Pareto Paths with SHARC[⋆]

Daniel Delling and Dorothea Wagner

Universität Karlsruhe (TH), 76128 Karlsruhe, Germany, {delling,wagner}@ira.uka.de

Abstract. Up to now, research on speed-up techniques for DIJKSTRA's algorithm focused on *single-criteria* scenarios. The goal was to find the quickest route within a transportation network. However, the quickest route is often not the best one. A user might be willing to accept slightly longer travel times if the cost of the journey is less. A common approach to cope with such a situation is to find *Pareto-optimal* (concerning other metrics than travel times) routes. Such routes have the property that each route is better than any other route with respect to at least one metric under consideration, e.g., travel costs or number of train changes. In this work, we study multi-criteria search in road networks. On the one hand, we focus on the problem of limiting the number of Pareto paths. On the other hand, we present a multi-criteria variant of our recent SHARC algorithm.

1 Introduction

The computation of quickest paths in graphs is used in many real-world applications like route planning in road networks, timetable information for railways, or scheduling for airplanes. In general, DIJKSTRA's algorithm [1] finds a quickest path between a given source s and target t. Unfortunately, the algorithm is far too slow to be used on huge datasets. Thus, several speed-up techniques have been developed (see [2] for an overview) that can retrieve the quickest path in a road network within less than a millisecond.

However, the quickest route in transportation networks is often not the "best" one. For example, users traveling by car may be willing to accept (slightly) longer travel times if the costs of the journey (toll, fuel consumption) is lower. A possible approach to such better routes is to run a multi-criteria query which incorporates other metrics besides travel times for finding a set of attractive routes from which a user can choose. Unfortunately, all methods developed during the last years only work in single-criteria scenarios. We here present an augmented version of our recently developed SHARC (SHortcuts + ARC-flags) algorithm working in such a multi-criteria scenario.

1.1 Related Work

A lot of speed-up techniques for single-criteria scenarios have been developed during the last years. Due to space limitations, we direct the interested reader to [2], which gives a recent overview over single-criteria routing techniques.

[⋆] Partially supported by the Future and Emerging Technologies Unit of EC (IST priority – 6th FP), under contract no. FP6-021235-2 (project ARRIVAL) and the DFG (project WA 654/16-1).

Basics. The straightforward approach to find all Pareto optimal paths is the generalization [3, 4] of DIJKSTRA's algorithm: Each node $v \in V$ gets a number of multi-dimensional labels assigned, representing all Pareto paths to v. For the bicriteria case, [3] was the first presenting such a generalization, while [5] describes multi-criteria algorithms in detail. By this generalization, DIJKSTRA loses the label-setting property, i.e., now a node may be visited more than once. It turns out that a crucial problem for multi-criteria routing is the number of labels assigned to the nodes. The more labels are created, the more nodes are reinserted in the priority queue yielding considerably slow-downs compared to the single-criteria setup. In the worst case, the number of labels can be exponential in $|V|$ yielding impractical running times [3]. Hence, [3, 6] present an FPAS for the bicriteria shortest path problem.

Speed-up Techniques. Most of the work on speed-up techniques for multi-criteria scenarios was done on networks deriving from timetable information. In such networks, [7] observed that the number of labels is often limited such that the brute force approach for finding *all* Pareto paths is often feasible. Experimental studies finding all Pareto paths in timetable graphs can be found in [8–10]. However, to the best of our knowledge, all previous work only uses basic speed-up techniques for accelerating the multi-criteria query. In most cases a special version of A^* is adapted to this scenario. Unfortunately, the resulting speed-ups only reach up to a factor of 5 which is much less than for the (single-criteria) speed-up techniques developed during the last years.

1.2 Our Contribution

In this work, we present the first efficient speed-up technique for multi-criteria routing, namely an augmented version of SHARC [11]. Similar to the time-dependent version of SHARC [12], the key observation is that the basic concept of SHARC stays untouched. By augmenting the main subroutines of SHARC to multi-criteria variants and by changing the intuition when setting Arc-Flags [13, 14], we end up in a very efficient multi-criteria speed-up technique.

We start our work on multi-criteria routing with basic definitions in Section 2. We also shortly report how SHARC works in a single-criteria scenario. In Section 3, we show how the main ingredients of SHARC—DIJKSTRA's algorithm, contraction, and arc-flags—can be augmented such that correctness can be guaranteed in a multi-criteria scenario. It turns out that adaption of contraction is straight-forward, while for arc-flags, we have to alter the intuition of a `true` arc-flag slightly. In Section 4 we assemble our augmented ingredients to present a multi-criteria variant of SHARC. The key observation is that the basic concept of SHARC stays untouched, we only need to additionally augment the last ingredient, i.e., arc-flags refinement. This routine can be generalized by substituting local single-criteria DIJKSTRA-searches by multi-criteria ones.

