Time-Dependent Route Planning with Generalized Objective Functions[⋆]

Gernot Veit Batz and Peter Sanders

Karlsruhe Institute of Technology (KIT), 76128 Karlsruhe, Germany
{batz,sanders}@kit.edu

Abstract. We consider the problem of finding routes in road networks that optimize a combination of travel time and additional time-invariant costs. These could be an approximation of energy consumption, distance, tolls, or other penalties. The resulting problem is NP-hard, but if the additional cost is proportional to driving distance we can solve it optimally on the German road network within 2.3 s using a multi-label A* search. A generalization of time-dependent contraction hierarchies to the problem yields approximations with negligible errors using running times below 5 ms which makes the model feasible for high-throughput web services. By introducing tolls we get considerably harder instances, but still we have running times below 41 ms and very small errors.

1 Introduction

In the last years time-dependent route planning in road networks has gained considerable interest [1]. This has resulted in several algorithmic solutions that are efficient in both time [2, 3] and space [4, 5]. Even *profile queries*, that compute a result not only for a single but *all* departure times, can be answered fast [4]. However, all these techniques only deal with minimum *travel times*. Very little work has been done on more general time-dependent objective functions so far. This is disappointing from a practical perspective since real world route planners based on static network models take into account many other aspects one would not like to give up. For example, some approximation of energy consumption (just distance traveled in the most simple case) is important to avoid large detours just to save a few minutes of travel time – even environmentally insensitive users will not appreciate such solutions due to their increased cost. Similarly, many users want to avoid toll roads if this does not cost too much time. We might also want additional penalties, e.g., for crossing residential areas or for using inconvenient roads (narrow, steep, winding, bumpy,. . .). In Sect. 3 we show that this seemingly trivial generalization already makes the problem NP-hard. Nevertheless, on road networks computing such routes is feasible using a multi-label search algorithm based on A* – at least for some instances (Sect. 4). However, this algorithm is much too slow for applications like web-services that need running time in

[⋆] Partially supported by DFG project SA 933/5-1,2. The original publication is available at www.springerlink.com.

the low milliseconds in order to guarantee low delays and in order to handle many user requests on a small number of servers. We therefore generalize time-dependent contraction hierarchies (TCHs) [2, 4] to take additional constant costs into account. Although our adaptation gives up guaranteed optimality in favor of low query time, it turns out that the computed routes are nearly optimal in practice (Sect. 5). Just like original TCHs [6], the preprocessing of the adapted *heuristic TCHs* can be parallelized pretty well for shared memory. We support these claims by an experimental evaluation (Sect. 6).

Related Work. Time-dependent route planning started with classical results on minimum travel times [7, 8] and generalized minimum objective functions [9]. Dean provided a introductory tutorial on minimum travel time paths [10]. The aforementioned TCHs have also been parallelized in distributed memory [11].

2 Preliminaries

We model road networks as directed graphs $G = (V, E)$, where nodes represent junctions and edges represent road segments. Every edge $u \to v \in E$ has two kinds of weights assigned, a *travel-time function (TTF)* $f : \mathbb{R} \to \mathbb{R}_{\geq 0}$ and an *additional constant cost* $c \in \mathbb{R}_{\geq 0}$. We often write $u \to_{f|c} v$.

The TTF f specifies the time $f(\tau)$ we need to travel from u to v via $u \to_{f|c} v$ when departing at time $\tau \in \mathbb{R}$. The constant cost c specifies some additional expenses that incur on traveling along $u \to_{f|c} v$ independently from the departure time τ. So, the *total (time-dependent) cost* of traveling along the edge $u \to_{f|c} v$ is $C(\tau) = f(\tau) + c$. In road networks we usually do not arrive earlier when we start later. So, all TTFs f fulfill the *FIFO-property*, that is $\tau' + f(\tau') \geq \tau + f(\tau)$ for $\tau' > \tau$. Moreover, we model TTFs as periodic piecewise linear functions, usually with a period of 24 hours. Given a path $\langle u \to_{f|c} v \to_{g|d} w \rangle$ in G the TTF of the entire path is denoted by $g * f := g \circ (f + \mathrm{id}) + f$, that is $(g * f)(\tau) = g(f(\tau) + \tau) + f(\tau)$. Note that $f * (g * h) = (f * g) * h$ holds for TTFs f, g, h.

The time needed to travel along a path $P = \langle u_1 \to_{f_1|c_1} \cdots \to_{f_{k-1}|c_{k-1}} u_k \rangle$ instead of a single edge also depends on the departure time and is described by the TTF $f_P := f_{k-1} * \cdots * f_1$. The additional costs simply sum up along P and amount to $c_P := c_1 + \cdots + c_{k-1}$. Hence, the cost of traveling along P amounts to $C_P(\tau) := f_P(\tau) + c_P$ for departure time τ. As a lower and upper bound of this cost we further define $\underline{C}_P := \min f_1 + \cdots + \min f_{k-1} + c_P$ and $\overline{C}_P := \max f_1 + \cdots + \max f_{k-1} + c_P$ respectively.

