Flow-Based Guidebook Routing

Hannah Bast * Sabine Storandt*

Abstract

Public-transportation route-planning systems typically work as follows. The user specifies a source and a target location, as well as a departure time. The system then returns one or more optimal trips at or after that departure time. In this paper, we consider *guidebook routing*, where the goal is to provide time-independent answers that are valid over long periods of time. An example answer could be: Take Bus 10 to the main station, from there take Tram 11 or 13 (whichever comes next) to your target station. Trip duration: 30 minutes. Frequency: every 20 minutes. Valid: weekdays from 6am - 8pm. We show how to compute such guidebook routes efficiently and with provably good quality. An evaluation on real-world data shows that few guidebook routes usually suffice for good coverage. We also show how guidebook routing can be used to speed up transfer patterns, a state-of-the-art method for public transportation routing.

1 Introduction

When planning a trip with a public-transport information system, the standard setting is to specify a source location and a target location, and a departure time and date. The route planner then outputs one or more trips departing at or after the specified time, and which are in some way optimal. Typical optimization criteria are total travel time and number of transfers, and a typical scenario is to output all Pareto-optimal solutions with respect to these criteria.

In this paper we consider *guidebook routing*, where the user specifies only source and target location, but neither day nor time of departure. A typical answer is then of the kind:

From source station A, take bus number 19 or 52 (whichever comes first) to station X, and from there take bus number 6 to the target station B, Weekdays from 6am to 8pm, every 15 minutes.

There might also be alternatives, for example:

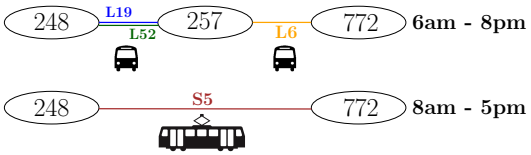

Figure 1: Exemplary guidebook routes.

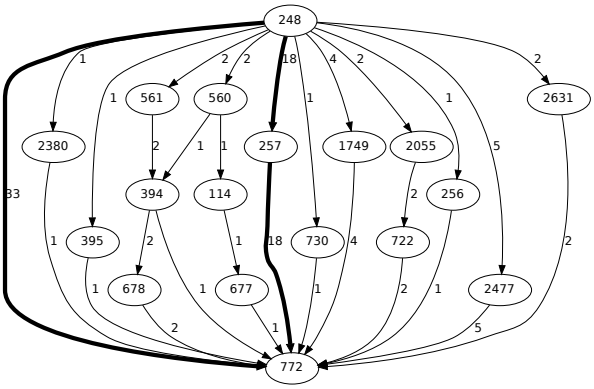

Figure 2: Variety of journeys in the transit network of Austin from one station to another (inner nodes indicate stations where a new vehicle is boarded or walking starts) for a whole day. Edge values indicate the number of departure times for which the trip was included in the solution. Only bold edges occur in a large percentage of queries; the remaining variations increase the graph size remarkably, though.

Take the direct subway S5 from A to B, Weekdays 8am to 5pm, every 20 minutes.

Figure 1 provides an illustration of these two answers. For travel times outside the service intervals, the user may ask for more alternatives. This approach has several advantages.

The first advantage is that this is a simpler and more intuitive user interface in several respects. There is no need to specify the departure time, which simplifies asking the query. In return, the answer is more informative, because we get information not only for a particular time but for a whole (and typically large)

*Institut für Informatik, Albert-Ludwigs-Universität Freiburg, 79110 Freiburg, Germany, {bast,storandt}@informatik.uni-freiburg.de

time interval. Note that this is something which users familiar with a particular transit (sub-)network do in their head anyway. It is also a common way to provide transit directions in travel guides, hence the name "guidebook routing". Also note the built-in robustness against delays or minor schedule changes. If a connection has a sufficiently high frequency, say every 10 minutes, it does not really matter if a vehicle comes a few minutes earlier or later. I simply take the next one that comes.

A second, less obvious advantage is the following. In complex transit[1] networks, specification of the exact departure time often gives a confusing variety of optimal solutions. For example, it may happen that if you increase the departure time by a single minute, a completely different connection (involving different lines and different transfer stations) becomes optimal, but only insignificantly so compared to the previous connection (e.g. one minute faster). This solution is optimal in the theoretical sense, but still more confusing than helpful to a user; see Figure 2 for a more complex example from a real dataset. This kind of artificial diversity becomes a real problem in a fully multi-modal setting, that is, when considering more than two criteria. Guidebook routing could provide an elegant alternative to cut down on the number of solutions suggested to the user.

A third, even less obvious advantage is that guidebook-style queries have the potential to be processed significantly more efficiently than traditional queries, where the exact departure time is specified. In Section 4, we will review transfer patterns, a state-of-the-art method for route planning in large transit networks, which is also currently in use at Google Maps. In a nutshell, transfer patterns precompute a compact representation of the sequences of transfers of all optimal paths between all pairs of stations. This is intimately related to guidebook routing in the sense that it asks only for the sequences of transfers of paths. We will see that this intimate connection can be exploited to improve precomputation space and query time of transfer patterns routing.

