Fast Detour Computation for Ride Sharing*

Robert Geisberger, Dennis Luxen, Sabine Neubauer, Peter Sanders, and Lars Volker

Karlsruhe Institute of Technology, 76128 Karlsruhe, Germany
{geisberger,luxen,sanders}@kit.edu; bine.ka@gmx.de; lv@lekv.de

Abstract

Ride sharing becomes more and more popular not least because internet services help matching offers and request. However, current systems use a rather simple-minded functionality allowing to search for the origin and destination city, sometimes enriched with radial search around the cities. We show that theses services can be substantially improved using innovative route planning algorithms. More concretely, we generalize previous static algorithms for many-to-many routing to a dynamic setting and develop an additional pruning strategy. With these measures it becomes possible to match each request to n offers using $2n + 1$ exact travel time computations in a large road network in a fraction of a microsecond per offer. For requests spread over Germany according to population density, we are able to reduce the number of failing entries substantially. We are able to find a reasonable match for more than 60% of the failing entries left by contemporary matching strategies. Additionally, we halve the average waste of resources in the matches found compared to radial search.

1998 ACM Subject Classification G.2.2

Keywords and phrases ride sharing, algorithm engineering, carpool

Digital Object Identifier 10.4230/OASIcs.ATMOS.2010.88

1 Introduction

The concept of ride sharing can be described by the following observation: Two people, who we call driver and passenger, wish to travel from individual starting locations to destinations. These independent journeys have starting and ending locations that are relatively close to each other in the current setting. So, for economic reasons the travelers team up for some part of their journeys. They share the same vehicle for some time. Ride sharing creates a trade-off situation for the participants. Namely, cost of driving and owning a vehicle versus the time, money and resources needed to organize a shared ride and then split the overall cost among the participants.

Improving the matching mechanism has many beneficiaries. For example, ride sharers may use special carpool lanes or companies that live off brokerage fees can offer a more valuable service to their customers. Also, saved resources contribute to climate and environmental protection. Also, another possible benefit is reduced overall congestion, which is especially important in metropolitan areas.

There exist a number of web sites that offer ride sharing matching services to their customers. Unfortunately, as far as we know, all of them suffer from limitations in their method of matching.

Only a very small and limited subset of all the possible locations is actually modeled. This rather limited modeling has several shortcomings. For customers from sparsely populated

* Partially supported by DFG grant SA 933/5-1

10th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS'10).
Editors: Thomas Erlebach, Marco Lübbecke; pp. 88–99

OpenAccess Series in Informatics
Schloss Dagstuhl Publishing, Germany

areas, it can be quite difficult to decide on one of the possible origin and destination places as their location offered by the ride sharing service. Radial search around large cities has been introduced to help selecting approximate start and end points of a trip, but still it only helps the selection of a larger city nearby. The selection has to be done more or less manually because sometimes circumcircles intersect each other, which makes it even harder for the user to choose a valid starting point that leads to good matches. Also, from a technical point of view, a trip changes its start and end points when it is mapped into a predefined set of only a few locations. As a consequence, a correct ranking of possible matches by detour is too much to expect from such a ride sharing matching system.

Another downside is that matching services do not support what we call lazy pickup. The systems ignore any possible intermediate stop, if they are not given explicitly beforehand. Note that equally short routes for same pairs of origin and destination nodes can take arbitrarily different paths. Consider the following example to visualize the problem. Anne and Bob both live in Germany. Anne, the driver, is from Karlsruhe and wants to go to Berlin. Bob on the other side lives in Frankfurt and would like to travel to Leipzig. Taking the fastest route in our example, Anne drives from Karlsruhe via Nürnberg to Berlin and is never getting close enough to team up with Bob. However, there is a path from Karlsruhe to Berlin via Frankfurt, which also passes by the city of Leipzig and is only about one percent longer than the shortest path. In today's services it is mandatory to predetermine any possible stops artificially by hand, if they would like to pick up any passengers along a single predetermined route. Obviously, this reduces the possibility of flexible matching a lot. Matches that would have been perfect from a practical point of view, as in our example, are impossible to make since the route of the trip had to be fixed before it even started.

1.1 Related Work

Previous research on ride sharing focused on multiple areas. Several authors [5, 13, 8] investigated the socio-economic prerequisites of wide-spread customer adoption and overall economic potential of ride sharing. For example, Hartwig and Buchmann [8] analyze the ride sharing business case given that there exists a central service platform that can be accessed by mobile devices.

Other authors [15, 14] identified the missing spatial resolution of concurrent ride sharing services and examined a sensor network approach to metropolitan-local ride sharing offerings. Hand-held mobile devices function as nodes of the sensor network and communicate locally over short distances. Unfortunately, the work focuses on heuristic communication strategies. Likewise, no performance guarantees are possible and rides are only matched heuristically. Matching is done greedily and the first ride to go geometrically closer to the destination is taken. Note that geometric routing might lead to arbitrarily bad routes on a road network in the worst case.

