and t_e) in our model.

- The *get_ solution* folder

- *get_ bigpath.py*: read the LP solution (the $\{y_e\}$), then convert it into the big path $\boldsymbol{P}_0$ using the algorithm 1.
- *get_ 8car_ solution.py*: code to convert the big path $\boldsymbol{P}_0$ into 8 paths, outputs the file ready for submission.
- subfolder *data*: contains some LP solutions and other files.