r-2013-01-24

František Hajnovič
ferohajnovic@gmail.com

February 27, 2013

Contents

1 EA Oracle based on betweenness centrality

1.1 First version

The idea for this algorithm stems from the observation, that most of the shortest paths in a graph pass through a small set of nodes with high betweenness centrality measure.

Let us consider betweenness centrality (BC) measure of nodes in a graph. This is a measure that specifies the node's importance in terms of the number of shortest paths that pass through the node.

Arguably, pre-computing shortest paths between nodes with a high BC would speed up subsequent queries - we could do a bidirectional Dijkstra search from required pair of vertices and once settling a high-BC node from each side, look up in a table the rest of the shortest path. The idea is similar to Transit nodes () algorithm and similarly, we would have to check if a query is not "local", that is, if the two Dijkstra searches did not already collide and found a shortest path even before settling a high-BC node.

In a timetable scenario, we can do something similar. We will make use of the following observation. Let us define the cost of an arc in the underlying graph to be the length of a quickest elementary connection serving that arc. Then, in timetables, often the optimal connection goes along the way of the shortest path in the underlying graph. There are exceptions to this of course, notably in a case, when following the shortest path would mean lot of fast connection, but with lot of waiting as well, while the optimum is to choose longer connections without much changes. Thus obeying the shortest route from the underlying graph is not always the best strategy. The solution to this could however be to try out also next sub-optimal shortest paths. A good criterion for trying out also these paths would be to measure the waiting time when following optimal path - in case it is too long, try out

also other shortest paths. We might also try out minimum-hop paths - though these are not bound to be those with the smallest number of changes (thus somehow minimizing the waiting time as well), they prefer connections with fewer stops and longer traversal times between them. Also, they could take us via a larger city in a scenario, where EA between a pair of nearby smaller villages is searched for (which is often the quickest connection in reality).

This approach can still fail because of the way we assigned costs to the arcs of the UG. What if there is an extremely fast elementary connection on an arc, that goes through it only once per day? In such case, the shortest path would (misleadingly) lead us through this arc, however, catching the quick connection would be infeasible (or require lot of waiting). However:

- It rarely happens in practice that one elementary connection between a pair of stops would be considerably faster then another one. It could happen if:
  - We merge timetables of different types - e.g. airlines with buses. Then an airplane going from Bruxelles to Paris would be much faster then a bus driving the distance (*without stop!*). It is not very common, however, for a slower transportation vehicle to go without stop the same distance as a several times faster transportation vehicle.
  - E.g. we have a TGV and an ordinary train serving the same elementary connection. This is not a very common situation too, for the same reasons as stated above.
- We could mitigate this problem by assigning costs to the UG's arcs as an average of the lengths of the connection serving the arc.

Another problem is the scarcity of the connections that go along the shortest path. What if the shortest path leads e.g. through a desert where buses operate only once in a long time and a better option would be to choose a longer itinerary with more frequent connections? The problem is that by taking shortest path in the UG, we ignore the (possibly wild) time-dependent nature of the timetable. Therefore, we could perform a random sampling of best connection between high-BC cities in a TE graph first, extracting the underlying shortest paths from the connections (and sorting them in by the number of times they appeared). In the case that the true shortest path in UG is served by frequent connections, we will probably find it (if we don't, we will simply get it from the UG), as well as other good alternatives.

So we may combine shortest (and sub-optimal) paths from UG with those obtained by random sampling to get a set of shortest paths to try out when querying for a EA between two high-BC cities. Still, it may not be enough to find a true EA - this would happen in a situation when travelling via shortest UG paths would mean too much waiting *and* the best path to follow is infrequent (and thus we have not found it in the random sampling).

An interesting question is how many shortest paths in UG the connections from A to B use - this is actually a measure of the timetable's regularity.

There are two more problems:

1. The high-BC nodes might actually form a small central part of a graph, which itself is e.g. a separator separating otherwise large components. Then the "local" Dijkstra searches would explore considerable part of the graph before arriving at a high-BC node. Moreover, pre-computation of shortest paths for nodes close together would not help the matters too much.
2. Bidirectional search is not a possibility in a time-dependent scenario. We could however pre-compute shortest UG paths and sample paths from each high-BC node to each *other* node, increasing the time complexity of pre-processing and the space complexity of the oracle as well, but eliminating this problem.

It would be thus convenient to have a small set of "access" nodes that would be close enough to every node from the UG graph. OK, maybe the access nodes should not be those with high BC.

Frequency characteristics of a path - its weakest link determines it.

How to find sub-optimal paths?

2 Neural network approach

- multilayer perceptron, output paths
- multilayer perceptron, output EA
- multilayer perceptron, output "should I stay or should I go?"
- recurrent network

3 New introduction

New name - Oracle based approach to solving of earliest arrival problem in timetables

4 Data analysis

Following is an analysis of the data. Results were obtained by running *ttblazer* application.

UG

Basic			
Connectivity **Paths** **Highway dimension**	**Degrees**		**Betweenness**
Name	**Type**	**# nodes**	**# arcs**
Normal	**# of comp.**	**Largest**	**Strong**
# of comp	**Largest**	**Max length**	**Avg length**
Max size	**Avg size**	**Max degree**	**Avg degree**
Avg betw.	**HD**		
ug_air01.ug	US domestic flights (Jan/2008)	287	4668
ug_cpru.ug	Regional bus	877	2416
ug_cpza.ug	Regional bus	1128	2778
ug_london.ug	Underground rails	321	732
ug_montr.ug	Montreal public transport	313	349
ug_svk.ug	Road network of SVK	181386	425829
ug_zsr.ug	Country-wide rails	233	588

Table 1: Underlying graphs properties

TT

Name	Type	# el. conn.	Load time
tt_cpza	Regional bus	61747	4.451s
tt_cpru	Regional bus	38540	2.275s
tt_zsr	Country-wide rails	932052	66.6668s

Table 2: Timetables - main properties

TE

Name	Type	# nodes	# arcs
te_cpza	Regional bus	59276	119857
te_cpru	Regional bus	36458	74070
te_zsr	Country-wide rails	1706077	2637896

Table 3: Time-expanded graphs - main properties

TD

Name	Type	# nodes	# arcs
td_cpza	Regional bus	1108	2839
td_cpru	Regional bus	871	2487
td_zsr	Country-wide rails	233	588

Table 4: Time-dependent graphs - main properties

5 Open points

- What to focus on during X-mas?

6 To do

Theory:

- Properties propagation in step a), step b)...

Practice:

- United airlines extract data, California road netw., public transport - buses
- Analyse properties: HD, separator, betweenness, scale-free distr., planarity, avg. distance, degrees for TE
- Building timetables from UG
- Machine learning