The experimental evaluation in Section 5 confirms the excellent speed-up achieved by our multi-criteria variant of SHARC: The speed-up over the generalized DIJKSTRA's algorithm is the same as in a single-criteria scenario. However, it turns out that in road networks, multi-criteria searches yield too many possible routes to the target. Hence, we introduce several reasonable constraints how to prune unattractive paths both during preprocessing and queries. Here, the key observation is that we define a main metric (we use travel times) and only allow other paths if they do not yield too long of a delay. Moreover, we also introduce a constraint called *pricing*. Paths with longer travel times are only accepted if they yield significant improvements in other metrics. With these additional constraints we are able to compute reasonable Pareto paths in continental-sized road networks. In addition, we run experiments with similar metrics where we do not need the just mentioned constraints and also present results on synthetic data sets. We conclude our work with a summary and possible future work in Section 6.

2 Preliminaries

The main difference between single- and multi-criteria routing is that the labels assigned to edges contain more than one weight. In this work, we restrict ourselves to vectors in $\mathbb{R}^k_+$. Let $L = (w_1, \dots, w_k)$ and $L' = (w'_1, \dots, w'_k)$ be two labels. We use the following notation and operations in $\mathbb{R}^k_+$: L *dominates* another label L' if $w_i < w'_i$ holds for one $1 \leq i \leq k$ and $w_i \leq w'_i$ holds for each $1 \leq j \leq k$. The sum of L and L' is defined by $L \oplus L' = (w_1 + w'_1, \dots, w_k + w'_k)$. We call $\underline{L} = \min_{1 \leq i \leq k} w_i$ the minimum component of L, the maximum component $\overline{L}$ is defined analogously.

We also restrict ourselves to directed graphs $G = (V,E)$ with a length function $len : E \to \mathbb{R}^k_+$, assigning a k-dimensional label to each edge. Note that we allow multi-edges. The reverse graph $\overleftarrow{G} = (V, \overline{E})$ is the graph obtained from G by substituting each $(u,v) \in E$ by (v,u).

The 2-core of an undirected graph is the maximal node induced subgraph of minimum node degree 2. The 2-core of a directed graph is the 2-core of the corresponding simple, unweighted, undirected graph. All nodes not being part of the 2-core are called 1-shell nodes. Note that connected components within the 1-shell are trees. Since each tree is attached to the 2-core, we call these trees *attached trees*.

A *partition* of V is a family $\mathscr{C} = \{C_0, C_1, \dots, C_k\}$ of sets $C_i \subseteq V$ such that each node $v \in V$ is contained in exactly one set C_i. An element of a partition is called a *cell*. A *multilevel partition* of V is a family of partitions $\{\mathscr{C}^0, \mathscr{C}^1, \dots, \mathscr{C}^{L-1}\}$ such that for each $l < L-1$ and each $C^l_i \in \mathscr{C}^l$ a cell $C^{l+1}_j \in \mathscr{C}^{l+1}$ exists with $C^l_i \subseteq C^{l+1}_j$. In that case the cell C^{l+1}_j is called the *supercell* of C^l_i. The supercell of a level-$L-1$ cell is V. Note that the number of levels is denoted by L. We denote $\mathbb{c}_j(u)$ the level-j cell u is assigned to. The *boundary nodes* B_C of a cell C are all nodes $u \in C$ for which at least one node $v \in V \setminus C$ exists such that $(v,u) \in E$ or $(u,v) \in E$.

In a multi-criteria scenario, the length $d(s,t)$ of an s–t path $P = (e_1, \dots, e_r)$ is given by $len(e_1) \oplus \dots \oplus len(e_r)$. In contrast to a single-criteria scenario, many paths exist between two nodes that do *not* dominate each other. In this work, we are interested in the *Pareto-set* $\mathscr{D}(s,t) = \{d_1(s,t) \dots d_x(s,t)\}$ consisting of all non-dominated path-lengths $d_i(s,t)$ between s and t. We call $|\mathscr{D}(s,t)|$ the *size* of a Pareto-set. Note that by storing a predecessor for each d_i, we can compute all Pareto-paths as well.

SHARC-Routing. The original arc-flag approach [13, 14] first computes a partition $\mathscr{C}$ of the graph and then attaches a *bitvector* to each edge e. A bitvector contains, for each cell $C_i \in \mathscr{C}$, a flag $AF_{C_i}(e)$ which is true if a shortest path to a node in C_i starts with e.

A modified DIJKSTRA then only considers those edges for which the flag of the target node's cell is true. This idea was extended to a 2-level setup in [15]. Preprocessing of static SHARC [11] is divided into three sections. During the *initialization* phase, we extract the 2-core of the graph and perform a *multi-level* partition of G. Then, an *iterative* process starts. At each step i we first *contract* the graph by *bypassing* unimportant nodes and set the arc-flags *automatically* for each removed edge. In the contracted graph we compute the arc-flags of level i by growing a *partial* centralized shortest-path tree from each cell C_j^i. In the *finalization* phase, we assemble the output-graph, refine arc-flags of edges removed during contraction and finally reattach the 1-shell nodes removed at the beginning.