Xing et al. [15] give an approach to ad-hoc ride sharing in a metropolitan area that is based on a multi-agent model and show the validity of their approach by simulation on a rather small metropolitan network. But in its current form the concept does not scale. As the authors point out it is only usable by a few hundred participants and not by several thousands or more participants that a real world ride sharing service would have.

To the best of our knowledge, there exists no previous work on fast detour computation, which would enable drivers and riders to meet somewhere in between. (Slow) detour computation is simple, it breaks down to several shortest paths computations and is therefore solvable by Dijkstra's algorithm [4] out of the box. In practice, however, these computations take too much time. There exist speed-up techniques for the shortest path computation, see

[3] for an overview. Transit Node Routing [1, 2] is the fastest technique whereas Contraction Hierarchies (CH) [7, 6] has the best trade-off between preprocessing and query time. The detour computation is similar to a distance table computation. The locations of offers in the database are fixed and lots of matching requests are performed so that preprocessing pays off. Currently the fastest algorithm to compute distance tables is [10] in combination with CH. The remaining parts of this paper are as follows. Section 2 gives an introduction into the technical model of ride sharing. Section 3 explains the algorithmic details while Section 4 gives an experimental evaluation on real-life data and an analysis of the results that we achieved. Section 5 draws conclusions and identifies future work.

2 Our Approach to Ride Sharing

We assume two types of users that we call *riders* and *drivers*. Drivers have a car and place *offers*, while riders place a *request* to be matched to an offer. A *service* is a more or less automated procedure to make those matchings.

For many services an offer only fits a request iff origin and destination locations and the possibly prefixed route of driver and rider are identical. We call such a situation a *perfect fit*. Some services offer an additional radial search around origin and destination and fixed way-points along the route. Usually, only the driver is able to prefix the route. The existence of these additions shows the demand for better route matching techniques that allow a small detour and intermediate stops. We call that kind of matching a *reasonable fit*. These fits are reasonable in the sense that the benefit of the match is much larger than the cost of the detour. However, previous approaches obviously used only features of the database systems they had available to compute the perfect fits. And we showed in the previous section that the previous approaches are not flexible, miss possibly interesting matches or require a lot of manual intervention.

We present an algorithmic solution to the situation at hand that leads to better results independent of the user's level of cooperation or available database systems. For that, we lift the restriction of a limited set of origin and destination points. Unfortunately, the probability of perfect fits is close to zero in this setting. But since we want to compute reasonable fits, our approach considers intermediate stops where driver and passenger might meet and depart later on. More precisely, we adjust the drivers route to pick up the passenger by allowing an acceptable detour.

We model the road network as a weighted graph $G = (V, E)$. A path P is a series of nodes $P = \langle v_1, \ldots, v_2 \rangle \in V$ with edges $(v_i, v_{i+1}) \in E$ between the nodes. The length $c(P)$ of a Path P is the sum of the weights, for example travel time, of all edges in P. Furthermore, $\mu(u, v)$ denotes the length of a shortest path in G for the origin destination pair $u, v \in V$. Consider the length of a not necessarily shortest path $c(P), P = \langle u, \ldots, v \rangle$ and the length of a shortest path $\mu(u, v)$. The *detour factor*ε is defined as the ratio of $\mu(u, v)$ and $c(P)$.

▶ **Definition 1.** Let $\varepsilon > 0$. We say that an offer $o = (s, t)$ and a request $g = (s', t')$ form a *reasonable fit* iff there exists a path $P = \langle s, \ldots, s', \ldots, t', \ldots, t \rangle$ in G with $c(P) \leq \mu(s, t) + \varepsilon \cdot \mu(s', t')$.

If we model riders' detour having the same cost as drivers detour, then the situation is completely symmetrical. The ε in Definition 1 depicts the maximal detour that is reasonable. Applying the ε to the riders path gives the driver an incentive to pick up the rider. A natural choice for the detour factor is $\varepsilon \leq 0.5$. For further explanation see Section 3.3.

3 Algorithmic Details

This section covers the algorithm to find all reasonable fits to an offer. We even solve the more general problem of computing all detours.

For a dataset of k offers $o_i = (s_i, t_i)$, i=1..k, and a single request $g = (s', t')$, we need to compute the $2k + 1$ shortest path distances $\mu(s', t')$, $\mu(s_i, s')$ and $\mu(t', t_i)$. The detour for offer o_i is then $\mu(s_i, s') + \mu(s', t') + \mu(t', t_i) - \mu(s_i, t_i)$. A naive algorithm would do a backward one-to-all search from s' using Dijkstra's algorithm and a forward one-to-all search from t' to compute $\mu(s_i, s')$ and $\mu(t', t_i)$. Another search returns the distance $\mu(s', t')$. We cannot prune the Dijkstra search early, as the best offer need not depart/arrive near the source/target of the request, so that each search takes several seconds on large road networks. In Section 4 we show that the running time of our algorithm is faster by several orders of magnitude.