1.1 Related Work We are not aware of any previous work in the computer science literature on the particular problem of guidebook routing, as we described it in our introduction. Recently, the feature has started to appear in a few web-based route planners, including: Rome2rio [2], GoEuro[3], and Google Transit [4]. None of these have published the algorithm behind that feature. No statements on the quality or coverage of the result is made (see our Section 6.2 for an explicit definition of coverage and a respective evaluation).

A prerequisite to our computation of guidebook routes is the set of all Pareto-optimal solutions between two given stations at all times; so-called *profile queries*. In the introduction, we already described transfer patterns [2], which, conceptually, pre-computes the sequences of transfers of all optimal paths between all pairs of stations at all times. An alternative approach is RAPTOR [4], which computes these solution sets in order of increasing number of transfers. Transfer patterns are pre-computation heavy (about 800h for the complete transit network of the New York metropolitan area), but provide fast query times also for very large networks (about 8ms for New York, 12ms for North America). RAPTOR uses no pre-processing and achieves good query times on metropolitan-size networks (100ms on London). In this paper, we concentrate on the transfer pattern approach, not only because it allows for better query times, but also because of the close relation between transfer patterns and guidebook routes, as we will explore in Section 4.

One pleasant side effect of guidebook routing, explained in the introduction, is that it reduces unwanted diversity in the result sets. There have been other approaches with that goal or side effect. In [8], multiple criteria are linearly combined to a single objective value. In [3], a notion of fuzzy domination of multi-criteria labels is introduced, and labels are ranked by their aggregated dominance of other labels in the result set. In [1], explicit types of desired and undesired connection types are introduced, which are then used to reduce result sets to reasonable sizes. In [11], routes are preferred that are robust against delays or other small schedule changes. As explained in our introduction, we also get a certain amount of robustness as a side effect. However, the primary goal of guidebook routing is to provide the user with information that is as compact and time-independent as possible.

1.2 Contribution In this paper we describe algorithms to extract guidebook routes efficiently and in a reasonable manner. For that purpose, we first introduce naive approaches and show their limitations for real-world instances. We then present a more sophisticated approach, which is based on constructing a flow network on the solution set. We then evaluate our method (and compare it to the naive approaches) on real-world data. It turns out that our guidebooks route sets are indeed small, stable and cover the optimal solutions for a large percentage of queries.

[1] We will use the terms "transit" and "public transportation" synonymously in this paper

[2] http://www.rome2rio.com

[3] http://www.goeuro.de

[4] http://transit.google.com

2 Preliminaries

In public transit route planning we distinguish between station-to-station (s-to-s) and location-to-location (L-to-L) queries. For s-to-s queries, the source and target location is a transit station. For L-to-L queries these can be arbitrary locations, and solutions involve walking or driving from the source location to the first transit station on the trip and then getting from the last transit station on the trip to the target location. Of course, our ultimate goal is to answer L-to-L queries. But as a first step we will investigate s-to-s queries. We then come back to L-to-L queries at the end of the paper. Therefore, without further specification, queries always mean s-to-s queries in the following.

The first step towards guidebook routing is to identify the set of all (Pareto-)optimal solutions for a certain period of time. A straightforward approach is to first compute the set of all departure events at a station and then start a Pareto-Dijkstra for every departure. A more sophisticated approach is to compute all these results at once with a single Dijkstra computation, referred to in the following as profile Pareto-Dijkstra. In this computation, labels are extended by information on the departure time, and during the Dijkstra computation labels are pruned if they do not improve solution with a later departure time.

Such profile Pareto-Dijkstras play a central role in the pre-computation of transfer patterns [2]. We briefly recapitulate this pre-computation and how the results are used at query time. Conceptually, one profile Pareto-Dijkstra is executed for each transit station, and for each the computation is run until all nodes in the graph are settled. Then the optimal paths are backtracked and the sequence of transfer stations, called transfer pattern (TP), is extracted from each optimal path. All these transfer patterns are stored in a directed acyclic graph (DAG), one per station. At query time, all transfer patterns between the source and target stations (there may be several source stations and several target stations) are extracted from these pre-computed graphs, and then overlaid to form the so-called query graph. Arcs in this query graph correspond to direct connections (trips involving only a single vehicle = no transfers). Direct-connection queries can be evaluated very efficiently (on the order of a few microseconds per query) using an adequate data structure; see [2] for details.

Running a profile Pareto-Dijkstra for every station costs time at least quadratic in the number of stations. Similarly, the size of all the corresponding transfer patterns would be quadratic in the number of stations. This is infeasible already for moderately-sized transit networks. To remedy this, a fraction of all stations (typically around 1%) is selected as hubs. Intuitively, these are stations, where many optimal trips transfer. Now profile Pareto-Dijkstras are run only from these hub stations. For a constant fraction f of stations selected as hubs the complexity is still quadratic, but now multiplied with f. For a value of $f = 0.01$, this is feasible (though still costly) also for continent-sized transit networks. For all non-hub stations, the profile Pareto-Dijkstras are run only until all paths are covered by a hub station. Further , the search is pruned when a path has not yet reached a hub station after three transfers. This simple heuristic very effectively prunes the search space, yet misses only very few optimal queries. Using hub stations, the query graph also contains all transfer patterns from the source stations to the next hub stations, as well as from all these hub stations to the target stations.

In this paper, we investigate constructing guidebook routes both from the result of a profile Pareto-Dijkstra, as well as from pre-computed transfer patterns (without and with hub stations).