The query of static SHARC is a multi-level Arc-Flags DIJKSTRA adapted from the two-level Arc-Flags DIJKSTRA presented in [15]. The query is a modified DIJKSTRA that operates on the output graph. The modifications are as follows: When settling a node n, we compute the lowest level i on which n and the target node t are in the same supercell. When relaxing the edges outgoing from n, we consider only those edges having a set arc-flag on level i for the corresponding cell of t.

3 Augmenting Ingredients

From our augmentation of SHARC to a time-dependent scenario [12], we learned that it is sufficient to augment its ingredients, i.e., local DIJKSTRA-searches, arc-flags computation, and contraction. In this section we show how to augment all these ingredients such that correctness is guaranteed even in a multi-criteria scenario.

3.1 Dijkstra

Computing a Pareto set $\mathscr{D}(s,t)$ can be done by a straightforward generalization of DIJKSTRA's algorithm, as presented in [3, 4]. For managing the different distance-vectors at each node v, we maintain a list of labels $\texttt{list}(v)$. The list at the source node s is initialized with a label $d(s,s) = (0,\dots,0)$, any other list is empty. We insert $d(s,s)$ to a priority queue. Then, in each iteration step, we extract the label with the smallest minimum component. Then for all outgoing edges (u,v) a temporary label $d(s,v) = d(s,u) \oplus len(u,v)$ is created. If $d(s,v)$ is not dominated by any of the labels in $\texttt{list}(v)$, we add $d(s,v)$ to $\texttt{list}(v)$, add $d(s,v)$ to the priority queue, and remove all labels from $\texttt{list}(v)$ that are dominated by $d(s,v)$. We may stop the query as soon as $\texttt{list}(t) \neq \emptyset$ and all labels in the priority queue are dominated by all labels in $\texttt{list}(t)$.

Pareto Path Graphs. In the following, we construct *Pareto path graphs (PPG)* by computing $\mathscr{D}(s,u)$ for a given source s and all nodes $u \in V$, with our generalized DIJKSTRA algorithm. We call an edge (u,v) a *PPG*-edge if $L \in \texttt{list}(u)$ and $L' \in \texttt{list}(v)$ exist such that $L \oplus len(u,v) = L'$. In other words, (u,v) is a PPG-edge iff it is part of at least one Pareto-optimal path from s to v. Note that by this notion one edge of two parallel ones can be a PPG-edge while the other one is not.

3.2 Arc-Flags

In a single-criteria scenario, an arc-flag $AF_C(e)$ denotes whether e has to be considered for a shortest-path query targeting a node within C. In other words, the flag is set if e is important for (at least one target node) in C. In [12], we adapted Arc-Flags to a time-dependent scenario by setting a flag to true as soon as it is important for at least one departure time. The adaption to a multi-criteria scenario is very similar: we set an arc-flag $AF_C(e)$ to true, if e is important for at least one Pareto path targeting a node in C.

Unlike in the time-dependent scenario—where we needed approximations—we can settle for the straightforward approach for augmenting Arc-Flags. We build a Pareto path graph in $\overleftarrow{G}$ for all boundary nodes $b \in B_C$ of all cells C at level i. We stop the growth as soon as all labels in the priority queue are dominated by all labels $L(v,b)$ assigned to the nodes v in the supercell of C. Then we set $AF_C(u,v) = \texttt{true}$ if (u,v) is a PPG-edge for at least one PPG grown from all boundary nodes $b \in B_C$. Moreover, we set all own-cell flags to true.

Multi-Level Arc-Flags. SHARC is based on multi-level Arc-Flags. Hence, we need to augment the concept of multi-level Arc-Flags to a multi-criteria scenario. The augmentation is similar to the one to time-dependent networks. We describe a two-level setup which can be extended to a multi-level scenario easily.

Preprocessing is done as follows. Arc-flags on the upper level are computed as described above. For the lower flags, we grow a PPG in $\overleftarrow{G}$ for all boundary nodes b on the lower level. We may stop the growth as soon as all labels attached to the nodes in the supercell of C dominate all labels in the priority queue. Then, we set an arc-flag to true if the edge is a PPG edge of at least one Pareto path graph.

3.3 Contraction

Our augmented Pareto contraction routine is very similar to a static one: we first reduce the number of nodes by removing unimportant ones and—in order to preserve Pareto paths between non-removed nodes—add new edges, called shortcuts, to the graph. Then, we apply an edge-reduction step that removes unneeded shortcuts.