To compute the distances, we adapt an algorithm for distance table computation [10]. This algorithm is based on non goal-directed, bidirectional search in a graph, preprocessed by a suitable speedup technique. Contraction Hierarchies [7] is currently the fastest one. Given a *forward search space* $S^{\uparrow}(x)$ from a source node x and a *backward search space* $S^{\downarrow}(y)$ from target node y, we can compute $\mu(x, y)$ by intersecting both search spaces. More formally, a forward search space $S^{\uparrow}(x)$ is a set of node/distance pairs $(u, d^{\uparrow})$ such that there is a path from x to u with distance $d^{\uparrow}$. And a backward search space $S^{\downarrow}(y)$ is a set of node/distance pairs $(u, d^{\downarrow})$ such that there is a path from u to y with distance $d^{\downarrow}$. The speedup technique guarantees that

$$\mu(x, y) = \min_{u \in V} \left\{ d^{\uparrow} + d^{\downarrow} \mid (u, d^{\uparrow}) \in S^{\uparrow}(x), (u, d^{\downarrow}) \in S^{\downarrow}(y) \right\} \quad . \tag{1}$$

Those nodes u that are in both search spaces are called *meeting nodes*. Note that $S^{\uparrow}(x)$ is independent of the target node y (non goal-directed) and can serve for any target node, and the same holds for $S^{\downarrow}(y)$. Both search spaces are small due to the preprocessing by the speedup technique [10, 7]. Nevertheless, we always compute the exact shortest path distance.

We solve our original problem by computing for each s_i the forward search space $S^{\uparrow}(s_i)$ in advance and store it. More precisely, we do not store each $S^{\uparrow}(s_i)$ separately, but we store *forward buckets*

$$B^{\uparrow}(u) := \left\{ (i, d^{\uparrow}) \mid (u, d^{\uparrow}) \in S^{\uparrow}(s_i) \right\} \tag{2}$$

with each potential meeting node u. To compute all $\mu(s_i, s')$ for the request, we compute $S^{\downarrow}(s')$, then scan the bucket of each node in $S^{\downarrow}(s')$ and compute all $\mu(s_i, s')$ simultaneously. We have an array of tentative distances for each $\mu(s_i, s')$. Initially, the distances are ∞, and we decrease them while scanning the buckets. The decrease happens following (3) that is deduced from (1) and (2).

$$\mu(s_i, s') = \min_{u \in V} \left\{ d^{\uparrow} + d^{\downarrow} \mid (i, d^{\uparrow}) \in B^{\uparrow}(u), (u, d^{\downarrow}) \in S^{\downarrow}(s') \right\} \quad . \tag{3}$$

Symmetrically, we compute *backward buckets* $B^{\downarrow}(u) := \left\{ (i, d^{\downarrow}) \mid (u, d^{\downarrow}) \in S^{\downarrow}(s_i) \right\}$ to accelerate the computation of all $\mu(t', t_i)$. Computing distances is very space- and cache-efficient, because it stores plain distances and scans consecutive pieces of memory. The single distance $\mu(s', t')$ is computed separately by computing the search spaces from s' and t' in the opposite directions.

Backward and forward buckets are stored in main memory and accessed as our main data structure and running queries on that data structure is easy.

3.1 Adding and Removing Offers

To add or remove an offer $o = (s, t)$, we only need to update the forward and backward buckets. To add the offer, we first compute $S^{\uparrow}(s)$ and $S^{\downarrow}(t)$. We then add these entries to their corresponding forward/backward buckets. To remove the offer, we need to remove its entries from the forward/backward buckets.

We make no decision on the order in which to store the entries of a bucket. This makes adding an offer very fast, but removing it requires scanning the buckets. Scanning all buckets is prohibitive as there are too many entries. Instead, it is faster to compute $S^{\uparrow}(s)$ and $S^{\downarrow}(t)$ again to obtain the set of meeting nodes whose buckets contain an entry about this offer. We then just need to scan those buckets and remove the entries. Also, we can omit removing offers by just having a separate bucket for each day, as described in the next section. We mark obsolete offers so that they will be ignored for any follow-up requests.

3.2 Constraints

In reality, offers and requests have constraints. For example, they specify a departure time window or they have restrictions on smoking, gender, etc. In this case, we need to extend the definition of a reasonable fit to meet these constraints by introducing additional indicator variables. As we already compute the detours of all offers, we can just filter the ones that violate the constraints of the request. Furthermore, our algorithm can potentially take advantage of these constraints, for example having buckets for each day separately. This way, we reduce the number of bucket entries that are scanned during a request. This significantly reduces the time to match a request as the bucket scans take the majority of the time.