Because of the FIFO-property there exists a minimum total cost for traveling from a node s to a node t for every departure time τ_0 in G, namely

$$\mathrm{Cost}(s, t, \tau_0) \ := \ \min\bigl\{C_Q(\tau_0) \bigm| Q \text{ is path from } s \text{ to } t \text{ in } G\bigr\} \cup \{\infty\} \ .$$

The function $\mathrm{Cost}(s, t, \cdot)$ is called the *cost profile* from s to t. If path P fulfills $C_P(\tau_0) = \mathrm{Cost}(u_1, u_k, \tau_0)$, then it is called a *minimum total cost path* for departure time τ_0. Note that waiting is never beneficial in the described setting.

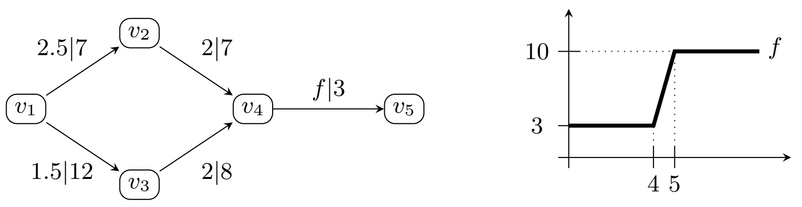

Fig. 1. A simple time-dependent graph. Most edges have constant TTFs. Only $v_4 \to v_5$ has the non-constant TTF f which is depicted on the right.

3 Complexity of Minimum Total Costs

On the first glance, computing minimum total cost paths looks related to the approach in static route planning with flexible objective functions [12] where a linear combination of objective functions is considered. However, this problem does not include any time-dependency but a time-*in*dependent parameter. Here, as we will show, we are stuck with a harder problem. Fig. 1 shows a simple example graph that demonstrates how quickly generalized time-dependent objective functions get unpleasant. There, $\langle v_1 \to v_3 \to v_4 \to v_5 \rangle$ is the only minimum total cost path from v_1 to v_5 for departure time 0.5. But its prefix path to v_4 has no minimal cost for departure time 0.5 and is suboptimal hence. This lack of *prefix optimality* implies that Dijkstra-like algorithms will not yield the correct result, as they throw suboptimal intermediate results away.

Our following proof shows that the problem is NP-hard. It uses a simple reduction to a partition problem[1] and is very much inspired by Ahuja et al. [13] who show the NP-hardness of a related problem with discrete time.

Theorem 1. *Computing a minimum total cost path for given start node, destination node, and departure time is NP-hard, even if there is only a single time-dependent edge with a constant number of bend points.*

Proof. To show our claim, we reduce the *number partitioning problem* to finding of a minimum total cost path: Given the numbers $b, a_1, \dots, a_k \in \mathbb{N}_{>0}$ we ask whether $x_1, \dots, x_k \in \{0, 1\}$ exist with $b = x_1 a_1 + \cdots + x_k a_k$. This question is an NP-complete problem [14]. Given an instance of number partitioning we construct the time-dependent graph depicted in Fig. 2. There are exactly 2^k paths from v_1 to v_{k+1}, all having the same total cost $2k + a_1 + \cdots + a_k$ but not necessary the same travel time. In particular, there is a path of travel time $b + k$ if and only if the underlying instance of number partitioning is answered yes. Hence, the answer is yes if and only if $\mathrm{Cost}(v_1, v_{k+2}, 0) = 1 + 2k + a_1 + \cdots + a_k$ holds. Note that the period of the TTF f has to be chosen sufficiently large. □

[1] Indeed, using a slightly more complicated construction we can also directly reduce the static bicriteria shortest path problem to our minimum total cost time-dependent problem. This requires only a single time-dependent edge.

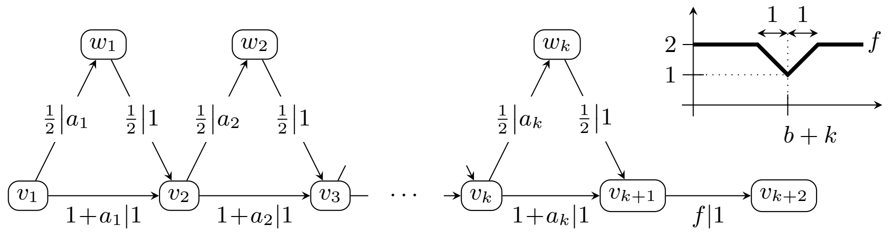

Fig. 2. A time-dependent graph encoding an instance of the number partitioning problem. All edges have constant TTFs except for $v_{k+1} \to v_{k+2}$ which has the TTF f depicted on the right.

