CS 584 Project 1 Report

Nathan Collins
(with Arthur Peters)

05 May 2011

1 Introduction

We compute minimal-weight routes for number-of-segments and number-of-reversals weightings using Dijkstra's single-source-shortest-path algorithm on a graph *derived* from the railway network. For a railway network of P nodes and S segments, our derived graph has $O(P)$ vertices and edges. Using a priority-queue based Dijkstra implementation, our algorithms run in $O(P \log P)$.

2 Algorithm

We reduce both minimal-weight routing problems to a shortest-path search on directed graph derived from the railway network, using different weightings for each algorithm. Computing minimal-weight routes from the railway network as given is difficult for at least two reasons:

1. the constraints on paths, due to switches, are not captured in the graph structure of the network,

2. the direction of travel on a segment, and direction of the train, are not captured in the graph structure of the network.

So, we define a derived graph, based on the railway network, which captures (1) and (2) in its structure.

2.1 Derived Graph

We now describe the derived graph, and some diagrams are attached at the end of this document. The code is a straight-forward implementation of the following defining equations, so we do not include pseudo-code as such.

Let $R = (P, S)$ be the railway network, with nodes (endpoints and switches) P and edges (segments) S. Let $G = (N, E)$ be the derived graph, which we will now describe. The nodes N are of the form

$$(s, (a, b), d)$$

for $s \in S$, $a, b \in P$, and $d \in \{F, B\}$, s.t. a and b are the endpoints of segment s in R. Such a node corresponds to the train being on segment s, traveling from a towards b, with the front of the train facing b iff $d = F$ (mnemonic: (F)orward, (B)ackwards).

The derived graph G is directed, so $E \subseteq N \times N$, and the edges in E correspond to reversals and transitions on R. Let $o : S \to P \to P$ compute the "other end" of a given segment, i.e.

$$o(s, a) = b \quad \text{and} \quad o(s, b) = a$$

iff $s \in S$ with endpoints $a, b \in P$. Then the edges E are as follows:

1. for each segment s from a to b in R, there are "reversal" edges corresponding to stopping the train and reversing direction. Namely, for all d we have

   $$(s, (a, b), d) \mapsto (s, (b, a), \bar{d}),$$

   where $\bar{F} = B$ and $\bar{B} = F$.

2. for each switch $n \in P$, with trunk t and branches ℓ and r, there are edges corresponding to traveling from t to ℓ or r, and traveling from ℓ or r to t. Namely, for all d we have

   $$(t, (o(t, n), n), d) \mapsto (b, (n, o(b, n)), d)$$

   for $b \in \{\ell, r\}$, corresponding to going from the trunk to a branch, and we have symmetric edges corresponding to going from a branch to the trunk.

NOTE: that there are no edges connecting two distinct branches of the same switch.

To ensure the derived graph G corresponds to *valid* transitions in R, we need to remove self loops, since the reversal edges ((1) above) would otherwise allow the train to "flip over" on the loops! The loops are removed in a preprocessing step. Namely, for each switch where both branches are equal, or one branch is also the trunk, we add a "phantom" point and a "phantom" segment to break the loop. For example, if `n:t;r,r` is a switch, then we replace it with `n:t;r',r` and add endpoints `n':r` and `n':r'`, where `n'` and `r'` are chosen fresh. Then, when deriving G as described above, we additionally add edges corresponding to traveling from `r` to `r'`, and vice-versa.

Now, paths in G correspond to *valid* transition sequences in the railway network, and hence the routing problems are reduced to a shortest-path search, once the appropriate edge weightings have been defined.

2.2 Weightings

There are two edge weightings of interest:

1. *number-of-reversals* (NOR): where transitions between segments in R is free, but reversals of travel direction costs 1.

2. *number-of-segments* (NOS): where transitions between segments in R costs 1, but reversals of travel direction are free.

So, $NOR, NOS : E \to \{0, 1\}$ are defined by

$$NOR(e) = 1 - NOS(e)$$

and

$$NOR((s1, (a1, b1), d1) \mapsto (s1, (a2, b2), d2)) = 1 \text{ if } d1 \neq d2 \text{ and } 0 \text{ otherwise.}$$

That is, $NOR(e)$ is 1 exactly when the direction the train relative the direction of travel changes across the edge e.

Now, we need to be careful not to count travel along the phantom segments, since they aren't present in the "real" network, so the actual definition of NOS is a little more complicated. Namely, $NOS(e)$ is 1 when $NOR(e)$ is 0 *and* e does *not* include a phantom segment.

2.3 Shortest Path Search

Once G and the weightings are defined, the routing problems are reduced to shortest path searches on G. Because the assignment only asks for single paths between fixed end points, we use a single-source shortest-path algorithm, namely Dijkstra's.

2.4 Graphing (Aside)

Although not required by the assignment, and so not described here, we implemented rendering of arbitrary railyway networks, derived graphs, and paths, using Graphviz. This was an aid to debugging, and produced pleasing pictures.

3 Basic Steps Accounting

As argued in the next section, the time complexity is bounded by $O(P \log P)$. Assuming the underlying search algorithm is a merge sort, we also get $\Omega(P \log P)$. Hence the best and worst case complexities are $\Theta(P \log P)$, and so the number of basic steps is *proportional* to

$$P \log P = 98 \log 98 \approx 449$$

for the given input datafile `testdata.txt` which has 98 points.

It occurs to me that this might not be what you had in mind. However, in order to make our program compute a more accurate value here, we would have to do away with all library routines used, since they don't report basic steps. We could have augmented Dijkstra's and derivation of G to produce some number, but this number would not be more meaningful than the "449" above, since we are programming in a pure language and rely on many $O(\log n)$ mappings.

4 Correctness and Complexity

Because we reduce the problem to a shortest path problem on the derived graph G, and then use a well-known shortest-path algorithm, we get correctness of routing from correctness of G. The time complexity is then the complexity of constructing G, plus the complexity of the search, and both turn out to be $O(P \log P)$.

4.1 Correctness of the Derived Graph G

The correctness of G is clear by construction: the nodes in G are segments in R annotated with direction of travel and relative train direction, and edges in G correspond to *valid* transitions between segments in R and reversals of travel direction on segments.

4.2 Complexity

Our implementation of Dijkstra's uses a balanced-binary-tree based priority queue, with all operations $O(\log n)$, and so our path search has time bounded by $O((E + N) \log N)$. We now argue that $E \in O(P)$: the railway network R has valence bounded by 3, and so $S \in O(P)$. Our derived N has at most 4 nodes for each segment of S, corresponding to all possible train travel and direction combinations, and so $N \in O(S)$. Finally, our derived G has valence bounded by 5, because there are at most two ways in and two ways out through switches, plus a reversal, and so $E \in O(N)$. Hence $E \in O(P)$ by transitivity, and so $O((E + N) \log N) = O(P \log P)$.

It remains to analyze the complexity of building G, which is also $O(P \log P)$. The "phantomization" is done in a linear pass over R's description in $O(P + S)$. We then build up a mapping from segments to their endpoints in $O(E+P \log P)$, by grouping and sorting pairs in $P \times S$ corresponding to the description of R. The segment mapping is used to build N in a linear pass, in $O(N \log S)$, the log factor coming from map lookups. Finally, we build G in a linear pass over S and N in time $O(E \log S)$, again with a log factor due to map lookups. As above, since $E \in O(P)$, all this work reduces to $O(P \log P)$.