3.3 Algorithmic Optimizations

We accelerate the request matching time by pruning bucket scans. We can omit scanning buckets when we limit the maximum detour to ε times the cost of the rider's shortest route. To do so, we look at a simple pricing scheme we know from algorithmic game theory. The so-called *fair sharing rule* [12] simply states that players who share a ride split costs evenly for the duration of the ride. Additionally, we define that drivers get compensated for their detours directly by riders using the savings from the shared ride. Implicitly, we give the driver an incentive to actually pick the passengers up at their start s' and to drop them off at their destination t'. Formally, we have that a match is economically worthwhile iff there exists a detour factor ε for which

$$\mu(s, s') + \mu(s', t') + \mu(t', t) - \mu(s, t) \leq \varepsilon \cdot \mu(s', t') \ .$$

The solid lines symbolize the distances that are driven, while the dashed one stands for the shortest path of the driver that is actually not driven at all in a matched ride.

Let's assume that ε is given, then we exploit the fact that we need to obtain $S^{\uparrow}(s')$ and $S^{\downarrow}(t')$ for the computation of $\mu(s', t')$. For $(u, d^{\uparrow})$ in $S^{\uparrow}(s')$ holds $d^{\uparrow} \geq \mu(s', u)$ and $(u, d^{\downarrow})$ in $S^{\downarrow}(t')$ holds $d^{\downarrow} \geq \mu(u, t')$. We compute the distance $\mu(s', t')$ before the bucket scanning, and additionally keep $S^{\uparrow}(s')$ and $S^{\downarrow}(t')$ that we obtained during this search. Then we can apply Lemma 2.

▶ **Lemma 2.** *Let $(u, d^{\downarrow}) \in S^{\downarrow}(s')$ and $(u, \overline{d}^{\downarrow}) \in S^{\downarrow}(t')$. We will not miss a reasonable fit when we omit scanning bucket $B^{\uparrow}(u)$ only if $d^{\downarrow} + \mu(s', t') > \overline{d}^{\downarrow} + \varepsilon \cdot \mu(s', t')$.*

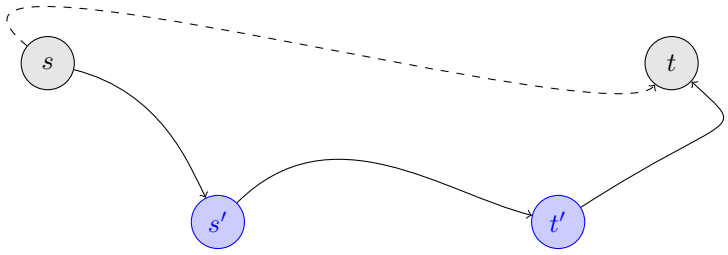

Figure 1 Request (s', t') and matching offer (s, t) with detour.

Let $(u, d^{\uparrow}) \in S^{\uparrow}(t')$ and $(u, \overline{d}^{\uparrow}) \in S^{\uparrow}(s')$. We will not miss a reasonable fit when we omit scanning bucket $B^{\downarrow}(u)$ only if $d^{\uparrow} + \mu(s', t') > \overline{d}^{\uparrow} + \varepsilon \cdot \mu(s', t')$.

Proof. Let $(u, d^{\downarrow}) \in S^{\downarrow}(s')$ and $(u, \overline{d}^{\downarrow}) \in S^{\downarrow}(t')$, and $d^{\downarrow} + \mu(s', t') > \overline{d}^{\downarrow} + \varepsilon \cdot \mu(s', t')$. Let $(i, d^{\uparrow}) \in B^{\uparrow}(u)$ be a pruned offer. If the path from s_i to s' via node u is not a shortest path, another meeting node will have offer o_i in its bucket, see (3). Therefore, WLOG we assume that $d^{\uparrow} + d^{\downarrow} = \mu(s_i, s')$. Let P be a path $P = \langle s_i, \ldots, s', \ldots, t', \ldots, t_i \rangle$ as visualised in Figure 2, then

$$
\begin{aligned}
c(P) &\geq && \mu(s_i, s') + \mu(s', t') + \mu(t', t_i) \\
&= && d^{\uparrow} + d^{\downarrow} + \mu(s', t') + \mu(t', t_i) \\
&\overset{\text{L.2}}{>} && d^{\uparrow} + \overline{d}^{\downarrow} + \varepsilon \cdot \mu(s', t') + \mu(t', t_i) \\
&\overset{d^{\uparrow}=\mu(s_i,u),\ \overline{d}^{\downarrow}\geq\mu(u,t')}{\geq} && (\mu(s_i, u) + \mu(u, t') + \mu(t', t_i)) + \varepsilon \cdot \mu(s', t') \\
&\overset{\triangle\text{-inequality}}{\geq} && \mu(s_i, t_i) + \varepsilon \cdot \mu(s', t')
\end{aligned}
$$