Node-Reduction. We iteratively *bypass* nodes until no node is *bypassable* any more. To bypass a node x we first remove x, its incoming edges I and its outgoing edges O from the graph. Then, for each combination of $e_i \in I$ and $e_o \in O$, we introduce a new edge with label $len(e_i) \oplus len(e_o)$. Note that we explicitly allow multi-edges. Also note that contraction gets more expensive in a multi-criteria scenario due to multi-edges. As for static node reduction, we use a heap to determine the next bypassable node. Let #*shortcut* be the number of *new* edges that would be inserted into the graph if x was bypassed and let $\zeta(x) = $#shortcut$/(|I| + |O|)$ be the *expansion* of node x. Furthermore, let $h(x)$ be the hop number of the hop-maximal shortcut. Then we set the key of a node x within the heap to $h(x) + 10 \cdot \zeta(x)$, smaller keys have higher priority. To keep the costs of shortcuts limited we do not bypass a node if its removal results in a hop number greater than h or an expansion greater than c. We say that the nodes that have been bypassed belong to the *component*, while the remaining nodes are called *core-nodes*.

Edge-Reduction. We identify unneeded shortcuts by growing a Pareto path graph from each node u of the core. We stop the growth as soon as all neighbors v of u have their final Pareto-set assigned. Then we may remove all edges from u to v whose label is dominated by at least one of the labels $\texttt{list}(v)$. In order to limit the running time of this procedure, we restrict the number of priority-queue removals to 1 000.

4 Multi-Criteria SHARC

With the augmented ingredients at hand, we are ready to augment SHARC. Remarkably, the augmentation is now very similar to time-dependent SHARC [12]. During perprocessing, we apply the augmented routines from Section 3 instead of their single-criteria counterparts, while the query is a modified multi-criteria DIJKSTRA pruning unimportant edges.

Preprocessing runs in several phase, explained in the following. During the *initialization* phase, we extract the 2-core of the graph and perform a multi-level partition of G according to an input parameter P. We can safely extract the 2-core since we can directly assign correct arc-flags to attached trees that are fully contained in a cell: Each edge targeting the 2-core gets all flags assigned true while those directing away from the 2-core only get their own-cell flag set true. By removing 1-shell nodes *before* computing the partition we ensure that an attached tree is fully contained in a cell by assigning all its nodes to the cell of its 2-core root. After the last step of our preprocessing we simply reattach the nodes and edges of the 1-shell to the output graph.

After the initialization, our *iterative* process starts. Each iteration step is divided into two parts: contraction and arc-flag computation. First, we apply a *contraction* step according to Section 3. In order to perserve correctness of multi-criteria SHARC, we have to use *cell-aware* contraction, i.e., a node u is never marked as bypassable if any of its neighboring nodes is *not* in the same cell as u. We have to set *arc-flags* for all edges of our output-graph, including those we remove during contraction. As for static SHARC, we can set arc-flags for all removed edges automatically. We set the arc-flags of the current and all higher levels depending on the tail u of the deleted edge. If u is a core node, we only set the own-cell flag to true (and others to false) because this edge can only be relevant for a query targeting a node in this cell. If u belongs to the component, all arc-flags are set to true as a query has to leave the component in order to reach a node outside this cell. Setting arc-flags of those edges not removed from the graph is more time-consuming since we apply the preprocessing of multi-level Arc-Flags from Section 3.

The *final* phase of our preprocessing-routine assembles the output graph. It contains the original graph, shortcuts added during preprocessing and arc-flags for all edges of the output graph. However, some edges may have no arc-flag set to true. As these edges are never relaxed by our query algorithm, we directly remove such edges from the output graph. Moreover, we improve on those flags set to true during the contraction process. by *Refinement of Arc-Flags*. This is achieved by propagating flags of edges outgoing from high-level nodes to those outgoing from low-level nodes. In a time-independent scenario [11], we grow shortest path trees to find the so called exit nodes of each node, while in a time-dependent scenario [12], we use profile graphs to determine these nodes. In our multi-criteria scenario, we now grow Pareto path graphs from each node. The propergation itself stays untouched, the only difference is that a node might have more than one predecessor, which all have to be examined when identifying the corresponding outgoing edge. Unfortunately, growing Pareto path graphs can get expensive. Hence, we limit the growth to $n\log(n)/|V_l|$, where V_l denotes the nodes in level l, priority-queue removals. In order to preserve correctness, we then may only propagate the flags from the exit nodes to u if the stopping criterion is fulfilled before this number of removals.

Query. Augmenting the SHARC-query is straightforward. For computing a Pareto-set $\mathscr{D}(s,t)$, we use a modified multi-criteria DIJKSTRA (Section 3) that operates on the output graph. The modifications are then the same as for the single-criteria variant of SHARC: When settling a node n, we compute the lowest level i on which n and the target node t are in the same supercell. Moreover, we consider only those edges outgoing from n having a set arc-flag on level i for the corresponding cell of t. In other words, we prune edges that are not important for the current query. The stopping criterion is the same as for a multi-criteria DIJKSTRA.

