Shortest Path with Alternatives for Uniform Arrival Times: Algorithms and Experiments

Tim Nonner and Marco Laumanns

IBM Research
{tno, mlm}@zurich.ibm.com

Abstract

The Shortest Path with Alternatives (SPA) policy differs from classical shortest path routing in the following way: instead of providing an exact list of means of transportation to follow, this policy gives such a list for each stop, and the traveler is supposed to pick the first option from this list when waiting at some stop. First, we show that an optimal policy of this type can be computed in polynomial time for uniform arrival times under reasonable assumptions. A similar result was so far only known for Poisson arrival times, which are less realistic for frequency-based public transportation systems. Second, we experimentally evaluate such policies. In this context, our main finding is that SPA policies are surprisingly competitive compared to traditional shortest paths, and moreover yield a significant reduction of waiting times, and therefore improvement of user experience, compared to similar greedy approaches. Specifically, for roughly 25% of considered cases, we could decrease the expected waiting time by at least 20%. To run our experiments, we also describe a tool-chain to derive the necessary information from the popular GTFS-format, therefore allowing the application of SPA policies to a wide range of public transportation systems.

1998 ACM Subject Classification G.2.2, G.1.6, F.2.2

Keywords and phrases Shortest Path, Stochastic Optimization, Public Transportation

Digital Object Identifier 10.4230/OASIcs.ATMOS.2014.15

1 Introduction

Despite the increasing availability of smartphones and real-time information, it is still a common practice, especially in high-frequency public transportation systems, to simply wait for the next arriving suitable bus (or other means of transportation like metros). This holds especially for systems which do not provide exact time-table information, but manage buses instead by the frequency they leave the terminal, e.g. the Dublin bus system[1]. In such a situation, an experienced local traveler should be aware of alternative suitable buses in order to minimize his waiting time by picking the first arriving one. Formally, such a selection process requires to find a trade-off between minimizing the waiting time by selecting a large set of alternatives, and minimizing the consequent travel time by selecting a small set of alternatives with short travel time, in the extreme case the single alternative with shortest travel time. Iterating this process through the whole network leads to an extension of the classical Shortest Path Problem, called *Shortest Path with Alternatives (SPA) Problem*. Datar and Ranade [6] observed that this extension can be solved efficiently in case of Poisson arrival times (with exponentially distributed inter-arrival times) of buses, which makes it practical even for large-scale public transportation systems. In contrast, Nonner showed that

[1] `http://www.dublinbus.ie/en/`

general arrival times result in an NP-hard problem [9], even for one-hop networks. Arguably, for systems where buses run with a given frequency, uniform arrival times (with uniformly distributed inter-arrival times) are the most suitable modelling choice, but they lack the nice properties of a memoryless process. In fact, Boyan and Mitzenmacher [5] showed that optimal policies for such a system have a more complicated structure, which might be hard to communicate: in addition to a list of alternatives for each stop, they require additional timing information for the bus picking process.

First, we show in this paper that an optimal SPA policy for uniform arrival times can be efficiently computed subject to the constraint that it has the structure implied by Poisson arrival times, thus giving a trade-off between providing a simple policy to execute and a realistic time assessment. Second, we run several experiments to illustrate the benefits of SPA policies. In fact, we are not aware of any such study, the only related experimental evaluation was done in the context of data delivery in bus networks [1]. We are interested in comparing the following policies: (P1) a classical single shortest path using an exact timetable, (P2) a SPA policy using a post-processed timetable with frequency information, but where we allow only a single alternative at each stop, and (P3) a SPA policy without this restriction. Comparing policies (P1) and (P2) allows us to reason about how efficient frequency based systems are compared to exact time-tables, and comparing policies (P2) and (P3) gives insights in how much we are able to improve by allowing multiple alternatives in such systems. Note that policy (P2) corresponds to a traveler who navigates greedily through the system, waiting at each stop for the single bus with best combined waiting and travel time.

To run our experiments, we build on the increasingly popular *General Transit Format Specification (GTFS)*[2], which allows us to collect timetable information of multiple European capitals[3][4]: Berlin, Budapest, Dublin, and Oslo. Interestingly, this format allows the specification of frequencies, exactly the information needed for our study. However, probably because the current shortest path computation methods do not benefit from this information, it is hardly ever provided. Even public transportation systems like the one of Dublin, which explicitly mention that their timetables should be interpreted as frequencies rather than exact times, do not make use of this extension. To deal with this lack of available frequency information, we derive it by counting runs-per-hour in standard time-tables, which aligns with the implicit behavior of a sample traveler.

