Customizable Contraction Hierarchies with Turn Costs

Valentin Buchhold
Karlsruhe Institute of Technology (KIT), Germany

Dorothea Wagner
Karlsruhe Institute of Technology (KIT), Germany

Tim Zeitz
Karlsruhe Institute of Technology (KIT), Germany

Michael Zündorf
Karlsruhe Institute of Technology (KIT), Germany

Abstract

We incorporate turn restrictions and turn costs into the route planning algorithm customizable contraction hierarchies (CCH). There are two common ways to represent turn costs and restrictions. The edge-based model expands the network so that road segments become vertices and allowed turns become edges. The compact model keeps intersections as vertices, but associates a turn table with each vertex. Although CCH can be used as is on the edge-based model, the performance of preprocessing and customization is severely affected. While the expanded network is only three times larger, both preprocessing and customization time increase by up to an order of magnitude. In this work, we carefully engineer CCH to exploit different properties of the expanded graph. We reduce the increase in customization time from up to an order of magnitude to a factor of about 3. The increase in preprocessing time is reduced even further. Moreover, we present a CCH variant that works on the compact model, and show that it performs worse than the variant on the edge-based model. Surprisingly, the variant on the edge-based model even uses less space than the one on the compact model, although the compact model was developed to keep the space requirement low.

2012 ACM Subject Classification Theory of computation → Shortest paths; Mathematics of computing → Graph algorithms; Applied computing → Transportation

Keywords and phrases Turn costs, realistic road networks, customizable contraction hierarchies, route planning, shortest paths

Digital Object Identifier 10.4230/OASIcs.ATMOS.2020.9

Supplementary Material We publish our extensions to the projects RoutingKit and InertialFlowCutter as pull requests on Github: `https://github.com/RoutingKit/RoutingKit/pull/77` and `https://github.com/kit-algo/InertialFlowCutter/pull/6`.

Acknowledgements We thank Peter Vortisch for providing the Stuttgart instance. We are also grateful to Transport for London (TfL) for permitting us to use their data, and to PTV AG for providing the London data. Further information about the London instance is provided by Tony Dichev (tonydichev@tfl.gov.uk).

1 Introduction

Motivated by computing driving directions, the last two decades have seen intense research on speedup techniques [3] for Dijkstra's shortest-path algorithm [15], which rely on a slow preprocessing phase to enable fast queries. Almost all previous experimental studies (e.g., [20, 23, 24, 14, 22, 1, 2, 4]) are restricted to the simplified model, where vertices represent intersections, edges represent road segments, and turn costs are ignored. While it has been widely believed that turn restrictions and turn costs are easy to incorporate, Delling et

© Valentin Buchhold, Dorothea Wagner, Tim Zeitz, and Michael Zündorf;
licensed under Creative Commons License CC-BY
20th Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems (ATMOS 2020).
Editors: Dennis Huisman and Christos D. Zaroliagis; Article No. 9; pp. 9:1–9:15

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

al. [11] show that most algorithms have a significant performance penalty. For long-range queries, one may argue that turn costs are negligible. When analyzing intracity traffic [8, 27] or dispatching autonomous vehicles operating within a particular city [6, 7], however, taking turn costs into account is of utmost importance.

A fairly recent development in the area of route planning are customizable speedup techniques, which split preprocessing into a slow metric-independent part, taking only the network structure into account, and a fast metric-dependent part (the *customization*), incorporating edge costs (the *metric*). Customizable route planning (CRP) [11] and customizable contraction hierarchies (CCH) [14] are the most prominent among them, and are both used in commercial and research software. While CRP was developed with turn costs in mind, CCH was not. In this work, we incorporate turn restrictions and turn costs into CCH.

Related Work. Turns can be encoded into the network structure by expanding the network so that road segments become vertices and allowed turns become edges [9, 28]. This is known as the *edge-based model* [3]. While any speedup technique can work on an expanded network, some are more robust than others [11]. We are aware of two algorithms that have been tailored to handle turns. First, Geisberger and Vetter [18] present a turn-aware version of (non-customizable) contraction hierarchies (CH) [17]. Second, Delling et al. [10] develop CRP with turns in mind. Both independently proposed a different turn representation. The *compact model* keeps intersection as vertices, but associates a *turn table* with each vertex.