Therefore, P is not a reasonable fit. The proof is completely symmetric for omitting the scan of $B^{\downarrow}(u)$. ◀

Assume, that a passenger will not pay unreasonable high costs to share a ride, i.e. if it is cheaper to travel on his or her own. It is easy to see that any reasonable passenger will not pay more for the drivers detour than the gain for the shared ride which is at most $\frac{1}{2} \cdot \mu(s', t')$. Therefore, we conclude $\varepsilon \leq 0.5$. Of course, we acknowledge cultural differences and that an $\varepsilon > 0.5$ may be perfectly alright in certain parts of the world. Figure 1 gives a sketch on the line of argumentation.

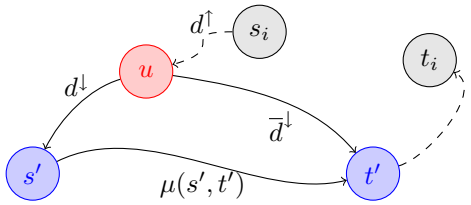

Figure 2 The difference $d^{\downarrow} + \mu(s', t') - \overline{d}^{\downarrow}$ is a lower bound on a detour via u.

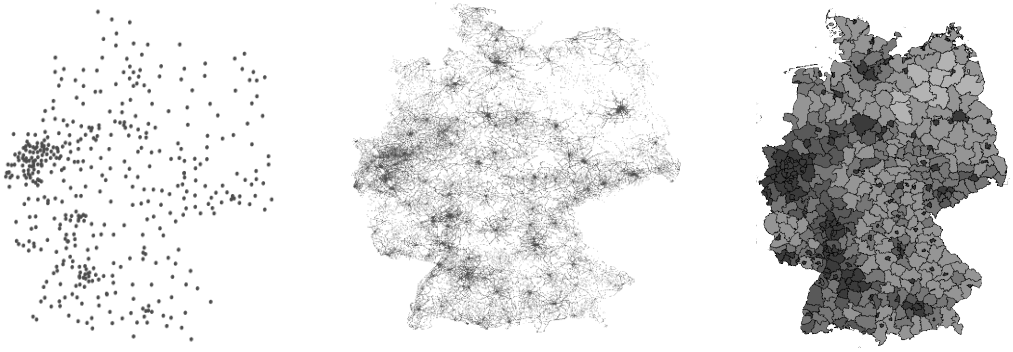

Figure 3 original node locations (left), perturbed node locations (middle), population density (right).

4 Experimental Results

4.1 Environment

Experiments have been done on one core of a single AMD Opteron Processor 270 clocked at 2.0 GHz with 8 GiB main memory and 2 × 1 MiB L2 cache, running SuSE Linux 11.1 (kernel 2.6.27). The program was compiled by the GNU C++ compiler 4.3.2 using optimization level 3.

4.2 Test Instances

Our graph of Germany is derived from the publicly available data of OpenStreetMap and consists of 6 344 491 nodes and 13 513 085 directed edges. The edge weights are travel times computed for the OpenRouteService car speed profile [1]. To test our algorithm, we obtained a dataset of real-world ride sharing offers from Germany available on the web. We matched the data against a list of cities, islands, airports and the like, and ended up with about 450 unique places. We tested the data and checked that the lengths of the journeys are exponentially distributed. This validates assumptions from the field of transportation science. We assumed that requests would follow the same distribution and chose our offers from that dataset as well.

To extend the data set to our approach of arbitrary origin and destination locations, we applied perturbation to the node locations of the data set. For each source node we unpacked the node's forward search space in the contraction hierarchy up to a distance of 3 000 seconds of travel time. From that unpacked search space we randomly selected a new starting point. Likewise we unpacked the backward search space of each destination node up to the distance and picked a new destination node. This approach applies to both offers and requests. We observed that perturbation preserved the distribution of the original data set.

Figure 3 compares the original node locations on the left to the result of the node perturbation in the middle. The right side shows a population density plot of Germany[2] to support the validity of the perturbation.

[1] See: http://wiki.openstreetmap.org/wiki/OpenRouteService
[2] Picture is an extract of an image available at episcangis.hygiene.uni-wuerzburg.de