For computing actual guidebook routes from the set of all possible solutions, we construct a flow network from this set. A flow network is a graph with a distinguished source and sink vertex and capacities assigned to the edges. A classical problem on such a flow network is to compute the maximum flow from source to sink, that does not exceed any capacity and fulfils the flow conservation constraint. This constraint demands that for every node, except for the source and the sink, the sum of the amount of flow coming in is equal to the sum of the amount of flow going out. The now famous Edmonds-Karp algorithm [6] solved the maximum-flow problem in time $\mathcal{O}(nm^2)$, where n is the number of nodes, and m is the number of edges in the network. The recent approach of [9], solves the problem in time $\mathcal{O}(mn)$. For our guidebook routes, we solve a variant of this problem, asking for a single path with maximum flow; see Section 3.2.

3 Computing Guidebook Routes

A *guidebook route* (GBR) is a time-independent description of a number of journeys in the given transit network. See Figure 1, where each GBR is described as a sequence of transfer stations with line numbers. The goal is to approximate the set of all optimal journeys over the day (or some other time period) by a small number of GBRs. Intuitively, therefore, a good GBR should cover a large number of optimal journeys (at different departure times). We refer to the number of optimal journeys covered by a GBR as its *frequency*. It appears reasonable to look for the GBRs with the largest frequency. We refer to this approach as *pattern count-*

ing. In the following we discuss side effects and variants of this approach. We then describe a more sophisticated approach based on extracting-maximum flow paths in a suitable network.

3.1 Pattern Counting If only the selection of a single GBR is allowed, a natural criterion is to pick the one with the largest frequency. Analogously, if only the selection of k GBRs is allowed, a natural criterion is to pick those with the k largest frequencies. This is both simple and efficient. However, in real-world transit network, we often have groups of very similar patterns, each of which individually have a relatively low frequency, yet the sum of these frequencies is large. An example is given in Figure 3 (left side). In such cases, no single pattern from the group is a good representative. Rather, all the patterns in the group can be considered variations of a single base pattern, which would make for a good representative. One possible approach here would be to allow something that could be called "fuzzy" counting. Namely, we could define a similarity measure between patterns and then count similar patterns as one. But this introduces new problems. Most notably, a pair-wise similarity measure is typically not transitive (if route A is similar to route B, and route B is similar to route C, that does not necessarily mean that A is similar to C). Also it seems hard to come up with a similarity measure that is not based on some (somewhat arbitrarily chosen) threshold.

3.2 Flow-Based Guidebook Routes We have seen that the simple pattern-counting approach has problems with groups of similar patterns, where each individual pattern has relatively low frequency, yet the sum of these frequencies is large. Instead of counting individual patterns, it therefore seems more reasonable to attempt to extract frequent underlying base patterns. Note that, in an extreme case, such an approach might find a base pattern with a large number of journeys with a very similar pattern, yet there is not a single journey with exactly that pattern. An example for such a base pattern is given in Figure 3 (right side).

Our basic approach to compute such GBRs consists of constructing a single condensed graph from the Pareto-optimal patterns. The nodes of this graph are all the (transfer) stations from the patterns. For each pattern $s_1 \ldots s_k$, there is an arc from station s_i to s_{i+1} in the graph, for $i = 1, ..., k-1$. Note that this way many patterns can (and in practice often will) contribute the same arc. We interpret these arc multiplicities as edge capacities $cap : E \to \mathbb{N}$. We then ask for a maximum flow path from the source station to the target station in this graph. This problem is also known as the bottleneck

Algorithm 1 Pseudo-code for retrieving the maximum flow path P in a graph $G(V, E)$ from a source node $s \in V$ to a sink node $t \in V$.

```
flow(v) ← 0   ∀v ∈ V
flow(s) ← ∞
pred(v) ← NULL   ∀v ∈ V
PQ.push(flow(s), s)
while !PQ.empty do
  v ← PQ.top
  if v == t then
    break
  end if
  PQ.pop
  for e = (v, w) ∈ E do
    if flow(w) < min{flow(v), cap(e)} then
      flow(w) ← min{flow(v), cap(e)}
      pred(w) ← e
      PQ.push(flow(w), w)
    end if
  end for
end while

P ← ∅
node ← w
while pred(node)! = NULL do
  P.push(pred(node))
  node ← pred(node).source
end while
return P
```

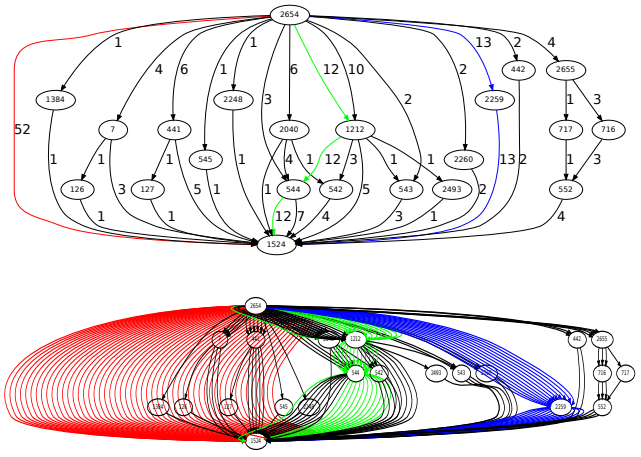

Figure 4: Top-3 max-flow paths (red 52, blue 13, green 12) in a condensed graph. The upper image shows edge multiplicities by labels, the lower one is an expanded version to better illustrate the flow rates.