Our main conclusions are: (1) frequency-based systems are not much worse than exact systems, at least on average, and (2) allowing multiple alternatives in a SPA policy provides a significant improvement. Specifically, although the average improvement in total travel time is relatively small, we could decrease the waiting time by at least 20% for roughly 25% of considered cases. Thus, policy (P3) is clearly superior to policy (P2). Hence, we think that providing SPA policies would be a natural extension to any public transportation planner. Another advantage of such policies is that they provide backup opportunities in case there are disruptions in the timetable.

Outline. In Section 2, we formally introduce the SPA Problem and present an efficient method to compute SPA policies for uniform arrival times subject to the constraint that the policy has the simple prefix structure implied by Poisson arrival times. Since we could not

[2] `https://developers.google.com/transit/gtfs/`
[3] `http://dublinked.com/datastore/datasets/dataset-254.php`
[4] `http://www.gtfs-data-exchange.com/`

find a counterexample during extensive experiments, we conjecture that an optimal SPA policy for uniform arrival times satisfies this constraint anyway, but proving this is an open problem. In Section 3, we introduce the popular GTFS-format and explain our approach to derive frequency information from this format. Finally, in Section 4, we describe our experiments, which are then discussed in Section 5.

Related work. Bertsekas and Tsitsiklis [4] discuss the problem of selecting a fixed probability distribution over the outgoing arcs of each node in order to also minimize the expected travel time. But since only a fixed distribution is selected at each node, this problem is more related to the classical shortest path problem. Having different alternatives is also an element in the recently introduced guidebook routing [3], but the focus is here to cover as many optimal routes as possible with a few such guidebook routes. Also Dibbelt et al. [7] consider the case of providing alternatives to recover from failed connections, but as for guidebook routing, it is assumed that a fixed timetable is given (with some random delay on top), whereas we assume from the beginning that stochastic frequencies are given as input. However, we think that algorithms for SPA could be valuable fast heuristics for problems with a fixed timetable and an additional random component. Another interesting problem in the context of stochastic routing is to maximize the probability to arrive on time [8]. But in contrast to SPA, only a single path is considered.

2 Algorithmic Approach

Consider the case of n buses labeled $1, 2, \ldots, n$ in a one-hop network, that is, they all leave the origin stop for the destination stop, possibly with different travel times and arrival patterns at the origin stop. More specifically, let T_i be the (possibly random) travel time of bus i, and let A_i be a random variable that describes the time until the next arrival of bus i at the origin stop, which is the time we have to wait in order to board this bus. The goal is now to select a fixed subset of *alternatives* $\sigma \subseteq \{1, 2, \ldots, n\}$ that minimize the expected combined travel and waiting time subject to the assumption that the traveler will pick the first arriving alternative.

If the A_i are Poisson then simple arithmetic shows that, for any set of alternatives σ, the combined expected waiting and travel time is

$$\frac{1 + \sum_{i\in\sigma} \frac{\mathbb{E}[T_i]}{\mathbb{E}[A_i]}}{\sum_{i\in\sigma} \frac{1}{\mathbb{E}[A_i]}}. \tag{1}$$

Using this, if we assume that the buses are ordered such that $\mathbb{E}[T_1] \leq \mathbb{E}[T_2] \leq \ldots \leq \mathbb{E}[T_n]$, then an optimal solution σ^* has the form $1, 2, \ldots, s$ for some $1 \leq s \leq n$ [6, 9], we say that it *forms a prefix*. Thus, there is only a linear number of possible optimal solutions, which can hence be efficiently enumerated. Even if we add some cardinality constraint k on the size of σ, there are still only $\mathcal{O}(n^2)$ many solutions to enumerate [9]. By applying these facts iteratively in a backward Dijkstra-scheme, it is then possible to compute an optimal SPA policy for a network with an arbitrary number of hops in polynomial time, see [6, 9]. In fact, a large part of the complexity of computing SPA policies is already contained in such one-hop networks.