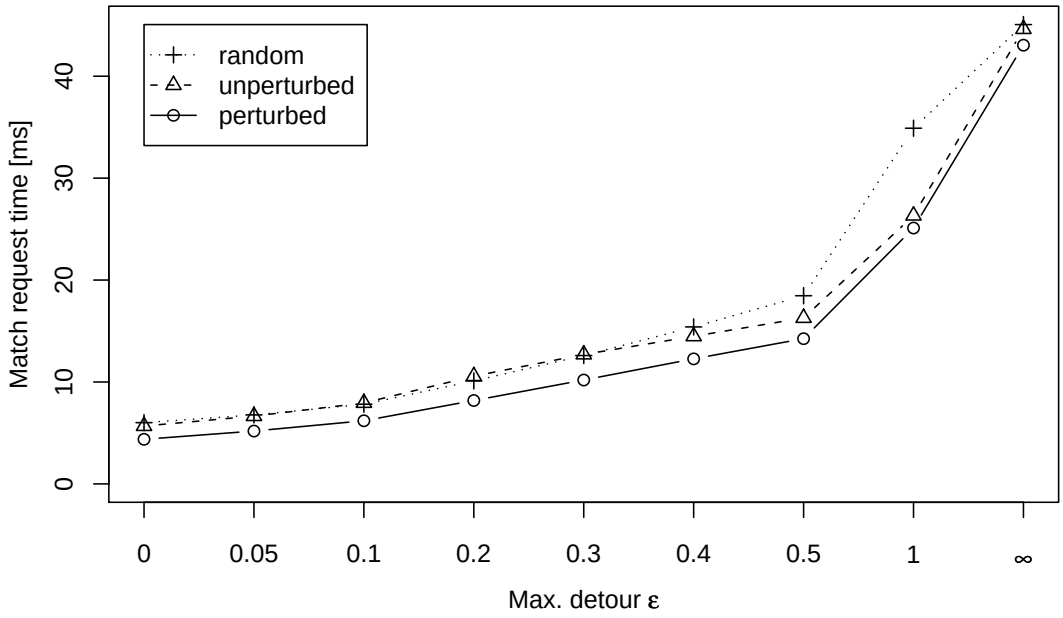

Figure 4 Match request performance for 100 000 offers.

We evaluated the performance of our algorithm for different numbers of offers where source and target are picked at random or from our un-/perturbed real-world dataset, see Table 1. We used CH (aggressive approach [6]) as bi-directed, non goal-directed speed-up technique. The size required for the bucket entries is linear with the number of offers, as a forward/backward search space have at most a few hundred nodes. The time to add an offer $o = (s, t)$ is independent of the number of offers, the main time is spent computing $S^{\uparrow}(s)$ and $S^{\downarrow}(t)$. However, removing an offer requires scanning the buckets, and therefore the more offers are in the database the more expensive it is. For our real-world offers, we have just 450 different source/target nodes, so that the bucket entries are clustered in only a few buckets, this still holds when we perturb the data. Of course, the bucket entries are more evenly distributed for completely random offers, the buckets are therefore smaller and removing an

Table 1 Performance of our algorithm for different types of offers/requests, numbers of offers and max. detours ε.

	#offers	bucket size	add offer	remove offer	match request [ms] $\varepsilon =$								
type		[MiB]	[ms]	[ms]	0.0	0.05	0.1	0.2	0.3	0.4	0.5	1	∞
perturbed	1 000	3	0.27	0.00	0.6	0.6	0.6	0.7	0.7	0.7	0.7	0.8	0.9
perturbed	10 000	28	0.24	0.29	0.9	1.0	1.1	1.3	1.5	1.6	1.8	2.7	4.1
perturbed	100 000	279	0.24	0.30	4.4	5.2	6.1	8.1	10.2	12.1	14.0	25.1	43.4
unperturbed	1 000	3	0.26	0.27	0.7	0.7	0.7	0.7	0.8	0.8	0.8	0.9	1.0
unperturbed	10 000	32	0.26	0.32	1.1	1.2	1.3	1.6	1.7	1.9	2.1	2.8	4.3
unperturbed	100 000	318	0.27	6.26	5.6	6.7	7.9	10.4	12.4	14.5	16.1	26.3	44.6
random	1 000	3	0.24	0.25	0.7	0.7	0.7	0.7	0.7	0.7	0.8	0.9	1.0
random	10 000	31	0.25	0.30	1.1	1.2	1.3	1.5	1.7	1.9	2.1	3.5	4.3
random	100 000	306	0.26	0.32	6.0	6.7	7.8	10.1	12.6	15.4	18.5	34.9	45.1