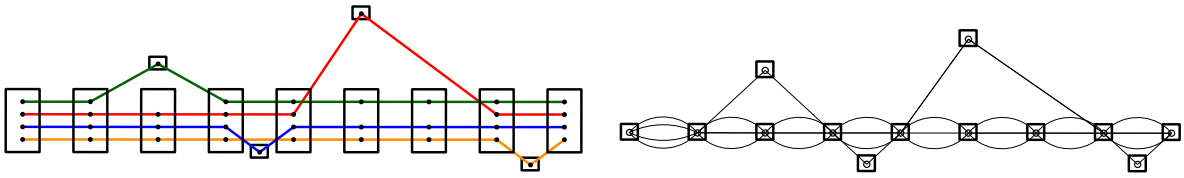

Figure 3: Pattern counting versus flow-based guidebook route extraction: In the left image, four different routes from source to target are depicted (only transfer stations are depicted). Choosing any of them as GBR, the other routes differ at least by two stations from the GBR. In the right the condensed graphs that the straight line route might be a better GBR.

shortest path problem [7].

Note the difference to the classical maximum-flow problem, as described in Section 1.1. We here want to retrieve a single path with large flow, which should then serve as a GBR. In the classical maximum-flow problem, the optimal flow can be (and typically is) distributed over many paths from the source to the sink. Also note that computing the path with maximum flow is much easier than computing the (distributed) maximum flow. Namely, it can be computed using a variant of Dijkstra's algorithm, where the cost of the path is the minimum of the costs of the edges on the path, and the objective is to maximize path cost. The running time of this algorithm is $\mathcal{O}(n \log n + m)$, where n is the number of nodes and m is the number of arcs in the flow graph. See Algorithm 1 for the details.

In practice, these pattern graphs rarely have cycles. In that case, we can sort the nodes of the graph topologically. Dijkstra can then process the nodes in that order, and does not need a priority queue. The running time then becomes linear.

We already mentioned above that, theoretically, it can occur that the GBR corresponding to the max-flow path might not correspond to a single valid journey in the original transit network. However, in practice this is very unlikely to happen. If one is unable to use a direct connection which is part of the suggested GBR, it means the last service of this connection lies already in the past. As GBRs preferably include trips which are served with high frequency, this seems to be an improbable scenario – especially for early departure times.

On the other hand, using max-flow paths mitigates the effect of small pattern distortions over the day in a clean way without the necessity to introduce any parameters. It can also be computed very efficiently (especially if the graph is a DAG) and allows for selecting the top-k routes by subtracting the flow value of the max-flow path from the capacities of all contained edges, and repeating the algorithm in this modified network (see Figure 4 for an illustration of the result for $k = 3$). Also it allows for a more compact storage of the pattern information, as we are only interested in local information and therefore can get rid of any overhead induced by remembering which direct connection belongs to which global pattern (e.g. in Figure 3 the condensed network only requires a third of the nodes needed to describe all global patterns on the left). Moreover this representation makes the flow graph very similar to the query graph used in the transfer pattern approach, which will come in handy for interactive use as we explore in the next section.

4 From Transfer Patterns to Guidebook Routes and Back

The guidebook route principle and the transfer pattern approach exhibit several common characteristics: Both use a time-independent representation of possible journeys in a graph, while ignoring connection information on a global level. Using transfer patterns, the time-independence is paid with maintaining all routes that are optimal at some point in time, while for guidebook routing we aim for keeping only a subset of the journeys or edges in the query graph respectively. Hence on the one hand transfer pattern graphs might be a good starting point to compute GBRs, and on the other hand if we are convinced that the selected GBRs are sufficient for all departure times, we might want to reduce the transfer pattern query graph to contain only them – hopefully decreasing the space consumption and query time later on. In the following we will describe these two ways of interaction between transfer patterns and guidebook routes in more detail.

4.1 Transfer Patterns as Basis for Guidebook Route Extraction

Of course, GBRs can be computed on the fly by running a profile Pareto-Dijkstra for the desired time interval, backtracking all optimal paths, constructing the condensed flow graph and then searching for maximum-flow paths until the set of original solutions is sufficiently covered. But this approach is much too slow for interactive use (as already a single Pareto-Dijkstra might be). Hence applying some sort

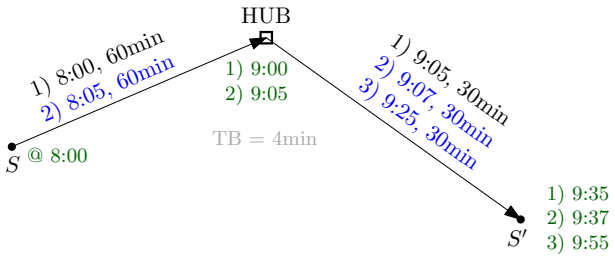

Figure 6: Possible connection from source station S to target station S', departing at 8:00, via a hub station. Blue journeys would be valid GBRs if up to 5 minutes increase in the arrival is declared ok. Assuming a transfer buffer (TB) of 4 minutes, using option 2) for the first part only allows to use option 3) for going from the hub to the target. This results in a 20minute overhead compared to the optimal route, i.e. the concatenation is not a valid GBR. However, considering only GBRs from the hub to S' would be feasible, because this would increase the arrival time only by 2 minutes when using option 2) instead of 1) for this part.