However, the term in (1) does not describe the combined waiting and travel time for uniform arrival times. For instance, if all A_i are uniformly distributed in $[0, 1]$ and all T_i are 0, then selecting i many buses results in a combined waiting and travel time of $\frac{1}{i+1}$. However, since then $\mathbb{E}[A_i] = \frac{1}{2}$, the term in (1) would yield $\frac{1}{2i}$, which is a lower bound on the real

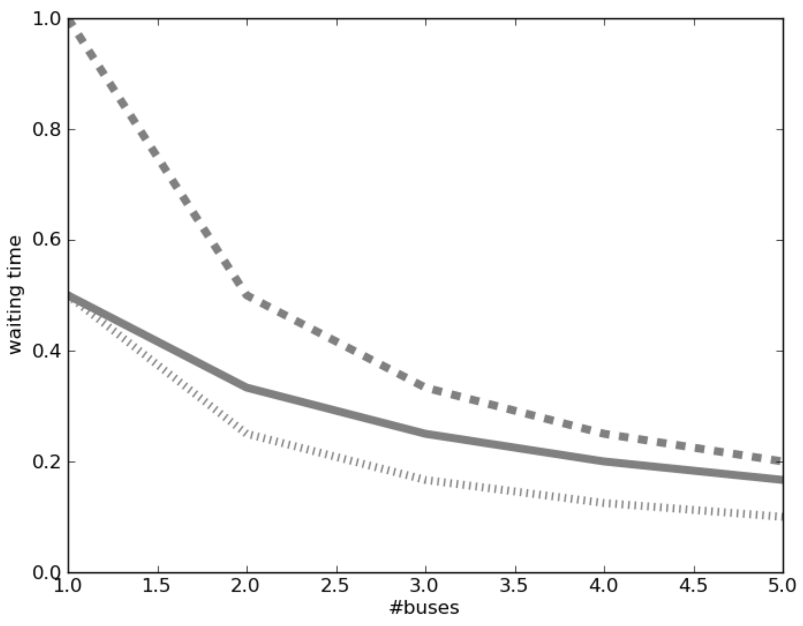

Figure 1 The functions $\frac{1}{i+1} \approx$ uniform (solid line), $\frac{1}{2i} \approx$ Poisson (dotted line), and $\frac{1}{i} \approx$ Poisson with $2\mathbb{E}[A_i]$ (dashed line).

cost. A better approximation for large i is $\frac{1}{i}$, which we receive if we replace $\mathbb{E}[A_i]$ by $2\mathbb{E}[A_i]$ in term (1). Indeed, it has been shown in [9] that this is an arbitrary good approximation for large i under reasonable assumptions, yielding a polynomial-time approximation scheme (PTAS).

Figure 1 illustrates the values of these different objectives in such a simple scenario. This picture shows that using Poisson arrival times as an approximation for uniform arrival times might result in quite a large gap. Specifically, using $\frac{1}{2i}$ as an approximation is exact for $i = 1$, but insufficient for large i. On the other hand, using $\frac{1}{i}$ is a good approximation for large i, but insufficient for $i = 1$. A reasonable heuristic to cover this case is to use $\frac{1}{2i}$ for $i = 1$ and $\frac{1}{i}$ otherwise, which still has a large gap for $i = 2$. Therefore, we decided not to use a heuristic, but to exactly compute the waiting time for uniform arrival times. However, to keep the space of possible policies small, we only consider solutions that form prefixes. This is motivated by the fact that it might be hard to communicate to a traveler that he should not pick a bus with a smaller travel time than a given one. Besides, we conjecture that such policies are optimal for uniform arrival times as well.

We use Algorithm 1 to compute an optimal set of buses for uniform arrival times subject to the constraint that they form a prefix. In this algorithm, we have two DP-arrays, Φ and Π, which we will explain first. For each bus i, let l_i be the value such that A_i is uniformly distributed in $[1, l_i]$. Consider then a prefix of buses $1, 2, \ldots, i$. Clearly, the earliest arriving bus from this set will arrive before time $l_{\min} := \min_{1 \le j \le i} l_j$. The goal is then to fill array Π such that $\Pi[i, j]$ is the probability that from the buses $1, 2, \ldots, i$ exactly j many arrive before time $l_{\min}$. Formally,

$$\Pi[i,j] = \sum_v \left(\prod_{z=1}^{i} p_z^{v_z} \prod_{z=1}^{i} (1-p_z)^{1-v_z} \right),$$

where the sum is over all $\{0,1\}$-vectors v of length i where exactly j entries are 1, and $p_z = \frac{l_{\min}}{l_z}$ is the probability that bus z arrives before time $l_{\min}$. Let then $\Phi[i,j]$ be the expected travel time conditioned on this event multiplied by j, thus

$$\Phi[i,j] = \sum_v \left(\prod_{z=1}^{i} p_z^{v_z} \prod_{z=1}^{i} (1-p_z)^{1-v_z} \sum_{z=1}^{i} v_z \mathbb{E}\left[T_z\right] \right).$$