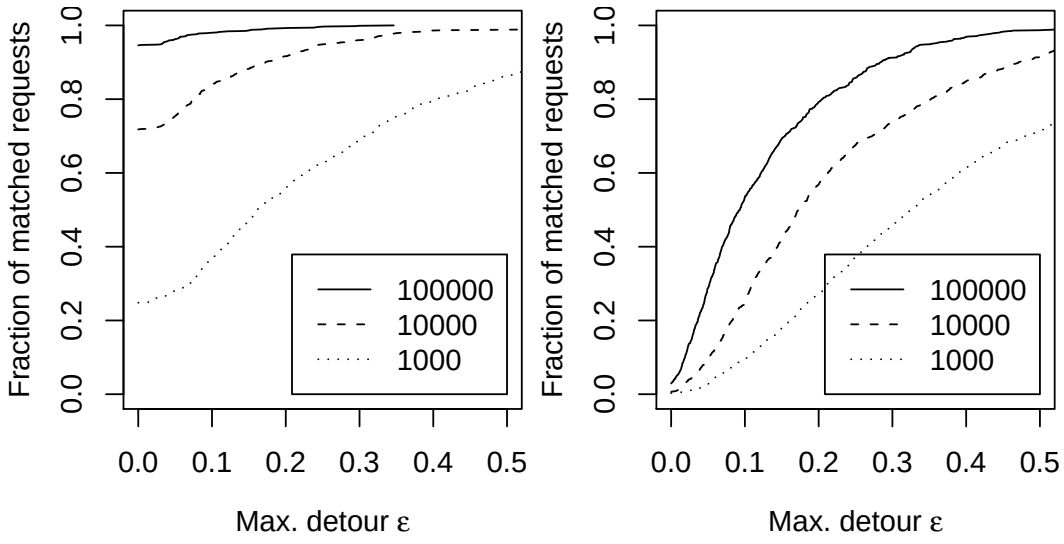

Figure 5 Fraction of rides matched for a given detour (unperturbed left, perturbed right).

offer takes less time. We report the time for matching a request for different values of ε. Even with no further optimization ($\varepsilon = \infty$), we can handle large datasets with 100 000 offers within 45 ms. In comparison, the fastest speedup technique today, Transit Node Routing (TNR) [2, 1] requires 1,9 μs[3] for each of the $2n + 1$ queries and would take about 380 ms for the largest dataset whereas our algorithm is 8.4 times faster. For a realistic $\varepsilon = 0.5$, we get a further speed-up of about 3. Figure 4 visualizes the performance for different ε. It mainly depends on ε and our algorithm is fairly robust against the different ways to pick source and target nodes.

Our method is also faster than TNR when we look at preprocessing. Although TNR does not need to add and store offers, our algorithm based on CH is still faster. The preprocessing of CH is one order of magnitude faster and has no space overhead, whereas TNR would require more than 1 GiB on our graph of Germany. This is more than enough time to insert even 100 000 offers and more than enough space to store the bucket entries, as Table 1 indicates.

We varied the allowed detour and investigated what influence it has on the number of matches that can be made. A random but fixed sample of 1 000 requests was matched against databases of various sizes. Figure 5 and Table 2 report on these experiments. The

[3] This query time is on the European road network, but since the number of access nodes should be the same on Germany, we can expect a similar query time there.

Table 2 Request matching rate for different values of maximum allowed detour.

	UNPERTURBED					PERTURBED				
	$\varepsilon =$					$\varepsilon =$				
#offers	0	0.05	0.1	0.2	0.5	0	0.05	0.1	0.2	0.5
1000	0.248	0.281	0.370	0.558	0.865	0.003	0.028	0.096	0.271	0.715
10000	0.718	0.755	0.840	0.917	0.989	0.006	0.093	0.248	0.569	0.914
100000	0.946	0.963	0.981	0.993	1.000	0.029	0.289	0.537	0.793	0.988

Table 3 Detour factors relative to the riders route length for the best answer achieved using radial search and using our algorithm.

#offers	radial search detour	smallest detour
1 000	0.806	0.392
10 000	0.467	0.227
100 000	0.276	0.128

experiment with unperturbed data represents the algorithms currently in use, where any user is allowed to do city-to-city queries only. For a realistic database size of 10 000 entries[4] and maximum allowed detour of $\varepsilon = 0.1$ we improve the matching rate to 0.84. This is a lot more than the 0.718 matching rate without detours. As expected, the matching rate increases with the number of offers.

The more realistic scenario with the perturbed data, where offers and requests are not only city-to-city, but point-to-point, becomes only practically possible with our new algorithm. The probability to find a perfect match in this scenario is close to zero. It is necessary to allow at least a small detour to find some matches. The ε required to find a match becomes larger, as we now also include intra-city detours and not only inter-city detours. Still, with a detour factor of $\varepsilon = 0.2$ we achieve a matching rate of 0.569, and for the maximum reasonable detour of $\varepsilon = 0.5$, we match 0.914 of all requests, that is 20% more than the 0.718 possible with a city-to-city perfect matching algorithm (unperturbed, $\varepsilon = 0$).

We also tested the quality of our algorithm against radial search. In the radial search setting, each request is matched against the offer with the smallest sum of Euclidean distances w.r.t. to origin and destination location of the request. This mimics radial search functions (with user supplied radii) offered in some current ride sharing systems. Table 3 reports the results. The average detour of all matches is less than half the detour that is experienced with radial search, which shows the performance of our approach. On the other hand, these numbers show the inferiority of radial search.