Solving the problem for *all* departure times is also an interesting question in route planning – in order to find a convenient departure time for example. In other words, we want to compute cost profiles. The following statement suggests that cost profiles are harder to obtain than travel time profiles[2].

Theorem 2. *In a time-dependent network with additional constant costs and n nodes a cost profile can have $2^{\Omega(n)}$ bend points, even if there is only a single time-dependent edge with a constant number of bend points.*

Proof. Consider the graph in Fig. 2 with $a_i := 2^i$ for $1 \le i \le k$. Then

$$\mathrm{Cost}(v_1, v_{k+2}, \tau) = \min_{I \subseteq \{1,\ldots,k\}} \left\{ f\left(\tau + k + \sum\nolimits_{i \in I} a_i\right) + 2k + 2^{k+1} - 1 \right\}$$

holds. So, $\mathrm{Cost}(v_1, v_{k+2}, \cdot)$ has 2^k local minima and at least 2^k bend points. □

Compared to cost profiles, travel time profiles are relatively simple. According to Foschini et al. [15] they can never have more than $Kn^{O(\log n)}$ bend points for a network with n nodes and K bend points in total. Note that our proof of Theorem 2 uses nearly the same construction as Hansen did to show that bicriterion settings can raise exponentially many Pareto optimal paths [16]. This strengthens the observation that the two problems are connected.

4 Exact Minimum Total Costs

Minimum total cost paths cannot be computed by Dijkstra-like algorithms as prefix optimality is violated. However, *suffix* optimality is always provided. So, can we not just search backward? But this would require us to know the arrival time which is part of what we want to compute. At least, on computing cost *profiles* this is not a problem. We utilize this in the following Dijkstra-like, label correcting, backward running algorithm that starts searching at the destination node. Theorem 2 implies that its execution is very expensive of course.

[2] If the additional constant costs are zero for all edges, then total cost and travel time is the same. In this special case a cost profile is called a *travel time profile*.

Backward Cost Profile Search. Given a start node s and a destination node t we want to compute the cost profile $\mathrm{Cost}(s, t, \cdot)$. The label of a node w is a pair $f_w|c_w$. There, f_w is a piecewise linear, *piecewise continuous* function that maps the departure time at w to the time one needs to travel from w to t using the cheapest path discovered for this departure time so far. Correspondingly, c_w is a *piecewise constant* function that maps the departure time to the sum of additional constant costs along the respective path. The function $f_w + c_w$ is a continuous and piecewise linear tentative cost profile for traveling from w to t. All nodes have $\infty|\infty$ as initial label, except for t which has $0|0$. With defining

$$\min(g|d,\, h|e) : \tau \mapsto \begin{cases} g(\tau) \,|\, d(\tau) & \text{if } (g + d)(\tau) \le (h + e)(\tau) \\ h(\tau) \,|\, e(\tau) & \text{otherwise} \end{cases}$$

we relax an edge $u \to_{f|c} v$ in backward direction as follows: If $f_v|c_v$ is the label of v we update the label $f_u|c_u$ of u by $\min(f_u|c_u,\; f_v * f \,|\, c_v \circ (f + \mathrm{id}) + c)$. □

In forward direction Dijkstra-like algorithms are not applicable, but the following variant of *multi-label search* runs in forward direction and works. It is similar to the algorithm by Hansen [16] that finds all Pareto optimal paths in bicriteria settings. Our algorithm generalizes the time-dependent version [7] of Dijkstra's algorithm in a way that a node can have multiple labels at a time.

Plain Multi-Label Search. Given a start node s, a destination node t, and a departure time τ_0 we want to compute $\mathrm{Cost}(s, t, \tau_0)$. Nodes w are labeled with pairs $\tau_w|\gamma_w$ where τ_w is a time we arrive at w and γ_w is the sum of the additional constant costs of the corresponding path from s to w. The initial label of s is $\tau_0|0$. In every step we *expand* a label $\tau_u|\gamma_u$ of a node u such that the value $\tau_u + \gamma_u$ is currently minimal amongst all *non-expanded* labels of all nodes. Expanding $\tau_u|\gamma_u$ means, that we *relax* all outgoing edges $u \to_{f|c} v$ of u: If the new label $\tau_u + f(\tau_u)|\gamma_u + c$ is not *weakly dominated* by any existing label of u, it is added to the *label set* L_u of u.[3] Every old label in L_u *strictly dominated* by $\tau_u + f(\tau_u)|\gamma_u + c$ is removed from L_u, all other old labels in L_u are kept. The algorithm stops as soon as a label $\tau_t|\gamma_t$ of t is expanded for the first time. Then, with $\tau_t + \gamma_t - \tau_0 = \mathrm{Cost}(s, t, \tau_0)$, we have found the desired result. □

In Sect. 3 we state that we cannot necessary throw intermediate results (i.e., labels) away. But in case of dominated labels we can. However, the running time of plain multi label search is not feasible as our experiments show (Sect. 6). This can be improved by switching over to A* search. The resulting algorithm is inspired by the multiobjective A* algorithm NAMOA [17].