Using inductive arguments, we see that Algorithm 1 fills these arrays. Now note that the expected waiting time of the prefix of buses $1, 2, \ldots, i$ is $l_{\min} \sum_{j=1}^{k} \frac{\Pi[i,j]}{j+1}$. On the other hand, the expected travel time is $\sum_{j=1}^{k} \frac{\Phi[i,j]}{j}$. Consequently, because of linearity of expectation, the final value of r_k is the total combined waiting and travel time when taking the first k buses, which implies the correctness of the algorithm. The running time is clearly $\mathcal{O}(n^3)$. Note that n is at most the maximum number of buses that pass any stop, and therefore cubic running time is feasible in practice.

▶ Algorithm 1. *Input:* T_i, l_i *for* $1 \le i \le n$

```
for $k$ in $1, \ldots, n$ :
  1. set all values in $\Phi$ and $\Pi$ to 0.0 and set $\Pi[0,0] = 1.0$
  2. $l_{\min} = \min_{1 \le i \le k} l_i$
  3. for $i$ in $1, \ldots, k$ :
     a. $p = \frac{l_{\min}}{l_i}$
     b. for $j$ in $1, \ldots, i$ :
           $\Pi[i,j] = (1-p)\Pi[i-1,j] + p\Pi[i-1,j-1]$
           $\Phi[i,j] = (1-p)\Phi[i-1,j] +$
           $p(\Phi[i-1,j-1] + \Pi[i-1,j-1]\mathbb{E}\left[T_i\right])$
     c. $\Pi[i,0] = 1.0 - \sum_{j=1}^{n} \Pi[i,j]$
  4. $r_k = \sum_{j=1}^{k} (l_{\min} \frac{\Pi[k,j]}{1+j} + \frac{\Phi[k,j]}{j})$
return the prefix $1, 2, \ldots, k$ that corresponds to the smallest $r_k$
```

Observe that Algorithm 1 does not provide an individual probability for each bus to be picked, which might be useful to analyze, for instance, the expected walking time, if each choice would imply a different walking time later on. It is somewhat surprising that it is possible to compute the combined waiting and travel time without getting this information as a byproduct. To derive this information, we can use the following Algorithm 2, which has again running time $\mathcal{O}(n^3)$. The input value k^* is the output of Algorithm 1, and the DP-array Π has the same meaning as in Algorithm 1. There is an additional DP-array Ψ, and this array is filled such that $\Psi[i,j,z]$ denotes the probability that from the prefix of buses $1, 2, \ldots, i$ exactly j many arrive before time $l_{\min}$ and z is one of them. The output P_i gives then the individual probability of a bus i to be picked as the first arriving one.

▶ Algorithm 2. *Input:* T_i, l_i *for* $1 \leq i \leq n$ *and* k^*

```
set all values in $\Psi$ and $\Pi$ to 0.0 and set $\Pi[0,0] = 1.0$ and $\Psi[0,0,0] = 1.0$
$l_{\min} = \min_{1 \leq i \leq k^*} l_i$
for $i$ in $1, \ldots, k^*$ :
  1. $p = \frac{l_{\min}}{l_i}$
  2. for $j$ in $1, \ldots, i$ :
     a. $\Pi[i,j] = (1-p)\Pi[i-1,j] + p\Pi[i-1,j-1]$
     b. $\Psi[i,j,i] = p\Pi[i-1,j-1]$
     c. for $z$ in $1, \ldots, i-1$ :
          $\Psi[i,j,z] = (1-p)\Psi[i-1,j,z] + p\Psi[i-1,j-1,z]$
  3. $\Pi[i,0] = 1.0 - \sum_{j=1}^{n} \Pi[i,j]$
for $i$ in $1, \ldots, k^*$ : return $P_i = \sum_{j=1}^{k^*} \frac{\Psi[k^*,j,i]}{j}$
```

3 GTFS Data Model

The goal of this section is to prepare input data in a way such that the techniques for uniform arrival times described in Section 2 can be applied. Specifically, we want to compute frequencies of buses, which is motivated by the fact that if a bus runs every 10 minutes, then a uniform arrival time in a 10 minutes interval is arguably the appropriate modeling choice.