5 Conclusions and Future Work

We developed an algorithmic solution to efficiently compute detours to match ride sharing offers and request. This improves the matching rate for the current city-to-city scenario. In the new scenario for arbitrary starting and destination points, our algorithm is the first one feasible in practice, even for large datasets. Our algorithm is perfectly suitable for a large scale web service with potentially hundreds of thousands of users each day. This new scenario can increase the quality of the matches and the user satisfaction, potentially increasing the usage of ride sharing in the population.

Other cost functions are possible as well and perhaps not a single one, but several functions are used. Ranking the functions to produce a pareto-optimal solution or computing the *skyline* [11], i.e. the set of maxima, is an interesting problem in its own right.

We identify the adaption of our algorithm to time-dependent road networks and the incorporation of shared memory parallelism as as well as a distributed implementation [9]

[4] The database size of 10 000 entries is a realistic case and closely resembles the current daily amount of matches made by a known German ride sharing service provider, see: http://www.ea-media.net/geschaftsfelder/europealive/geschaftsfelder.html

as fields of future work. Incorporating car switching and multiple passengers per car will bring new and interesting algorithmic challenges. Although, the algorithm is sufficiently fast when dealing with 100 000 entries, we'd like to further improve it to deal with even larger input sizes of a magnitude of order and more. An adaption of the algorithm to the shared taxi system of developing countries will be very interesting as well.

References

1 Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.

2 Reinhard Bauer, Daniel Delling, Peter Sanders, Dennis Schieferdecker, Dominik Schultes, and Dorothea Wagner. Combining Hierarchical and Goal-Directed Speed-Up Techniques for Dijkstra's Algorithm. *ACM Journal of Experimental Algorithmics*, 15:2.3, January 2010. Special Section devoted to WEA'08.

3 Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Engineering Route Planning Algorithms. In Jürgen Lerner, Dorothea Wagner, and Katharina A. Zweig, editors, *Algorithmics of Large and Complex Networks*, volume 5515 of *Lecture Notes in Computer Science*, pages 117–139. Springer, 2009.

4 Edsger W. Dijkstra. A Note on Two Problems in Connexion with Graphs. *Numerische Mathematik*, 1:269–271, 1959.

5 John F. Dillenburg, Ouri Wolfson, and Peter C. Nelson. The Intelligent Travel Assistant. In *ITSS 2002: Proceedings of the 5h International Conference on Intelligent Transportation Systems*, pages 691–696. IEEE Computer Society, September 2002.

6 Robert Geisberger. Contraction Hierarchies. Master's thesis, Universität Karlsruhe (TH), Fakultät für Informatik, 2008. `http://algo2.iti.uni-karlsruhe.de/documents/routeplanning/geisberger_dipl.pdf`.

7 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In Catherine C. McGeoch, editor, *Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08)*, volume 5038 of *Lecture Notes in Computer Science*, pages 319–333. Springer, June 2008.

8 Stephan Hartwig and Michael Buchmann. Empty Seats Travelling. Technical report, Nokia Research Center, 2007.

9 Tim Kieritz, Dennis Luxen, Peter Sanders, and Christian Vetter. Distributed Time-Dependent Contraction Hierarchies. In Paola Festa, editor, *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*, volume 6049 of *Lecture Notes in Computer Science*. Springer, May 2010.

10 Sebastian Knopp, Peter Sanders, Dominik Schultes, Frank Schulz, and Dorothea Wagner. Computing Many-to-Many Shortest Paths Using Highway Hierarchies. In *Proceedings of the 9th Workshop on Algorithm Engineering and Experiments (ALENEX'07)*, pages 36–45. SIAM, 2007.

11 H. T. Kung, Fabrizio Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4):469–476, 1975.

12 Noam Nisan, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani, editors. *Algorithmic Game Theory*. Cambridge University Press, 2007.

13 Masyuku Ohta, Kosuke Shinoda, Yoichiro Kumada, Hideyuki Nakashima, and Itsuki Noda. Is Dial-a-Ride Bus Reasonable in Large Scale Towns? — Evaluation of Usability of Dial-a-Ride Systems by Simulation —. In *Multiagent for Mass User Support - First International Workshop*, volume 3012 of *Lecture Notes in Computer Science*, pages 105–119. Springer, 2004.

14 Yun Hui Wu, Lin Jie Guan, and Stephan Winter. Peer-to-Peer Shared Ride Systems. In *GeoSensor Networks*, volume 4540 of *Lecture Notes in Computer Science*, pages 252–270. Springer, August 2006.

15 Xin Xing, Tobias Warden, Tom Nicolai, and Otthein Herzog. SMIZE: A spontaneous Ride-Sharing System for Individual Urban Transit. In *Proceedings of the 7th German Conference on Multiagent System Technologies (MATES 2009)*, volume 5774 of *Lecture Notes in Computer Science*, pages 165–176. Springer, September 2009.