5 Experiments

In this section, we present our experimental evaluation. Our implementation is written in C++ using solely the STL at some points. As priority queue we use a binary heap. Our tests were executed on one core of an AMD Opteron 2218 running SUSE Linux 10.3. The machine is clocked at 2.6 GHz, has 16 GB of RAM and 2 x 1 MB of L2 cache. The program was compiled with GCC 4.2, using optimization level 3.

Inputs. We use four real world road networks for our experimental evaluation. The first one is the largest strongly connected component of the road network of Western Europe, provided by PTV AG for scientific use. It has approximately 18 million nodes and 42.6 million edges. However, it turns out this input is too big for finding all Pareto routes. Hence, we also use three smaller networks, namely the road network of Luxemburg consisting of 30 661 nodes and 71 619 edges, a road network of Karlsruhe and surrounding (77 740 nodes, 196 327 edges), and the road network of the Netherlands (892 392 nodes, 2 159 589 edges). Note that we use the latter network for testing the impact of our rules of label reduction. As metrics we use travel times for fast cars/slow trucks, costs (toll + fuel consumption), travel distances, and unit lengths. Note that the last metric depicts the number of street segments of a route. Hence, it somehow reflects the number of turns of a journey.

Default Setting. For Europe, we use a 6-level partition obtained by SCOTCH [16] with 4 cells per supercell on levels 0 to 3, 8 cells per supercell on level 4, and 104 cells on level 5. A 3-level partition is applied when using Luxemburg and Karlsruhe as input, with 4 cells per supercell on levels 0 and 1, and 56 cells on level 2. For the Netherlands, we apply a 4-level partition, with 4 cells per supercell on levels 0 and 1, 8 cells on level 2, and 112 cells on level 3. We use $c = 2.5$ as maximal expansions during node-reduction and for the all levels. The hop-bound of our contraction is set to $h = 10$. To keep preprocessing times limited, we use an economical variant, i.e., we compute arc-flags only for the topmost level and do not refine arc-flags for the lowest two levels. For static single-criteria SHARC, this reduces preprocessing times by a factor of 3, but query performance increases only be a factor of 2. In the following, we report preprocessing times and the overhead of the preprocessed data in terms of *additional* bytes per node. Moreover, we provide the average number of settled nodes, i.e., the number of nodes taken from the priority queue, and the average query time. For random s-t queries, the nodes s and t are picked uniformly at random. All figures in this paper are based on 1 000 random s-t queries and refer to the scenario that only distance labels of the Pareto paths have to be determined, without outputting a complete description of the paths. However, our efficient implementation for unpacking shortcuts due to [17] needs about 4 additional bytes per node of preprocessed data. Then it takes less than 0.5 ms to unpack a shortest path. Since we allow multi-edges we could apply this unpacking routine to our multi-criteria variant of SHARC.

Full Pareto-Setting. Table 1 depicts the performance of multi-criteria SHARC on our Luxemburg instance in a full Pareto bicriteria setting. For comparison, we also report the performance of single-criteria SHARC on all five metrics.

We observe a good performance of multi-criteria SHARC in general. Preprecessing times are less than 15 minutes which is sufficient for most applications. Interestingly, the speed-up over DIJKSTRA's algorithm with respect to query times even increases when switching to multi-criteria SHARC. However, comparing single- and multi-criteria, we

Table 1. Performance of single- and multi-criteria SHARC applying different metrics for our Luxemburg and Karlsruhe inputs. Column *prepro* shows the computation time of the preprocessing in hours and minutes and the eventual *additional* bytes per node needed for the preprocessed data. For queries, we report the number of labels created at the target node, the number of nodes removed from the priority queue, execution times in milliseconds, and speed-up over DIJKSTRA's algorithm.

	Luxemburg						Karlsruhe					
	PREPRO		QUERY				PREPRO		QUERY			
	time	space	target	#del.	time	spd	time	space	target	#del.	time	spd
metrics	[h:m]	[B/n]	labels	mins	[ms]	up	[h:m]	[B/n]	labels	mins	[ms]	up
fast car (fc)	< 0:01	12.4	1.0	138	0.03	114	< 0:01	12.4	1.0	206	0.04	188
slow truck (st)	< 0:01	12.6	1.0	142	0.03	111	< 0:01	12.7	1.0	212	0.04	178
costs	< 0:01	12.0	1.0	151	0.03	96	< 0:01	15.4	1.0	244	0.05	129
distances	< 0:01	14.7	1.0	158	0.03	87	< 0:01	15.7	1.0	261	0.06	119
unit	< 0:01	13.7	1.0	149	0.03	96	< 0:01	14.1	1.0	238	0.05	147
fc + st	0:01	14.7	2.0	285	0.09	100	0:01	15.3	1.9	797	0.26	108
fc + costs	0:04	24.1	29.6	4 149	6.49	263	1:30	26.6	52.7	15 912	80.88	184
fc + dist.	0:14	22.3	49.9	8 348	20.21	78	3:58	23.6	99.4	31 279	202.15	153
fc + unit	0:06	23.7	25.7	4 923	5.13	112	0:17	26.6	27.0	11 319	16.04	200
costs + dist.	0:02	20.4	29.6	3 947	4.87	119	1:11	21.9	67.2	19 775	67.75	160

observe that query performance highly depends on the size of the Pareto set at the target node. For similar metrics (fast car and slow truck), bicriteria queries are only 3 times slower than a single-criteria queries. This stems from the fact that the average size of the Pareto-set is only 2. If more labels are created, like for fast car + costs, multi-criteria queries are up to 673 times slower. Even worse, this slow-down increases even further when we apply our Karlsruhe network. Here, the queries are up to 3 366 times slower.