The GTFS[5] format, formerly *Google Transit Format Specification*, allows the specification of public transportation time-tables in csv-files. Its basic entities are *trips* (defined in file *trips.txt*), which are described by a sequence of *stop times*, that is, combinations of stops and times (defined in file *stop_times.txt*). Thus, a trip only describes a single journey of a bus. It is important to note that there is no explicit grouping of trips into similar ones. One option is the *route* specification, but different trips assigned to the same route might have a different stop sequences. Another way is to associate trips with their corresponding *shapes*. However, shapes are more intended to describe a possible visualization. Therefore, the only way to logically group trips is to preprocess them into *lines* with a similar stop sequence and route identifier. We do this in hourly buckets, and then, for simplicity, take the first trip in any bucket as the one that defines the inter-stop travel times for the bucket. The number of trips in one bucket or *runs-per-hour (rph)* is then used to compute their frequency, e.g., if there are 6 runs-per-hour, then we assume that a trip runs every 10 minutes. This translates into 10 *headway minutes* or 600 *headway seconds* in GTFS, and would correspond to a waiting time uniformly distributed in interval $[0, 600]$ in terms of seconds. The following table gives an example of such a hourly bucket or *frequency* in file *frequencies.txt*.

trip_id	start_time	end_time	headway_secs	exact_times
freq_trip_0	08:00:00	08:59:59	600	0

Using this scheme, we can add frequencies to instances where such information is not available, that is, we compute the additional file *frequencies.txt*. Note that this file is part

[5] `https://developers.google.com/transit/gtfs/`

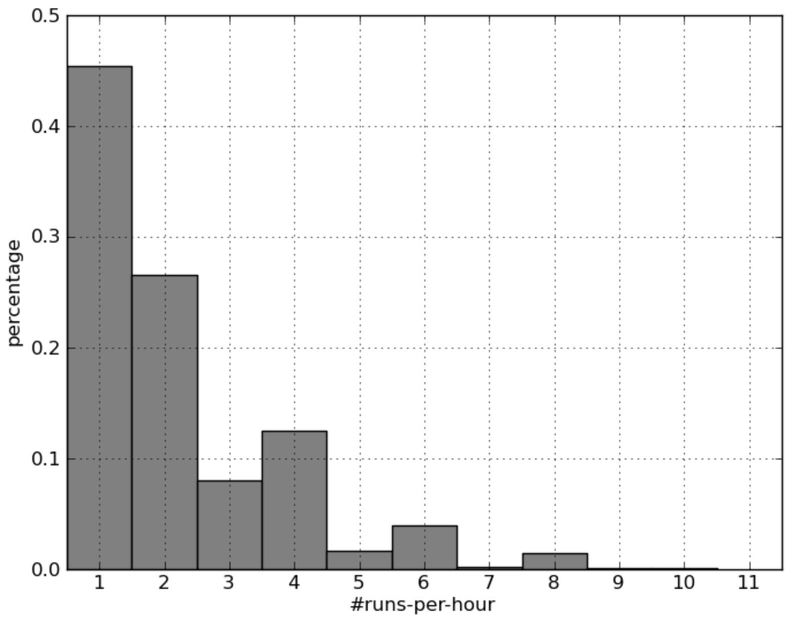

Figure 2 Runs-per-hour histogram of Dublin.

of the specification of GTFS, but it is almost never provided. Table 1 shows the original number of trips and the final number of frequencies for the considered instances.

Figure 2 shows a histogram of the average number of runs-per-hour in Dublin. Note that there is a peak at 4 and 6, which corresponds to having a trip every 15 and 10 minutes, respectively.

Clearly, more fine-grained methods could be applied to derive the necessary frequency information, for instance to avoid having sharp borders between buckets. It is also reasonable to only turn trips into frequencies if the number of runs-per-hour is above some threshold, say 3, but this would require some heuristic approach for mixing normal trips with frequencies during the routing process. Therefore, to allow an easy reproduction of results, we decided to use the presented basic method due to its simplicity.

4 Experiments and Results

First, since we are more interested in high-frequency inner-city traffic, we restrict the stops to the more central ones. This is done via first computing the geographic center of all stops, and then a function that indicates the decreasing density of stops when moving away from this center. We finally limit the radius of stops to consider such that the density is at least half the maximum density in the very center.