of preprocessing seems worthwhile. The auxiliary data created for the transfer pattern approach almost contains everything we need for our flow-based GBR extraction – only the multiplicities of direct connection aka the edge capacities are missing. But naturally this information is computed anyway in the transfer pattern creation process, so we can just store these numbers along. This also does not interfere with the fact, that the query graphs are not stored explicitly, but are composed from subgraphs on query times. Here, multiple occurring edges are simply joined and their capacities get summed up. In the resulting graph, we can compute max-flow paths just as described before and evaluate only them for a certain departure time instead of the whole query graph.

4.2 Guidebook Routes and Reduced Transfer Patterns On TP construction time automatically all information necessary for our GBR extraction algorithm are available, as they both rely on the output of a profile Pareto-Dijkstra run. Hence GBR computation could be used as an additional preprocessing step for TP. So we build the condensed graph and extract max-flow paths repeatedly until all original solution of the Pareto-Dijkstra are covered by the respective GBRs (for some predefined cover criterion). Then we only consider the set of GBRs as basis for the TP graph (see Figure 5 for a real-world example). Note, that this process does not work automatically as soon as hub stations are selected. Even if the GBRs cover all solutions from the source node to the hub, and also from the hub to the target, their concatenation might not cover all original solutions. For example, lets say a solution is covered if by going by guidebook it exists an alternative solution which takes not more than 5 minutes of additional travel time. Now, going from the source to the hub using a GBR which lets you arrive 5 minutes later at the hub, you might miss the connection to the target which you would have get if travelling optimally. So you eventually have to wait 20 minutes for the next departure, clearly exceeding your covering budget (see Figure 6 for an illustration). This problem is not new, though. Even in the original TP paper, it was observed that you actually need two runs from the hub stations – one assuming you are already inside a vehicle, one for transferring at the hub – to not miss any solutions. But to reduce preprocessing time, a restriction to only the latter run was tested, and evaluation showed that in practice only a very small percentage of queries was affected by this heuristic approach [2]. In our case, there are also some remedies allowing us to guarantee coverage in total, e.g. the TP to the hubs could be maintained unaffected and then the GBR selection process can be applied only for the runs with a hub as source station; but of course this would limit the power of GBRs. Luckily, our experiments will reveal that 'trusting' our conventional approach also when using hubs works well in practice.

5 Refinements

The flow-based guidebook route extraction is a rather general approach, which can be seen as a basic framework that is customizable in different ways. At first, we discuss options of modifying the flow graph such that certain requirements (as fulfilling certain cover criteria with a small number of max-flow paths/GBRs) are met even better. Then we extend our station-to-station algorithm to more complex location-to-location queries, focusing on constructing the flow graph when several source and target stations are of interest.

5.1 Modifying the Flow Graph Often at daytime connections are more frequent and better clocked than at night. Hence the number of necessary GBRs to cover all optimal solutions should shrink when restricting the time interval to e.g. 8:00-18:00 and handle requests for departure times outside the interval separately. This can be easily implemented by starting the profile Pareto-Dijkstra only for the desired time slot. Besides, we could also tag single GBRs with their valid time interval to pass this information on to the user. This interval can be computed as the period in which optimal solutions are best covered (among all GBRs) when choosing this GBR. Of course, this should be also included in the preprocessing as on the fly computation is too

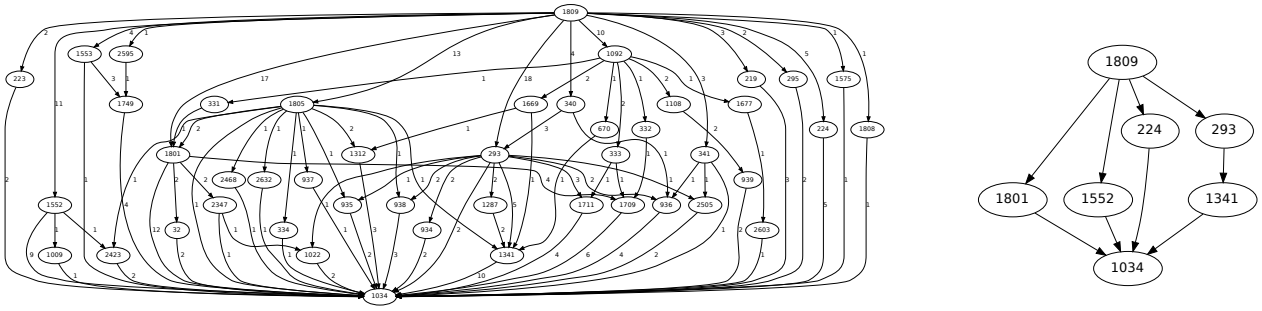

Figure 5: Transfer pattern graph between two stations (in Austin) for a whole week on the left, and the respective guidebook transfer pattern graph on the right.