Summarizing, the number of labels created, and thus, the loss in query performance over single-criteria queries, is too high for using a full Pareto-setting for a big input like Western Europe. Hence, we show in the following how to reduce the number of labels such that "unimportant" Pareto-routes are pruned as early as possible.

Reduction of Labels. As observed in Tab. 1, the number of labels assigned to a node increases with growing graph size. In order to efficiently compute Pareto-paths for our European road network, we need to reduce the number of labels both during preprocessing and queries. We achieve this by tightening the definition of dominance. Therefore, we define the travel time metric to be the dominating metric W. Then, our tightened definition of dominance is as follows: Besides the constraints from Section 2, we say a label $L = (W, w_1, \ldots, w_{k-1})$ dominates another label $L' = (W', w'_1, \ldots, w'_{k-1})$ if $W \cdot (1 + \varepsilon) < W'$ holds. In other words, we only allow Pareto-paths which are up to ε times longer (with respect to the dominating metric). Note that by this notion, this has to hold for all sub-paths as well.

Table 2 reports the performance of bicriteria SHARC using the tightened definition of dominance (with varying ε) during preprocessing *and* queries. As input, we use three networks: Karlsruhe, the Netherlands, and Europe. We here focus on the probably most important combination of metrics, namely fast car travel time and costs). We observe that our additional constraint works: Preprocessing times decrease and query perfor-

Table 2. Performance of bi-criteria SHARC with varying ε using travel times and costs as metrics. The inputs are Karlsruhe, the Netherlands, and Europe.

	Karlsruhe				The Netherlands				Europe			
	PREP	QUERY			PREP	QUERY			PREP	QUERY		
ε	time [h:m]	target labels	#del. mins	time [ms]	time [h:m]	target labels	#del. mins	time [ms]	time [h:m]	target labels	#del. mins	time [ms]
0.000	< 0:01	1.0	265	0.09	0:01	1.0	452	0.21	0:53	1.0	3 299	2.6
0.001	< 0:01	1.1	271	0.09	0:01	1.1	461	0.21	1:00	1.1	3 644	4.1
0.002	< 0:01	1.1	302	0.10	0:01	1.2	489	0.22	1:03	1.2	4 340	7.1
0.005	< 0:01	1.3	307	0.11	0:01	1.4	517	0.24	1:18	1.4	5 012	11.3
0.010	< 0:01	1.5	322	0.11	0:01	1.7	590	0.27	1:58	2.4	9 861	19.2
0.020	< 0:01	1.9	387	0.13	0:01	2.2	672	0.32	4:10	5.0	24 540	48.1
0.050	< 0:01	2.5	495	0.18	0:02	3.3	1 009	0.51	14:12	23.4	137 092	412.7
0.100	< 0:01	4.2	804	0.33	0:04	4.8	1 405	0.82	>24:00	–	–	–
0.200	0:01	6.4	1,989	1.86	0:09	7.2	2 225	1.67				
0.500	0:02	14.0	3 193	3.61	0:39	12.8	4 227	4.85				
1.000	0:13	24.0	9 072	14.86	3:44	20.0	12 481	26.85				
∞	1:30	52.7	15 912	80.88	>24:00	–	–	–				

mance gets much better. However, as expected, very small ε values yield a small subset of the Pareto-set and high ε values yield high preprocessing times. For small and mid-size inputs, i.e., less than 1 million nodes, setting ε to 0.5 yields a reasonable amount of Pareto paths combined with good preprocessing times and good query performance. Unfortunately, for our European input, only $\varepsilon \leq 0.02$ yields practical preprocessing and query times.