Multi-Label A* Search. With the *heuristic* function $\underline{h}_t(w) := \min\{\underline{C}_Q \mid Q$ is path from w to $t\}$ which fulfills $0 \le \underline{h}_t(w) \le \mathrm{Cost}(w, t, \tau)$ for all $w \in V$, $\tau \in \mathbb{R}$ we modify the plain multi-label search described above as follows: We do not choose the label $\tau_u|\gamma_u$ of an node u with minimal $\tau_u + \gamma_u$ amongst all non-expanded

[3] $\tau|\gamma$ *weakly dominates* $\tau'|\gamma'$ if and only if $\tau \le \tau'$ and $\gamma \le \gamma'$ holds. If we also have $\tau \ne \tau'$ or $\gamma \ne \gamma'$, then we speak of *strict dominance*.

labels of all nodes, but the label with minimal $\tau_u + \gamma_u + \underline{h}_t(u)$. The final result remains unchanged, but the *order* in which the labels are expanded can change a lot. If a label of t is expanded earlier, we have to expand less labels before the computation stops. This can save much running time. $\square$

Our experiments show that $\underline{h}_t$ makes the computation more feasible on road networks (Sect. 6). To compute $\underline{h}_t$ we could run Dijkstra's algorithm in a backward manner starting from t as an initial step of the computation. But then we would process the whole graph as we would not know when to stop. Instead we perform an initial backward *interval search* [4] to compute the intervals $[\underline{h}_t(w), \overline{h}_t(w)]$ for all reached nodes w (with $\overline{h}_t(w) := \max\{\overline{C}_Q \mid Q \text{ is path from } w \text{ to } t\}$). This enables us to stop the interval search as soon as the minimum key of the priority queue exceeds $\overline{h}_t(s)$. Also, multi-label A* search can use $\overline{h}_t$ to maintain an upper bound of the desired result $\mathrm{Cost}(s, t, \tau_0)$ and then desist from relaxing edges $u \rightarrow_{f|c} v$ where $(\tau_u + \gamma_u) + (f(\tau_u) + c) + \underline{h}_t(v)$ exceeds this upper bound – or where v has not even been reached by the interval search.

5 Heuristic Total Costs with TCHs

Essentially, a contraction hierarchy [18, 19] orders the nodes by some notion of *importance* with more important nodes higher up in the hierarchy. The hierarchy is constructed bottom up during *preprocessing*, by successively *contracting* the least important remaining node. Contracting a node v means that v is removed from the graph while preserving all optimal routes. To do so, we have to insert a *shortcut* edge $u \to w$ for every deleted path $\langle u \to v \to w \rangle$ that is part of an optimal route. The result of this construction is the contraction hierarchy (CH). It is stored as a single graph in a condensed way: Every node is materialized exactly once and the original edges and shortcuts are put together and *merged* if necessary. The CH has the useful property that optimal paths from s to t can be constructed from an *upward path* $\langle s \to \cdots \to x \rangle$ and a *downward path* $\langle x \to \cdots \to t \rangle$.[4] This enables us to find an optimal route basically by performing two upward searches, a forward and a backward search each starting from s and t. As a well-constructed CH should be flat and sparse, such a *bidirectional* search should only take little running time.

Preprocessing. When contracting a node v we want to find out whether a shortcut $u \to w$ has to be inserted for a path $\langle u \to v \to w \rangle$. In other words, we have to discover whether $C_P(\tau) = \mathrm{Cost}(s, t, \tau)$ holds for some path $P := \langle s \to \cdots \to u \to v \to w \to \cdots \to t \rangle$ and some $\tau \in \mathbb{R}$. However, the lack of prefix-optimality imposed by the additional constant costs makes this question difficult to answer as non-optimal routes can now be part of optimal ones. To decide correctly, we could examine all *non-dominated* (i.e., Pareto optimal) paths from u to w. But as we had to do this for *all* departure times, we expect that this is too expensive

[4] Upward and downward paths use only edges leading from less important to more important nodes and from more important to less important nodes respectively.

with respect to time and space. Also, this might produce so many shortcuts that also the query times would be disappointing.