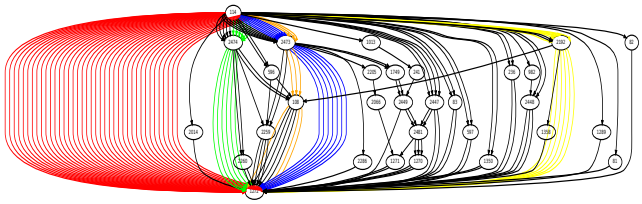

Figure 7: Top-5 flows marked red, green, blue, yellow and orange. The journeys that led to the blue edges are already covered by the GBR induced by the green path. Hence the set of GBRs can be reduced to only four (red, green, yellow, orange) to cover all optimal results.

expensive.

Further we could interleave the GBR extraction and the coverage evaluation by removing edges from the flow graph which belong to already covered solutions. This also decreases the number of necessary GBRs, see Figure 7 for an example. Moreover we could decide to extract the top-k GBRs and then go through the list of Pareto-optimal solutions and simply add everything that is not covered as additional GBR. This can be very helpful in case there are only a few outlier routes with a very high number of transfers and very small travel time, which can not be covered by routes with a smaller transfer value. Also the selection of GBRs with a certain transfer number can be forced artificially by enumerating all candidate paths with the respective hop distance and selecting the one with the maximal flow among them.

5.2 Location-to-Location Queries Answering L-to-L queries changes the guidebook setting, as now several so called access station of the source and the target location become relevant. Access stations are typically defined as the set of stations inside a certain walking or driving radius around the location. We propose two ways to incorporate flow-based GBR extraction in this scenario: In the one-step approach we build a joint flow graph for all optimal routes from a source access station to some target access station. If we connect the source node to all its access stations via an edge with a capacity of ∞ and the same vice versa with the access stations of the target and the target node, we can apply the same max-flow path algorithm as before (see Figure 8 for a small example). While this seems to be the easiest way to extract GBRs for L-to-L queries, it might lead to all GBRs originating in the same source access stations which might be undesirable for getting a diverse output set. Therefore we might switch to a two-step-approach, where we build a different flow graph for each source access station, and get individual max-flow path(s) to target access station(s).

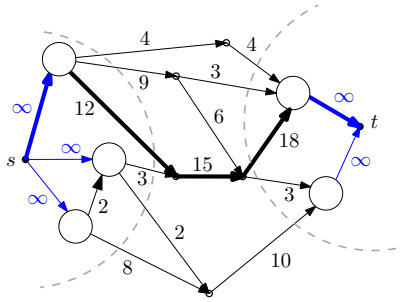

Figure 8: Example of a combined flow graph for source location s and target location t. The dashed grey lines indicate the feasible walking radius, the stations marked with circles are the respective access stations. The max-flow path is printed bold.

6 Evaluation

In this section, we want to measure how useful guidebook routes are for real-world instances. We first give some details on the used graph model and data sets,

then motivate the aspects under which we evaluate GBRs and finally present results on runtime and quality of our proposed algorithms.

feed	# stations	# routes	# trips	time (s)
Austin	2.719	84	21.757	0.12
Texas	11.654	152	21.616	1.08
Toronto	10.891	181	96.384	1.85
Madrid	4.635	219	221.507	3.95

Table 1: Test graphs for our evaluation along with the important features. *time* denotes the runtime of a single Pareto-Dijkstra run in the respective time-dependent graph (averaged over 100 random queries).

6.1 Settings and Data Sets The concept of guidebook routes is independent of the used graph model, of course, but the chosen model affects the runtime of (profile) Pareto-Dijkstra computations. We used the conventional time-dependent model [10] for our experiments, with walking allowed between the stations for an arbitrarily long time (assuming a speed of $5^{km/h}$), and a uniform transfer buffer of five minutes. The used Pareto-criteria are travel time and number of transfers, as in the original transfer pattern paper. The real-world data got extracted from GTFS feeds [5], an overview of the characteristics of our test transit networks can be found in Table 1.
All algorithms were implemented in C++ and runtimes got measured on a single core of an Intel i5-3360M CPU with 2.80GHz and 16GB RAM.

6.2 Performance Metrics The desired characteristics of a set of guidebook routes are to be optimal solution covering, with the single GBRs being stable over long periods of time resulting in a preferably small GBR set. To evaluate coverage, we first have to introduce practical cover criteria.

Definition 1. (Coverage) *Given a set of journeys A and another set B for a departure time, specified each by arrival time (AT) at the target and number of transfers (NT), we say a journey $a \in A$ is covered if it exists a journey $b \in B$ with $NT(b) \leq NT(a)$, and $AT(b) \leq 1.1 \cdot AT(a)$ and $AT(b) \leq AT(a) + 5min$. Moreover we call A covered by B if every element in A is covered.*

So we claim that the number of transfers should not grow at all when going by guidebook, and the arrival time should increase only slightly. Of course, the parameters chosen here are somewhat arbitrary, but we think being suboptimal by at most five minutes reflects the tolerance range of a user. Hence for station-to-station queries, we will evaluate with how many GBRs this criterion can be fulfilled. Moreover, given a fixed number of GBRs, we ask for the already covered intervals (with a time interval $[t, t']$ being covered if no uncovered departure is contained), as well as the number of infeasible departures, where going by guidebook does not allow for reaching the target.