Our Contribution. The contribution of this work is twofold. First, we propose several optimizations that accelerate CCH on the edge-based model by exploiting properties of the expanded network (Section 3). We reduce the increase in customization time from up to an order of magnitude to a factor of about three (which is reasonable since the expanded network is three times larger than the original network, which ignores turn costs). The increase in preprocessing time is reduced even further.

Second, we introduce a CCH variant that works on the compact model, and discuss various issues we found (Section 4). An extensive experimental evaluation shows that the edge-based variant significantly outperforms the compact variant (Section 5). Surprisingly, the variant on the edge-based model even uses less space than the one on the compact model.

Outline. Section 2 formally defines the problem we solve and has background information. Section 3 presents optimizations that accelerate CCH on the edge-based model. Section 4 introduces a CCH variant that works on the compact model. Section 5 presents an extensive experimental evaluation of both variants. Section 6 concludes with final remarks.

2 Preliminaries

We are given a directed graph $G = (V, E)$ where vertices represent intersections and edges represent roads. A cost function $\ell : E \to \mathbb{R}_{\geq 0}$ assigns an arbitrary cost to each edge. We are also given two functions $r : E \times E \to \{0, 1\}$ and $c : E \times E \to \mathbb{R}_{\geq 0} \cup \{\infty\}$. If $r(e, f) = 0$, the head of e is the tail of f and the turn from e to f is allowed. The cost of the turn is given by $c(e, f)$. Note that r and c have to be *consistent*, i.e., $r(e, f) = 1$ implies $c(e, f) = \infty$. Since r depends on the network topology, it is part of the input to the preprocessing phase. The turn cost function c is part of the input to the customization phase since it depends on the current traffic situation and personal preferences.

A path P from a point along an edge e_0 to a point along an edge e_k is a triple that consists of a sequence of edges $\langle e_0, \ldots, e_k \rangle$ with $r(e_i, e_{i+1}) = 0$, a real-valued offset $o_0 \in [0, 1]$ on e_0, and a real-valued offset $o_k \in [0, 1]$ on e_k. The cost of a path is the sum of the costs of its

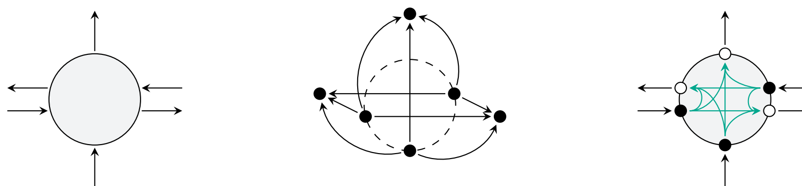

Figure 1 Turn representations (from left): simplified model, edge-based model, compact model.

constituent edges and turns, i.e., $\ell(P) = (1-o_0)\cdot\ell(e_0)+\sum_{i=1}^{k}(c(e_{i-1}, e_i)+\ell(e_i))-(1-o_k)\cdot\ell(e_k)$. Given a source edge e_s with offset o_s and a target edge e_t with offset o_t, the problem we consider is computing a shortest path from the start point along e_s to the end point along e_t. For simplicity, we assume that $o_s = 1$ and $o_t = 1$ in the rest of this paper.

In the following, we discuss both common ways to represent turn costs and restrictions. After that, we describe Dijkstra's algorithm and CH, both on standard graphs (simplified or edge-based graphs) and on compact graphs. We also discuss CCH on the simplified model.

2.1 Turn Representation

The *simplified model* ignores turn costs and restrictions; see Figure 1 (left). To actually incorporate them, there are two common ways. We explain each in turn.

Edge-based Model. The *edge-based model* [9, 28] expands the network so that road segments become vertices and allowed turns become edges; see Figure 1 (middle) for an example. More precisely, the edge-based graph $G_e = (V_e, E_e)$ is obtained from G as follows. The vertices of G_e are the edges of G, i.e, $V_e = E$. The edges of G_e are the allowed turns of G, i.e., $E_e = \{(e, f) : e, f \in E, r(e, f) = 0\}$. The cost of an edge $(e, f) \in E_e$ is defined as $\ell_e(e, f) = c(e, f) + \ell(f)$. The main advantage of the edge-based model is that most route planning algorithms can be used as is on it, without further modifications.