Second, on the remaining stops, we do a K-means clustering with $K = 20$, and from each cluster, we pick the centroid as a sample. This gives a representative selection of 20 stops. For instance, Figure 3 shows the inner-city of Dublin in dark grey with roughly labeled centroids. Each experiment is then executed on all 380 origin-destination pairs from this selection and every two hours between 8 o'clock and 18 o'clock to obtain averages of 2280 runs.

Table 1 Transformation of instances.

instance	Berlin	Budapest	Dublin	Oslo
planning date	10-06-11	14-09-12	19-11-12	06-12-13
#trips	45872	49905	7308	14200
#frequencies	20245	11554	3319	6353
#rph (average)	2.27	4.32	2.2	2.24

Table 2 Experimental results.

	Berlin	Budapest	Dublin	Oslo
travel (P1)	49.01	53.04	45.47	34.11
travel (P2)	49.22	50.45	44.41	36.58
travel (P3)	47.52	49.42	43.06	35.1
walk (P1)	8.35	13.23	10.04	7.35
walk (P2)	6.14	11.27	9.39	8.07
walk (P3)	6.05	10.57	8.5	8.09
wait (P1)	9.23	8.04	6.44	11.05
wait (P2)	13.12	9.43	11.49	16.38
wait (P3)	11.59	9.0	10.3	14.52
%trav. impr.	3.2	2.1	3.7	4.6
%trav. impr. (25%)	5.1	3.4	5.8	7.7
%wait impr.	5.8	2.3	7.6	8.2
%wait impr. (25%)	18.4	15.7	22.6	20.4

Other assumptions are: (1) we use the euclidean distance (on the earth surface) to approximate walking times between stops with walking speed 4km per hour, (2) we assume that we are allowed to switch buses at most three times, and (3) we assume that the traveler is aware of arriving buses at other stops within 50 meters, and hence we can select alternatives within this tolerance.

We consider three policies: (P1) an exact shortest path using the original GTFS-instance, (P2) a shortest path with alternatives using the derived GTFS-instance with frequencies where we allow only a single alternative at each stop, (P3) a shortest path with alternatives with an arbitrary number of alternatives at each stop.

We list our experimental results in Table 2. First, we give the average total travel times in minutes for the different policies, and then the respective walking and waiting times. Afterwards, we compare policies (P2) and (P3) in the second block. Rows *%trav. impr.* and *%wait impr.* give the average percentage of travel and waiting time improvement when allowing an arbitrary number of alternatives, respectively, and rows *%trav. impr. (25%)* and *%wait impr. (25%)* give the minimum travel and waiting time improvement in the top 25%-quantile.

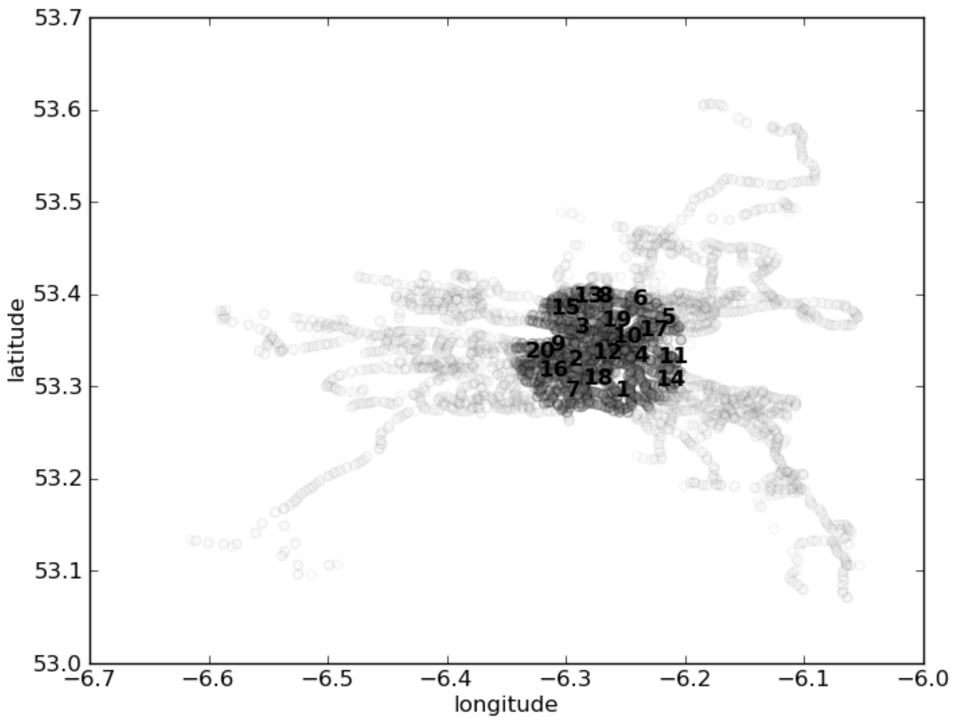

Figure 3 Origin-destination selection for Dublin.