Therefore we switched over to a heuristic version which may loose the one or another optimal path. We insert a shortcut $u \rightarrow_{g*f \mid d\circ(f+\mathrm{id})+c} w$ for a deleted path $\langle u \rightarrow_{f|c} v \rightarrow_{g|d} w\rangle$ when $(g * f + d \circ (f + \mathrm{id}) + c)(\tau) = C_{\langle u \rightarrow_{f|c} v \rightarrow_{g|d} w\rangle}(\tau) = \mathrm{Cost}(u, w, \tau)$ holds for some $\tau \in \mathbb{R}$ (note that c and d can now be piecewise constant functions and that f and g can now have points of discontinuity). To check this condition we compute $\mathrm{Cost}(u, w, \cdot)$ using cost profile search (Sect. 4). If a shortcut $u \rightarrow_{g*f \mid d\circ(f+\mathrm{id})+c} w$ has to be inserted for a path $\langle u \rightarrow_{f|c} v \rightarrow_{g|d} w\rangle$ it can happen that an edge $u \rightarrow_{h|e} w$ is already present. In this case we *merge* the two edges, that is we replace $u \rightarrow_{h|e} w$ by $u \rightarrow_{\min(g*f \mid d\circ(f+\mathrm{id})+c,\ h|e)} w$. It is this kind of merging which introduces noncontinuous functions to the preprocessing and the resulting *heuristic* TCH.

Just like in case of the original travel time TCHs [2, 4, 6] preprocessing of heuristic TCHs is a computationally expensive task (or maybe even more expensive with respect to Theorem 2). So, to make the preprocessing feasible, we adopted many of the techniques applied to travel time TCHs.

Querying. Given a start node s, a destination node t, and a departure time τ_0, querying works similar as in case of travel time TCHs: In a first phase, we perform the aforementioned bidirectional search where forward and backward search only go upward. Here, the forward search is a plain multi-label search starting from s with initial label $\tau_0|0$. The backward search is a interval search as performed before running the multi-label A* search (Sect. 4). Every node x where the two searches meet is a *candidate* node. Both searches apply *stall on demand* known from the original travel time TCHs. But, in case of the forward search we stall a node based on strict dominance. In case of the backward search we stall a node based on upper and lower bounds.

In a second phase we perform a *downward search* which is again a plain multi-label search But this time it runs downward in the hierarchy starting from the candidate nodes processing only edges touched by the backward search. The labels taken from the forward search act as initial labels.

6 Experiments

Inputs and Setup. As input we use a road network of Germany provided by PTV AG for scientific use. It has 4.7 million nodes, 10.8 million edges, and TTFs reflecting the midweek (Tuesday till Thursday) traffic collected from historical data, i.e., a high traffic scenario with about 7.2 % non-constant TTFs. For all edges also the *driving distance* is available which we use as a basis to form the additional constant costs. Our idea is that the additional constant cost of an edge estimates the energy consumption raised by traveling along that edge. With typical gasoline prices we assume that driving 1 km costs 0.1 €. To prize the travel time we use rates of 5 €, 10 €, and 20 € per hour. So, using 0.1 s as unit of time and m as unit of distance we obtain time-dependent total costs of

$$\textit{time} + \lambda \cdot \textit{distance}$$

where λ has the values 0.72, 0.36, and 0.18 respectively. Accordingly, the additional constant edge costs are simply the driving distance scaled by the respective value of λ. Hence, we have three instances of Germany, one for every value of λ.

We also consider the effect of tools. To do so, we fix an extra price of 0.1 € per km on motorways. This means that edges belonging to motorways have double additional constant edge costs. This yields three further instances of Germany, one for every value of λ.

The experimental evaluation was done on different 64-bit machines with Ubunbtu Linux. All running times have been measured on one machine with four Core i7 Quad-Cores (2.67 Ghz) with 48 GiB of RAM with Ubuntu 10.4. There, all programs were compiled using GCC 4.4.3 with optimization level 3. Running times were always measured using one single thread except of the preprocessing where we used 8 threads.

The performance of the algorithms is evaluated in terms of running time, memory usage, error, and how often *deleteMin* is invoked. The latter comes from the fact, that we implemented all algorithms using priority queues. To measure the average running time of time-dependent cost queries we use 1 000 randomly selected start and destination pairs, together with a departure time randomly selected from $[0\text{h}, 24\text{h})$ each. To measure the errors and the average number of invocations of *deleteMin* we do the same but with 10 000 test cases instead of 1 000. The memory usage is given in terms of the average *total* space usage of a node (not the overhead) in byte per node. For TCH-based techniques, all figures refer to the scenario that not only the total time-dependent costs but also the routes have to be determined. This increases the running time and the memory consumption a bit.