Compact Model. The *compact model* [18, 11] keeps intersections as vertices, but associates a $p \times q$ *turn table* T_v with each vertex v, where p and q are the numbers of incoming and outgoing edges, respectively. The entry $T_v(i, j)$ represents the cost of the turn from the i-th incoming edge e to the j-th outgoing edge f, i.e., $T_v(i, j) = c(e, f)$. For each edge (v, w), its tail corresponds to an *exit point* at v and its head corresponds to an *entry point* at w. Note that the entry points in the compact model translate directly to the vertices in the edge-based model; see Figure 1 (right) for an example. We denote by $v|i$ the i-th exit (or entry) point at v and by $(v|i, w|j)$ the edge whose tail corresponds to the i-th exit point at v and whose head corresponds to the j-th entry point at w. The main advantage of the compact model is its low space overhead since turn tables can be shared among vertices (the number of distinct turn tables for continental instances such as the road network of Western Europe used in our experiments is in the thousands rather than millions [11]).

2.2 Dijkstra's Algorithm

Dijkstra's algorithm [15] computes the shortest-path distances from a source vertex s to all other vertices. For each vertex v, it maintains a *distance label* $d(v)$, which represents the cost of the shortest path from s to v seen so far. Moreover, it maintains an addressable priority queue Q [25] of vertices, using their distance labels as keys. Initially, $d(s) = 0$ for the source s, $d(v) = \infty$ for each vertex $v \neq s$, and $Q = \{s\}$.

The algorithm repeatedly extracts a vertex v with minimum distance label from the queue and *settles* it by *relaxing* its outgoing edges (v, w). To relax an edge $e = (v, w)$, the path from s to w via v is compared with the shortest path from s to w found so far. More precisely, if $d(v) + \ell(e) < d(w)$, the algorithm sets $d(w) = d(v) + \ell(e)$ and inserts w into the queue. It stops when the queue becomes empty. Note that Dijkstra's algorithm has the label-setting property, i.e., each vertex is settled at most once. Therefore, when computing a point-to-point shortest path from a source s to a target t, we can stop when t is settled.

On the Compact Model. For correctness, we must work on entry points instead of vertices. That is, we maintain a distance label $d(v|i)$ for each entry point $v|i$ and a queue Q of unsettled entry points. Initially, $d(s|i) = 0$ for the entry point $s|i$ corresponding to the head of the source edge, $d(v|j) = \infty$ for all other entry points $v|j$, and $Q = \{s|i\}$. To settle an entry point $v|i$, we set $d(w|k) = \min\{d(w|k), d(v|i) + T_v(i, j) + \ell(e)\}$ for each outgoing edge $e = (v|j, w|k)$. Each entry point is settled at most once, however, each vertex can be visited multiple times. Note that Dijkstra's algorithm on the compact model essentially simulates the execution on the edge-based model.

2.3 Contraction Hierarchies

Contraction hierarchies (CH) [17] is a two-phase speedup technique to accelerate point-to-point shortest-path computations, which exploits the inherent hierarchy of road networks. To differentiate it from customizable CH, we sometimes call it *weighted* or *standard* CH. The preprocessing phase heuristically orders the vertices by importance, and *contracts* them from least to most important. Intuitively, vertices that hit many shortest paths are considered more important, such as vertices on highways. To contract a vertex v, it is temporarily removed from the graph, and *shortcut* edges are added between its neighbors to preserve distances in the remaining graph (without v). Note that a shortcut is only needed if it represents the only shortest path between its endpoints, which can be checked by running a *witness search* (local Dijkstra) between its endpoints.

The query phase performs a bidirectional Dijkstra on the augmented graph that only relaxes edges leading to vertices of higher *ranks* (importance). The stall-on-demand [17] optimization prunes the search at any vertex v with a suboptimal distance label, which can be checked by looking at upward edges coming into v.