6.3 Experimental Results Firstly, we computed the number of necessary GBRs to cover all solutions (with the above given definition of coverage) between two stations over a day (Wednesday). The respective results can be found along with some features of the test graphs (average number of departures from a station over a day, as well as average number of (distinct) patterns) in Table 2. Obviously, the number of GBRs is much smaller than the number of different patterns, indicating that many routes can be covered by a single GBR. When applying flow graph modifications – namely deleting edges of already covered paths – the number of GBRs dropped on average by 1-2. Iterating over the whole week we got an increase of about 50% in the numbers of GBRs as connections typically differ on weekends (also increasing the Pareto-set of course); using Monday to Friday the increase was smaller and mainly derived from the fact that now long overnight connections were possible.
Note, that computing solutions based on the TP graph reduced to GBRs typically leads to a superset of solutions compared to evaluating the GBRs individually, as also combinations of two or more routes that share a transfer station are possible (just like for normal TP graphs). This effect is not included in the evaluation here, hence maybe an even smaller set of GBRs would suffice when extraction is already performed in the preprocessing step.
For comparison with the Pattern Counting (PC) baseline, we analysed the coverage behaviour in dependency of the number of GBRs as well as the percentage of infeasible departure times. The results for Toronto are summarized in Table 3 (for the other test graphs we observed very similar outcomes). We see that FLOW outperforms PC especially for a small number of GBRs significantly in terms of coverage, while exhibiting a slightly higher number of infeasible departure times – but the latter is remarkably low for both approaches. For a higher number of GBRs the two approaches are comparable in both criteria, as then typically not only many paths with the same frequency but also several

[5] http://code.google.com/p/googletransitdatafeed/wiki/PublicFeeds

	average #			
	departures	patterns	distinct patterns	GBRs (8-18)
Austin	59	79	26	11 (6)
Texas	40	104	27	14 (10)
Toronto	164	328	84	24 (18)
Madrid	1104	1034	163	37 (12)

Table 2: Station-to-station queries for 1000 randomly chosen source/target pairs, evaluated over a day. The number of GBRs indicates how many GBRs were necessary for full coverage, the number in brackets to cover the interval between 8:00 and 18:00.

	# GBRs	1	2	3	4	5	6	7
PC	covered	11.76	24.67	40.08	45.08	59.56	70.88	72.03
	infeasible	0.23	0.05	0.01	0.00	0.00	0.00	0.00
FLOW	covered	37.00	38.13	40.34	53.66	65.80	71.16	75.44
	infeasible	0.45	0.18	0.03	0.01	0.00	0.00	0.00

Table 3: Comparison of pattern counting (PC) and flow-based (FLOW) GBR extraction. *covered* and *infeasible* are given in percent.

	extraction time	coverage
Austin	$3\mu s$	85
Texas	$2\mu s$	76
Toronto	$5\mu s$	74
Madrid	$4\mu s$	81

Table 4: Experimental results for location-to-location queries averaged over 100 randomly chosen source/target location pairs along with departure times. *extraction time* indicates the time to extract the top-8 guidebook routes from a single flow graph between all source/target access stations. *coverage* is the percentage of solutions covered by these guidebook routes.

max-flow paths with the same value are present in the remaining graph and tie breaking sometimes works in favour of the one or the other method. Incorporating the described flow graph modifications in Section 5.1, the superiority of FLOW becomes more noticeable as the coverage value increases by about 5% for the first GBR and over 10% for the remaining ones.

In Figure 9 an example shows the typical development of covered departure intervals (here each 15 minutes) when selecting more and more GBRs on the one side with PC and on the other side with FLOW. We observe that typically journeys in the early morning and late evening are only sparsely covered, but the number of 'good' intervals increases significantly with each additional GBR for both approaches – but faster for the FLOW approach. To measure the impact of hub stations on the quality of our GBRs, we performed the following experiment: We selected a source s and a target station t and a departure time randomly and computed the optimal Pareto-set. Then we selected one station H that was contained in some optimal path (not necessarily a transfer station) and simulated it being a hub. To that end we computed all optimal paths and subsequently GBRs for going from s to H and from H to t. Afterwards we evaluated for all departure times at s if there exists an optimal solution containing H. If this is the case we looked if the concatenation of the GBRs from s to H to t induces a journey that covers this optimal solution. In less than one percent of 100 queries in the public transit network of Toronto (even less for the other test graphs) a solution could not be covered, implying that the presence of hub stations does not compromise the applicability of GBRs in practice.
For location-to-location queries, we measured the solution coverage for random departure times. So we let a Pareto-Dijkstra reveal the optimal set of journeys between two locations (with access stations being selected in a $400m$ radius around the locations), and used the one-step or the two-step approach to compute 8 GBRs in the constructed flow graph(s) for the respective source and target stations. We observed that the results of the two methods differ only slightly: For graphs with a sparse set of stations, most journeys started with the same source station anyway, hence forcing the selection of different ones with the two-step approach is unnecessary. If the stations were rather dense (as in Madrid), the result set contained journeys with varying source stations, nevertheless there were a lot of different solutions with very similar Pareto-costs, hence they could be easily covered. So only in 6% of the queries in Madrid we observed a slightly better coverage with the two-step

