Volake rozpravy

František Hajnovič
Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava, Slovakia
ferohajnovic@gmail.com

April 2012

1 Methods

Methods for time-dependent routing:

- Time-dependent SHARC

2 Notes from M.K. thesis

- Boost library: for command line options, test cases,
- CMake: for making the code
- Doxygen: for documentation

3 Some notes

3.0.1 "Wild" timetables can ruin structure

Imagine a simple graph formed by n vertices connected in a line. For such a graph, shortest path routing is trivial - just head in a direction of the target vertex. However, provided that two conditions hold, it is easy to imagine a timetable on top of such a graph which suddenly makes it extremely difficult to compute shortest connection in such a timetable. The two conditions are the following:

- we allow **express lines**, i.e. connections that can pass multiple nodes (and edges) without stopping at any of them. This does not mean new edges in the underlying graph, rather it epitomizes vehicles of different speed.
- we allow **overtaking**, i.e. such connections, that we can find among them pairs of connections (A, B) such that, A starts sooner or at the same time as B but B arrives earlier to the destination.

Provided these two conditions, we can now basically build *any* timetable graph on top of the underlying graph, which will have nothing in common with the underlying map except for the number of vertices.

Figure 1: Underlying graph with extremely easy structure and easy route-planning possibilities

Unfortunately, both express lines and overtaking are happening in real world situations (e.g. in a public transportation including both buses and subway system).

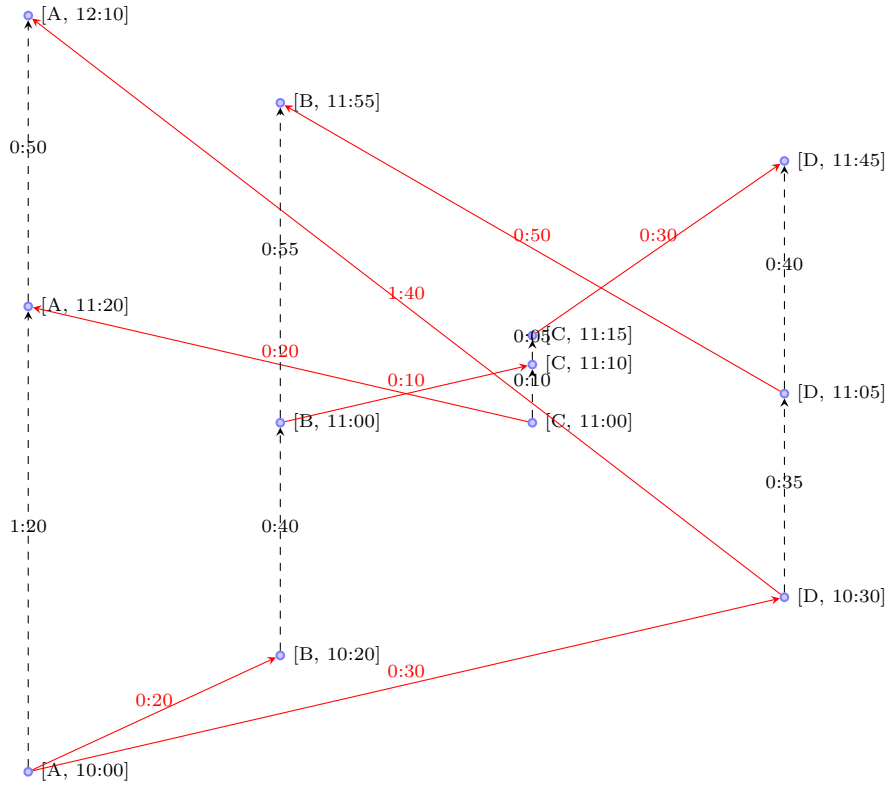

Figure 2: An example timetable graph built on top of the shown underlying graph in picture 1. Express lines and overtaking have ruined the regular structure.

4 Preprocessing time of Gavoille's algorithm for planar graphs

In planar graphs, we can find a separator of size $\mathcal{O}(\sqrt{n})$ ([Eri] in linear time ($\mathcal{O}(n)$, [GPPR04]). Moreover, after splitting the graph by removing this separator, the created components are of size no more than 2/3 of the original size (their size is $\leq 2/3n$). Thus, Gavoille's algorithm [GPPR04] can be applied to create an efficient distance labeling scheme.

The algorithm's preprocessing phase (we include just the points that require nontrivial running time) proceeds as follows:

1. A required separator of size $\mathcal{O}(\sqrt{n})$ is first found in $\mathcal{O}(n)$ time
   - $\mathcal{O}(n)$
2. Shortest paths are computed between all the pairs of vertices inside the separator, e.g. by Floyd-Warshall algorithm with time complexity $\mathcal{O}(n^3)$. The separator is, however, at most $\mathcal{O}(\sqrt{n})$ large, thus the required time for this step is $\mathcal{O}(n^{3/2})$
   - $\mathcal{O}(n^{3/2})$
3. From all the boundary nodes of the separator (those leading out to some component(s)), we compute a shortest path tree. We can do this with Dijkstra's algorithm, whose best version runs in time $\mathcal{O}(m + n \log n)$. As the average degree of any planar graph is < 6, $m = \mathcal{O}(n)$ and the bound can be improved to $\mathcal{O}(n \log n)$ for planar graphs. We potentially run an instance of Dijkstra's algorithm from each vertex of the separator, so this step takes at most $\sqrt{n}(\frac{2}{3}n + \sqrt{n}) \log(\frac{2}{3}n + \sqrt{n}) = \mathcal{O}(n^{3/2} \log n)$.
   - $\mathcal{O}(n^{3/2} \log n)$

So we see that the resulting running time is $\mathcal{O}(n^{3/2} \log n)$, with n being the size of the graph at the beginning of the procedure. The algorithm then proceeds recursively, that is, the procedure described is run for the created components again, until trivial components of size 1 are found (after at most $\mathcal{O}(\log n)$ steps). The running time therefore depends on the size of the created components. As we would like to make an upper bound, we need to know the worst case - what sizes do the components have to have in order to maximize remaining running time?

If we look at the function $f(n) = \mathcal{O}(n^{3/2} \log n)$, we see that already from $n \geq 2$ the function increases faster than a linear function, thus $f(2n) > 2f(n)$ from that point on. That means, that one large component would increase the running time more than more smaller components. Thus we can bound the time complexity of the whole algorithm by assuming that every round of the recursion the largest possible components would be created (each with the size at most $2/3n$). This leads to the final estimate $f(n) + f(\frac{2}{3}n) + f((\frac{2}{3})^2 n) + ... = \mathcal{O}(f(n)) = \mathcal{O}(n^{3/2} \log n)$.

References

[Eri] Jeff Erickson. Graph separators.

[GPPR04] Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz. Distance labeling in graphs. *Journal of Algorithms*, 53(1):85 – 112, 2004. ISSN 0196-6774. URL `http://www.sciencedirect.com/science/article/pii/S0196677404000884`.