On the Compact Model. Recall that we must maintain and compute distance labels for entry points (rather than vertices) in the compact model. Therefore, when contracting a vertex v, we need to preserve the distances between all entry points in the remaining graph (without v). In general, we cannot avoid self-loops and parallel edges. See Figure 2 for an example. Contracting vertex v' creates a self-loop at vertex v, because the through movement from v's left entry point to its right exit point is costlier than the path via v'. Analogously, contracting v results in two parallel edges between vertices u and w. When entering u from the west and leaving w to the east, the shortest path is via v. In contrast, when entering u from the north and leaving w to the north, it is better to traverse the edge between u and w.

Self-loops make the computation of shortcuts more complicated. Each shortcut is no longer a concatenation of exactly two edges, but can also include one or more self-loops at the middle vertex. For example, in Figure 2, the shortcut between u and w includes the self-loop at v. Therefore, Geisberger and Vetter [18] use the witness search not only to decide whether a shortcut is necessary but also to compute the cost of the shortcut.

More precisely, to contract a vertex v, they run a witness search for each exit point $u|i$ such that there is at least one incoming edge $(u|i, v)$. Initially, they set $d(v'|j) = \ell(e)$ for each edge $e = (u|i, v'|j)$. Moreover, each entry point $v'|j$ is inserted into the queue. The

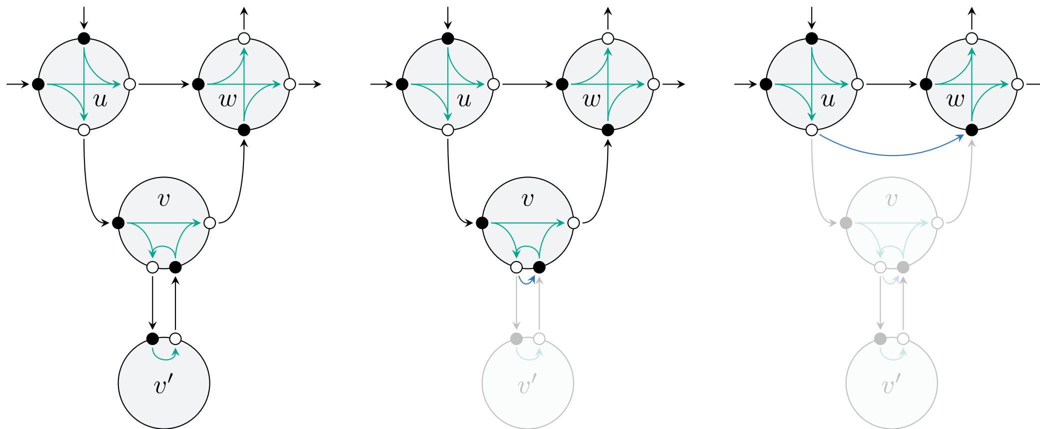

Figure 2 Vertex contraction on the compact model. Original edges are shown in black, turn edges are shown in green, and shortcut edges are shown in blue. Each original edge and each right-, left- and U-turn movement has cost 1. Each through movement has cost 10. Left: A subgraph before contraction. Middle: Contracting vertex v' creates a self-loop at v (cost 3). Right: Contracting v creates a shortcut edge between u and w (cost 7), resulting in two parallel edges between them.

witness search stops when each entry point $w|l$ such that there is at least one edge $(v, w|l)$ has been settled. A shortcut $s = (u|i, w|l)$ is only added if it is built from an edge $(u|i, v)$, zero or more self-loops at v, and an edge $(v, w|l)$. The shortcut has cost $\ell(s) = d(w|l)$.

The query phase runs a bidirectional version of the turn-aware Dijkstra described above, but does not relax edges leading to lower-ranked vertices. Note that the stall-on-demand optimization can also be applied in the compact model [18].

2.4 Customizable Contraction Hierarchies

Customizable contraction hierarchies (CCH) [14] are a three-phase technique, splitting CH preprocessing into a relatively slow metric-independent phase and a much faster customization phase. The metric-independent phase computes a *separator decomposition* [5] of the unweighted graph, determines an associated *nested dissection order* [19] on the vertices, and contracts them in this order without running witness searches (which depend on the metric). Therefore, it adds every potential shortcut. The customization phase computes the costs of the edges in the hierarchy by processing them in bottom-up fashion. To process an edge (u, w), it enumerates all triangles $\{v, u, w\}$ where v has lower rank than u and w, and checks if the path $\langle u, v, w\rangle$ improves the cost of (u, w). Alternatively, Buchhold et al. [8] enumerate all triangles $\{u, w, v'\}$ where v' has higher rank than u and w, and check if the path $\langle v', u, w\rangle$ improves the cost of (v', w), accelerating customization by a factor of 2.