Further Reduction. As observable in Tab. 2, our approach for reducing the number of labels is only practical for very small ε if we use Europe as input. As we are interested in paths with bigger ε values as well, we add another constraint, called *pricing*, in order to define dominance. Besides the constraints from Section 2 and from above, we say a label $L = (W, w_1, \dots, w_{k-1})$ dominates another label $L' = (W', w'_1, \dots, w'_{k-1})$ if $\sum_i w'_i / \sum_i w_i > W/W' \cdot \gamma$ holds for some constant γ. In other words, we only accept labels with longer travel times if this results in a decrease in the other metrics under consideration. With this further tightened definition of label dominance, we are finally ready to run multi-criteria queries on our European instance. Table 3 shows the performance of multi-criteria SHARC with varying γ in a bicriteria scenario (travel times + costs) for Europe. Note that we *fix* $\varepsilon = 0.5$. It turns out that our additional constraints work. With $\gamma = 1.0$, we create 5.3 labels in 35.42 ms on average at the target node, being sufficient for practical applications. Preprocessing times are still within reasonable times, i.e., less than 8 hours. If we want to generate more labels, we could set $\gamma = 0.999$. However, query times drop to almost 300 ms and preprocessing increases drastically. Summarizing, bicriteria queries for travel times and travel costs are possible if we use $\gamma = 1.0$ and $\varepsilon = 1.5$.

Table 3. Performance of bi-criteria SHARC with varying γ using travel times and costs as metrics. ε is fixed to 0.5. The input is Europe.

	PREPRO		QUERY		
	time	space	target	#del.	time
γ	[h:m]	[B/n]	labels	mins	[ms]
1.100	0:58	19.1	1.2	2 538	1.8
1.050	1:07	19.6	1.3	3 089	2.2
1.010	1:40	20.4	1.7	4 268	3.2
1.005	2:04	20.6	1.9	5 766	4.1
1.001	3:30	20.8	2.7	7 785	6.1
1.000	7:12	21.3	5.3	19 234	35.4
0.999	15:43	22.5	15.2	87 144	297.2
0.995	>24:00	–	–	–	–

Similar Metrics. Our last experiment for road networks deals with the following scenario. We are interested in the quickest route for different types of vehicles. Hence, we perform multi-criteria queries on metrics all based on travel times. More precisely, we use typical average speeds of fast cars, slow cars, fast trucks, and slow trucks. Due to the very limited size of the resulting Pareto-sets, we afford not to use our tightened definition of dominance for this experiment. Tab. 4 shows the performance of multi-criteria SHARC in such a single-, bi- and tri-, and quadro-criteria scenario.

We observe that a full Pareto-setting is feasible if metrics are similar to each other, mainly because the number labels is very limited. Interestingly, the speed-up of multi-criteria SHARC over multi-criteria DIJKSTRA is even *higher* than in a single-criteria scenario. The slow-down in preprocessing times and query performance is quite high but still, especially the latter is fast enough for practical applications. Quadro-criteria

Table 4. Performance of multi-criteria SHARC applying different travel time metrics. The inputs are the Netherlands and Europe.

	The Netherlands						Europe					
	PREPRO		QUERY				PREPRO		QUERY			
	time	space	target	#del.	time	speed	time	space	target	#del.	time	speed
metrics	[h:m]	[B/n]	labels	mins	[ms]	up	[h:m]	[B/n]	labels	mins	[ms]	up
fast car(fc)	0:01	13.7	1.0	364	0.11	1 490	0:25	13.7	1.0	1,457	0.69	7 536
slow car(sc)	0:01	13.8	1.0	359	0.10	1 472	0:24	13.8	1.0	1,367	0.67	7 761
fast truck(ft)	0:01	13.9	1.0	365	0.10	1 332	0:23	13.9	1.0	1,486	0.71	7 324
slow truck(st)	0:01	13.9	1.0	363	0.10	1 306	0:25	13.9	1.0	1,423	0.68	7 647
fc+st	0:05	16.2	2.2	850	0.33	2 532	2:24	18.3	3.8	6 819	4.35	12 009
fc+ft	0:05	16.2	2.0	768	0.29	2 371	1:30	18.3	3.2	5 466	3.91	11 349
fc+sc	0:05	15.5	1.2	520	0.19	1 896	1:08	17.1	2.0	4 265	2.26	10 234
sc+st	0:05	16.2	1.9	742	0.29	2 009	1:53	18.1	3.3	5 301	4.02	10 874
sc+ft	0:05	16.2	1.7	679	0.26	1 850	1:49	16.2	3.2	5 412	3.65	10 663
ft+st	0:05	15.7	1.3	551	0.21	1 692	1:28	17.4	3.0	5 157	3.73	12 818
fc+sc+st	0:06	19.0	2.3	867	0.37	2 580	2:41	20.3	4.5	6 513	5.70	12 741
fc+sc+ft	0:06	18.9	2.0	764	0.32	2 385	2:47	21.5	3.9	5 989	4.87	12 144
sc+ft+st	0:06	19.0	1.9	740	0.30	2 134	2:59	22.0	4.2	6 348	5.12	13 412
fc+sc+ft+st	0:07	21.8	2.5	942	0.43	2 362	4:41	24.5	6.2	12 766	7.85	15 281

queries need less than 8 ms for our European road networks, being sufficient for most applications. A generalized DIJKSTRA needs about 120 seconds on average for finding a Pareto-set in this quadro-criteria scenario. This speed-up of more than 15 000 is achieved by a preprocessing taking less than 5 hours.