Results. For the different values of λ, Table 1 summarizes the behavior of the different algorithmic techniques. These are multi-label A* search (Sect. 4), heuristic TCHs (Sect. 5), and the original travel time TCHs [4]. In case of the travel time TCHs we did not use their original query algorithm, but the algorithm of the heuristic TCHs as described in Sect. 5. It turns out that with tolls we get much harder instances than without tolls, as the observed space usages and running times are the larger there. Without tolls we get the hardest instance at $\lambda = 0.72$, which corresponds to an hourly rate of 5 €. For smaller values of λ, which involve that time and total costs are more correlated, things seem to be easier. With tolls, in contrast, we get the hardest instance at $\lambda = 0.36$. This is because tolls are negatively correlated to travel time.

Without tolls, multi-label A* has running times similar to Dijkstra's algorithm, as its running time is mainly governed by the running time of the Dijkstra-like backward interval search in this case. This follows from the relatively low number of invocations of *deleteMin* (the invocations raised by the backward interval search are not included in Table 1). Note that this low number of invocations implies that a more efficient realization of the heuristic function $\underline{h}_t$

Table 1. Behavior of time-dependent total cost queries for different values of λ. Node ordering is performed in parallel with 8 threads. delMins= number of invocations of deleteMin (without interval search in case of A*), MAX and AVG denote the maximum and average relative error, rate= percentage of routes with a relative error $\geq 0.05\%$.

		space	order	query		error		
		usage	time	delMin	time	MAX	AVG	rate
method	λ	[B/n]	[h:m]	#	[ms]	[%]	[%]	[%]
Germany midweek, no toll								
multi-label A*	0.72	130	–	253 933	2 328.72	–	–	–
heuristic TCH	0.72	1 481	0:28	2 142	4.92	0.09	0.00	0.00
travel time TCH	0.72	1 065	0:21	1 192	2.67	12.40	0.68	1.19
multi-label A*	0.36	130	–	184 710	2 208.76	–	–	–
heuristic TCH	0.36	1 316	0:26	1 774	4.22	0.03	0.00	0.00
travel time TCH	0.36	1 065	0:21	1 183	2.33	7.69	0.27	0.08
multi-label A*	0.18	130	–	150 970	2 234.04	–	–	–
heuristic TCH	0.18	1 212	0:25	1 464	3.51	0.01	0.00	0.00
travel time TCH	0.18	1 065	0:21	1 165	2.33	3.85	0.08	0.00
Germany midweek, with toll								
heuristic TCH	0.72	1 863	1:05	4 676	14.96			
travel time TCH	0.72	1 065	0:21	2 631	4.54			
heuristic TCH	0.36	2 004	1:16	10 725	40.96			
travel time TCH	0.36	1 065	0:21	2 634	4.46			
heuristic TCH	0.18	1 659	0:46	7 347	27.90			
travel time TCH	0.18	1 065	0:22	2 482	4.39			

would turn multi-label A* search into a pretty fast algorithm. Hub-labeling [20] may be an appropriate candidate. However, with tolls multi-label A* is no longer efficient. Accordingly, Table 1 omits some errors and error rates as we do not have the exact results for tolls.

With 5 ms and 41 ms heuristic TCHs have much faster query time than multi-label A*. Though being heuristic in theory, the method is practically exact for the kind of costs examined here, as there are nearly no routes with a relative error significantly away from 0 %. Admittedly, there are outliers, but they are not serious. However, the memory consumption is quite large. But it is likely that similar techniques as used for ATCHs [4] can reduce the memory consumption very much. Please note that our preprocessing is partly prototypical and takes considerably longer as in case of our original TCH implementation [4]. It may be possible to do the preprocessing faster hence.

Simply using the original TCHs with the adapted query algorithm of heuristic TCHs is not a good alternative to heuristic TCHs. Though the average error is relatively small, some more serious outliers spoil the result. This is supported by Fig. 3 and 4 which show the distribution of the relative error over the Dijkstra rank[5]. In practice, even few serious outliers may annoy some users which may

[5] For $i = 5..22$ we select a number of random queries such that the time-dependent version of Dijkstra's algorithm settles 2^i nodes. We call 2^i the *Dijkstra rank*.