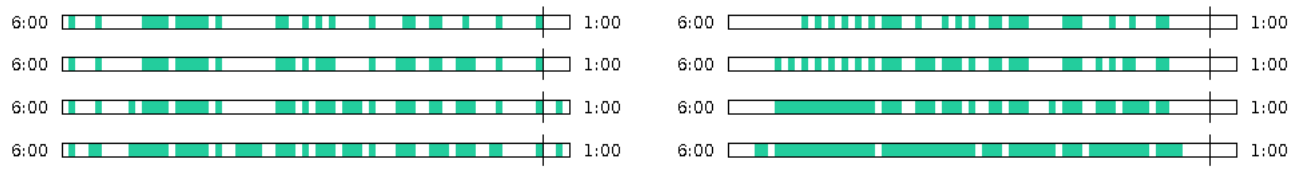

Figure 9: Covered intervals (green) and uncovered intervals (white) with incremental guidebook route extraction for a manually chosen example query. On the left when applying pattern counting, on the right with flow-based computation.

	TP →	GBR (top-5)	→ TP	
	graph size	extraction time	check time	graph size
Austin	42 + 64	$2\mu s$	$1.8s$	11 + 16
Texas	62 + 97	$2\mu s$	$1.5s$	13 + 19
Toronto	71 + 105	$4\mu s$	$4.0s$	21 + 26
Madrid	85 + 142	$3\mu s$	$27.6s$	17 + 21

Table 5: Experimental results for interleaved transfer pattern and guidebook route computation. All values are averaged over 100 randomly chosen source-target station pairs. *graph size* describes the number of nodes + edges of the (reduced) transfer pattern graph (TP).

approach. Hence the computational overhead induced by the two-step approach might not be justifiable. For the one-step method, extraction time and coverage for all our test graphs can be found in Table 4. It shows that the selection of GBRs for station pairs already suffices to answer more complex L-to-L queries in a sound manner.

To enable real-time GBR presentation, TP-based preprocessing is necessary. In Table 5 we first give the original TP query graph sizes (number of nodes plus edges) and subsequently the runtime to extract GBRs, which is in the order of microseconds and clearly allows for extraction on query time. Also we present the time to check if the GBRs already fulfil the cover criterion. Note, that the latter is only performed when the GBR extraction is included in the preprocessing, for direct queries we would just choose the top-k flow-paths (choosing k e.g. as the respective average GBR value in Table 2). Finally, we show the size of the TP query graph when reduced to the set of covering GBRs. Obviously, the number of nodes and edges decreases drastically as a lot of variations are pruned. For Madrid this results in a TP graph with the size about a sixth of the original one. Because the number of edges is proportional to the runtime for evaluating the query graph, this also translates into speed-up.

So all in all, GBRs were shown to be a useful and – with appropriate preprocessing – efficient tool for reducing all optimal journeys over a period of time to a small set of stable routes.

7 Concluding Remarks and Future Work

In this paper, we motivated and evaluated algorithms for computing guidebook routes on the fly and interleaved with transfer patterns. Our experiments show that normally a very small set of guidebook routes suffices to cover all optimal solutions, which is beneficial for memory consumption, query times and producing reasonable output size sets for the users. As a next step, the applicability of guidebook routes for more complex Pareto-criteria (e.g. including ticket prices, taxi duration) should be explored, which requires an extended definition of coverage. Also robustness and other journey characteristics could be considered to decide which guidebook routes to choose.

Another aspect is the representation of guidebook routes: At the moment we store them as sequence of transfer stations in congruency with the transfer pattern approach. But also a very specific representation as sequence of line numbers (Bus No.12, Subway S4) could be useful – or a description by the sequence of transportation modi (walk, bus, walk, train, walk), as it was shown in previous work that such modi restrictions already decrease query times significantly [5].

8 Acknowledgement

This work was partially supported by a Google Focused Research Award on Next-Generation Route Planning.

References

[1] H. Bast, M. Brodesser, and S. Storandt. Result diversity for multi-modal route planning. In *ATMOS*, pages 123–136, 2013.

[2] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. Fast routing in very large public transportation networks using transfer patterns. In *ESA*, pages 290–301. 2010.

[3] D. Delling, J. Dibbelt, T. Pajor, D. Wagner, and R. F. Werneck. Computing and evaluating multimodal journeys. Technical report, KIT Karlsruhe, 2012.

[4] D. Delling, T. Pajor, and R. F. F. Werneck. Round-based public transit routing. In *ALENEX*, volume 12, pages 130–140, 2012.

[5] J. Dibbelt, T. Pajor, and D. Wagner. User-constrained multi-modal route planning. *Networks*, 6:10, 2012.

[6] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM*, 19(2):248–264, 1972.

[7] V. Kaibel and M. Peinhardt. On the bottleneck shortest path problem. Technical report, ZIB Berlin, 2006.

[8] P. Modesti and A. Sciomachen. A utility measure for finding multiobjective shortest paths in urban multimodal transportation networks. *European Journal of Operational Research*, 111(3):495–508, 1998.

[9] J. B. Orlin. Max flows in O(nm) time, or better. In *STOC*, pages 765–774, 2013.

[10] G. Stølting Brodal and R. Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *ENTCS*, 92:3–15, 2004.

[11] B. Strasser. *Delay-Robust Stochastic Routing In Timetable Networks*. Diploma thesis, Karlsruhe Institute of Technologie, 2012.