6 Conclusion

In this work, we presented the first efficient speed-up technique for computing multi-criteria paths in large-scale networks. By augmenting single-criteria routines to multi-criteria versions, we were able to present a multi-criteria variant of SHARC. Several experiments confirm that speed-ups over a multi-criteria DIJKSTRA are at least the same as in a single-criteria scenario, in many cases the speed-up with respect to query times is even higher. However, if metrics differ strongly, the number of possible Pareto-routes increases drastically making preprocessing and query times impractical for large instances. By tightening the definition of dominance, we are able to prune unimportant Pareto-routes both during preprocessing and queries. As a result, SHARC provides a feasible subset of Pareto-routes in continental sized road network.

Regarding future work, one can think of other ways for pruning the Pareto-set. Maybe other constraints yield better subsets of the Pareto-set computable in reasonable time as well. An open challenging problem is the adaption of multi-criteria SHARC to a fully realistic timetable information system like the ones presented in [9, 10]. Another interesting question is how good alternative routes can be found in a single-criteria scenario.

References

1. Dijkstra, E.W.: A Note on Two Problems in Connexion with Graphs. Numerische Mathematik **1** (1959) 269–271
2. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Engineering Route Planning Algorithms. In Lerner, J., Wagner, D., Zweig, K.A., eds.: Algorithmics of Large and Complex Networks. Lecture Notes in Computer Science. Springer (2009) To appear. Online available at `http://i11www.ira.uka.de/extra/publications/dssw-erpa-09.pdf`.
3. Hansen, P.: Bricriteria Path Problems. In Fandel, G., Gal, T., eds.: Multiple Criteria Decision Making – Theory and Application –. Springer (1979) 109–127
4. Martins, E.Q.: On a Multicriteria Shortest Path Problem. European Journal of Operational Research **26**(3) (1984) 236–245
5. Theune, D.: Robuste und effiziente Methoden zur Lsung von Wegproblemen. PhD thesis (1995)
6. Warburton, A.: Approximation of Pareto Optima in Multiple-Objective Shortest-Path Problems. Operations Research **35**(1) (1987) 70–79
7. Müller–Hannemann, M., Weihe, K.: Pareto Shortest Paths is Often Feasible in Practice. In: Proceedings of the 5th International Workshop on Algorithm Engineering (WAE'01). Volume 2141 of Lecture Notes in Computer Science., Springer (2001) 185–197
8. Pyrga, E., Schulz, F., Wagner, D., Zaroliagis, C.: Efficient Models for Timetable Information in Public Transportation Systems. ACM Journal of Experimental Algorithmics **12** (2007) Article 2.4
9. Müller–Hannemann, M., Schnee, M.: Finding All Attractive Train Connections by Multi-Criteria Pareto Search. In: Algorithmic Methods for Railway Optimization. Volume 4359 of Lecture Notes in Computer Science. Springer (2007) 246–263
10. Disser, Y., Müller–Hannemann, M., Schnee, M.: Multi-Criteria Shortest Paths in Time-Dependent Train 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) 347–361
11. Bauer, R., Delling, D.: SHARC: Fast and Robust Unidirectional Routing. In Munro, I., Wagner, D., eds.: Proceedings of the 10th Workshop on Algorithm Engineering and Experiments (ALENEX'08), SIAM (April 2008) 13–26
12. Delling, D.: Time-Dependent SHARC-Routing. In: Proceedings of the 16th Annual European Symposium on Algorithms (ESA'08). Volume 5193 of Lecture Notes in Computer Science., Springer (September 2008) 332–343 Best Student Paper Award - ESA Track B.
13. Lauther, U.: An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background. In: Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung. Volume 22. IfGI prints (2004) 219–230
14. Hilger, M., Köhler, E., Möhring, R.H., Schilling, H.: Fast Point-to-Point Shortest Path Computations with Arc-Flags. In Demetrescu, C., Goldberg, A.V., Johnson, D.S., eds.: Shortest Paths: Ninth DIMACS Implementation Challenge. DIMACS Book. American Mathematical Society (2009) To appear.
15. Möhring, R.H., Schilling, H., Schütz, B., Wagner, D., Willhalm, T.: Partitioning Graphs to Speedup Dijkstra's Algorithm. ACM Journal of Experimental Algorithmics **11** (2006) 2.8
16. Pellegrini, F.: SCOTCH: Static Mapping, Graph, Mesh and Hypergraph Partitioning, and Parallel and Sequential Sparse Matrix Ordering Package (2007)
17. Delling, D., Sanders, P., Schultes, D., Wagner, D.: Highway Hierarchies Star. In Demetrescu, C., Goldberg, A.V., Johnson, D.S., eds.: Shortest Paths: Ninth DIMACS Implementation Challenge. DIMACS Book. American Mathematical Society (2009) Accepted for publication, to appear.