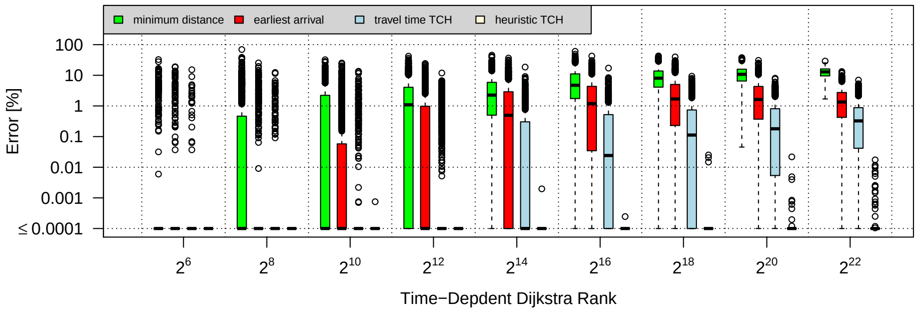

Fig. 3. Relative error of time-dependent total cost plotted over Dijkstra rank for different kinds of routes with $\lambda = 0.72$ and no tolls: minimum distance routes (green), earliest arrival routes (red), routes created using travel time TCHs (blue), and using heuristic TCHs (yellow). Number of queries per rank is 1000 for every algorithm.

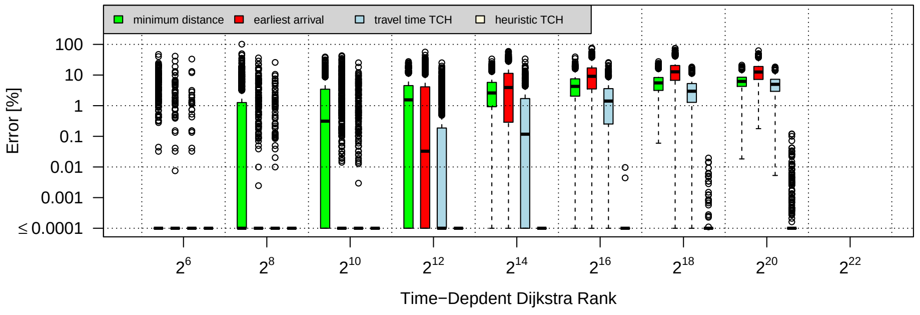

Fig. 4. Like Fig. 3 but with $\lambda = 0.36$ and with tolls. No error known for rank 2^{22}.

already lead to a bad publicity. Of course, this can affect the success of a product or service.

Fig. 3 and 4 also report the quality of minimum distance and minimum travel time routes with respect to time-dependent total costs. Obviously, the quality gets worse with increasing Dijkstra rank. However, serious outliers occur for all ranks. Fig. 5 shows how the relative error behaves for the three different values of λ – with and without tolls. We are able to report errors for tolls for all Dijkstra ranks up to 2^{20}. For higher ranks the multi-label A* gets too slow. It turns out that with tolls the errors are larger but still small.

7 Conclusions and Future Work

We have shown that time-dependent route planning with additional constant costs is NP-hard in theory, but more than feasible on real-life road networks when we optimize travel time with a penalty proportional to driving distance: Our exact multi-label A* search finds optimal routes on the German road network within 2.3 s. But this may not be the end of the story. Using hub-labeling [20] as a

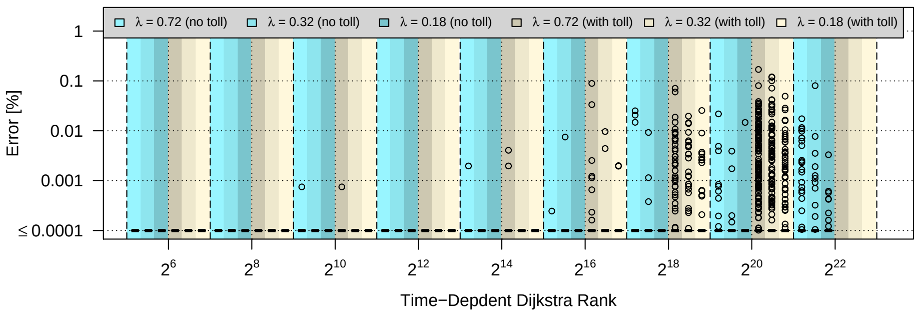

Fig. 5. Boxplot of relative error over Dijkstra rank with heuristic TCHs for $\lambda = 0.72, 0.36, 0.18$. As less than 25 % of the routes have greater error than 0.0001 %, all boxes and whiskers degenerate to small bars at the bottom. So, for readability we underlay all figures with colors as indicated in the legend. The number of queries per instance and rank is 1 000. With toll we do not know the error for rank 2^{22}.

heuristic oracle should yield average running times considerably smaller than 1 s. Our heuristic TCHs compute routes within 5 ms with mostly negligible errors. Motorway tolls, however, which are negatively correlated to travel time, make the problem considerably harder such that the multi-label A* search is no more feasible. But still our heuristic TCHs show running times below 41 ms and small errors. Our current implementation of heuristic TCHs is very space consuming. However, the careful use of approximation – which greatly reduced the space usage of original TCHs [4] – should also work well here. Faster computation of heuristic time-dependent cost profiles with very small error should also be possible with heuristic TCHs. Note that the very efficient *corridor contraction* [4] turns to be heuristic in this setting too. An A*-like backward cost profile search in the corridor with a preceding interval search in the corridor to provide a heuristic function may bring some speedup too.