5 Conclusion

We find that there are no large differences in travel times of all three policies, which is due to the fact that the travel time is dominated by the time spend in buses or walking, which are physical constraints that we cannot improve. It is interesting that policy (P1) does not clearly dominate the other policies.

Of course, policy (P1) is applied to the original GTFS-instance and hence gives exact routes, whereas policies (P2) and (P3) use the postprocessed GTFS-instance with frequencies, and therefore give policies with random travel time. Hence, this comparison should be considered as a high-level indicative study. However, in terms of total resources (buses) in the public transportation system, both sides are equal, only the latter case is stochastic.

As expected, the major difference between the three policies are the waiting times. Here we see that policy (P2) is strictly worse than policy (P1), it almost doubles the waiting time in some cases. However, a significant part of this waiting time increase can be absorbed by allowing more alternatives with policy (P3). More specifically, the benefit of allowing more alternatives is listed in the second block. We see again that the influence on the total travel time is relatively small, but around 25% of considered cases allow a reduction of waiting time of at least 20%.

We implemented our algorithms in C++ using the standard library and ran them on a single core of an Intel i5-2540M CPU with 2.60GHz and 8GB RAM. The shortest path

Table 3 Running times.

	Berlin	Budapest	Dublin	Oslo
(P1)	363	217	131	24
(P2)	613	314	472	72
(P3)	594	323	520	80

procedure is implemented using a standard Dijkstra-scheme. For the SPA policies, we implement the recurrence relation in this scheme using the algorithms described in Section 2, which gives additional overhead. On the other hand, we consider smaller instances in this case because of the clustering of trips into frequencies as described in Section 3. Table 3 lists the average running times in milliseconds for the different instances and policies. Note that the additional overhead due to the more involved recurrence relation is counterbalanced in a large part by using smaller instances. This shows that SPA policies can be practically computed even in interactive applications.

All our implementations build on a standard Dijkstra-scheme and do not further optimize this procedure. Therefore, many of the techniques used to speed-up this scheme could be used to derive faster running times, see for instance [2] for a comprehensive overview.

References

1 Utku Günay Acer, Paolo Giaccone, David Hay, Giovanni Neglia, and Saed Tarapiah. Timely data delivery in a realistic bus network. *IEEE Transactions on Vehicular Technology*, 61(3):1251–1265, 2012.

2 Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato Werneck. Route planning in transportation networks. Technical Report MSR-TR-2014-4, Microsoft Research, January 2014.

3 Hannah Bast and Sabine Storandt. Flow-based guidebook routing. In *Proceedings of the 16th Workshop on Algorithm Engineering and Experiments (ALENEX'14)*, pages 155–165, 2014.

4 Dimitri P. Bertsekas and John N. Tsitsiklis. An analysis of stochastic shortest path problems. *Math. Oper. Res.*, 16:580–595, August 1991.

5 Justin Boyan and Michael Mitzenmacher. Improved results for route planning in stochastic transportation. In *Proceedings of the 12th annual ACM-SIAM Symposium on Discrete Algorithms (SODA'01)*, pages 895–902, 2001.

6 Mayur Datar and Abhiram G. Ranade. Commuting with delay prone buses. In *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'00)*, pages 22–29, 2000.

7 Julian Dibbelt, Thomas Pajor, Ben Strasser, and Dorothea Wagner. Intriguingly simple and fast transit routing. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, pages 43–54, 2013.

8 Evdokia Nikolova, Jonathan A Kelner, Matthew Brand, and Michael Mitzenmacher. Stochastic shortest paths via quasi-convex maximization. In *Proceedings of the 14th Annual European Symposium on Algorithms (ESA'06)*, pages 552–563. Springer, 2006.

9 Tim Nonner. Polynomial-time approximation schemes for shortest path with alternatives. In *Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12)*, pages 755–765, 2012.