Our ideas should also be applicable to additional costs that are themselves time-dependent as long as the overall cost function has the FIFO-property. Even waiting, which can be beneficial in some situations, may be translatable in such a setting. Whether all this runs fast enough in practice, has to be found out experimentally of course.

Acknowledgements. We thank Robert Geisberger for fruitful discussions.

References

1. Delling, D., Wagner, D.: Time-Dependent Route Planning. In Ahuja, R.K., Möhring, R.H., Zaroliagis, C., eds.: Robust and Online Large-Scale Optimization. Volume 5868 of Lecture Notes in Computer Science. Springer (2009) 207–230
2. Batz, G.V., Delling, D., Sanders, P., Vetter, C.: Time-Dependent Contraction Hierarchies. In: Proceedings of the 11th Workshop on Algorithm Engineering and Experiments (ALENEX'09), SIAM (April 2009) 97–105

3. Delling, D.: Time-Dependent SHARC-Routing. Algorithmica **60**(1) (May 2011) 60–94 Special Issue: European Symposium on Algorithms 2008.
4. Batz, G.V., Geisberger, R., Neubauer, S., Sanders, P.: Time-Dependent Contraction Hierarchies and Approximation. [21] 166–177
5. Brunel, E., Delling, D., Gemsa, A., Wagner, D.: Space-Efficient SHARC-Routing. [21] 47–58
6. Vetter, C.: Parallel Time-Dependent Contraction Hierarchies (2009) Student Research Project. `http://algo2.iti.kit.edu/download/vetter_sa.pdf`.
7. Dreyfus, S.E.: An Appraisal of Some Shortest-Path Algorithms. Operations Research **17**(3) (1969) 395–412
8. Orda, A., Rom, R.: Shortest-Path and Minimum Delay Algorithms in Networks with Time-Dependent Edge-Length. Journal of the ACM **37**(3) (1990) 607–625
9. Orda, A., Rom, R.: Minimum Weight Paths in Time-Dependent Networks. Networks **21** (1991) 295–319
10. Dean, B.C.: Shortest Paths in FIFO Time-Dependent Networks: Theory and Algorithms. Technical report, Massachusetts Institute Of Technology (1999)
11. Kieritz, T., Luxen, D., Sanders, P., Vetter, C.: Distributed Time-Dependent Contraction Hierarchies. [21] 83–93
12. Geisberger, R., Kobitzsch, M., Sanders, P.: Route Planning with Flexible Objective Functions. In: Proceedings of the 12th Workshop on Algorithm Engineering and Experiments (ALENEX'10), SIAM (2010) 124–137
13. Ahuja, R.K., Orlin, J.B., Pallottino, S., Scutellà, M.G.: Dynamic Shortest Paths Minimizing Travel Times and Costs. Networks **41**(4) (2003) 197–205
14. Garey, M.R., Johnson, D.S.: Computers and Intractability. A Guide to the Theory of $\mathcal{NP}$-Completeness. W. H. Freeman and Company (1979)
15. Foschini, L., Hershberger, J., Suri, S.: On the Complexity of Time-Dependent Shortest Paths. In: Proceedings of the 22nd Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'11), SIAM (2011) 327–341
16. Hansen, P.: Bricriteria Path Problems. In Fandel, G., Gal, T., eds.: Multiple Criteria Decision Making – Theory and Application –. Springer (1979) 109–127
17. Mandow, L., Pérez-de-la-Cruz, J.L.: Multiobjective A* Search with Consistent Heuristics. Journal of the ACM **57**(5) (June 2010) 27:1–27:24
18. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. In McGeoch, C.C., ed.: Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08). Volume 5038 of Lecture Notes in Computer Science., Springer (June 2008) 319–333
19. Geisberger, R., Sanders, P., Schultes, D., Vetter, C.: Exact Routing in Large Road Networks Using Contraction Hierarchies. Transportation Science (2012) Accepted for publication.
20. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: A Hub-Based Labeling Algorithm for Shortest Paths on Road Networks. In Pardalos, P.M., Rebennack, S., eds.: Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11). Volume 6630 of Lecture Notes in Computer Science., Springer (2011) 230–241
21. Festa, P., ed.: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10). In Festa, P., ed.: Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10). Volume 6049 of Lecture Notes in Computer Science., Springer (May 2010)