There are two known query algorithms. First, one can run a standard CH search without modification. In addition, Dibbelt et al. [14] describe a query algorithm based on the *elimination tree* of the augmented graph. The parent of a vertex in the elimination tree is its lowest-ranked higher neighbor in the augmented graph. Bauer et al. [5] prove that the ancestors of a vertex v in the elimination tree are exactly the set of vertices in the CH search space of v. Hence, the elimination tree query algorithm explores the search space by traversing a path in the elimination tree, thereby avoiding a priority queue completely. Buchhold et al. [8] propose further optimizations for the elimination tree search, which achieve significant speedups for short- and mid-range queries.

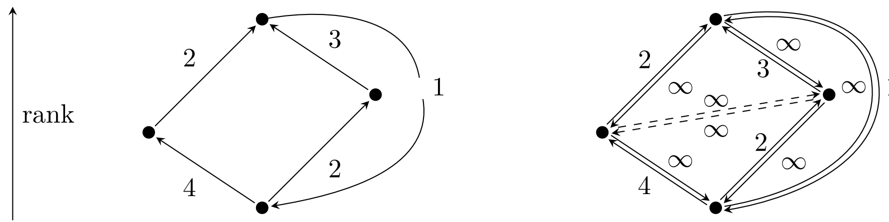

Figure 3 Original graph and final preprocessing result. The dashed shortcut has always weight ∞ in both directions and can be removed.

3 CCH on the Edge-based Model

CCH can be applied to the edge-based model without modifications. However, running times suffer significantly. We therefore propose optimizations to reduce the slowdown.

Contraction Order. Obtaining the nested dissection order is the most expensive part of preprocessing. We can apply the same ordering algorithms as for a nonturn CCH without modification to the edge-based graph. We refer to this order as the *edge-based order*. Since this approach is quite slow, we consider two additional ordering approaches.

Recall that the vertices of the expanded graph G_e are the edges of G. The *derived order* uses an order obtained on the nonturn graph and expands each vertex to all outgoing edges. Formally, the derived order is obtained by ordering the vertices of V_e by the rank of the tail of their corresponding edge in E.

We can also exploit the fact that vertices in G_e are edges in G and compute an edge order on G. Algorithms for obtaining separator decompositions in road networks like InertialFlow [26] and InertialFlowCutter [21] compute separators by finding a small balanced cut and deriving a separator from that cut. However, a cut in G corresponds directly to separator in G_e. Thus, we compute *cut-based orders* by computing a small balanced cut in G, using the nodes corresponding to the cut edges as the highest ranked vertices in the order for G_e and recursing on the sides of the cut. We extend InertialFlowCutter with this schema.

Infinite Shortcuts. CCH algorithms do not work on the original directed graph $G = (V, E)$, but on the corresponding bidirected graph $G' = (V, E')$ that is obtained from G by adding all edges $\{(w, v) : (v, w) \in E \wedge (w, v) \notin E\}$. This can lead to the insertion of unnecessary shortcuts; see Figure 3 for an example. We denote these unnecessary shortcuts as *infinite shortcuts* as the edges in both directions always have weight ∞. Infinite shortcuts can be identified by customizing the graph with the weight of every original edge set to zero. Afterwards, every bidirected edge that still has weight ∞ in both directions is an infinite shortcut and can be removed. We identify and remove infinite shortcuts in an additional preprocessing step, after obtaining the elimination tree.

Directed Hierarchies. In the simplified model, many edges have a corresponding reversed edge. This changes in the edge-based model and the amount of edges without a corresponding reversed edge increases. Then, the customized graph contains many edges with weight ∞ but a finite weight for the reversed edge. Like infinite shortcuts, these edges can be identified by customization with the zero metric. We remove these edges after obtaining the elimination tree. The result is a *directed hierarchy*. By enumerating lower triangles in both directions separately, the customization can be accelerated. As the elimination tree was computed on the bidirected graph before the edge removal, no adjustments are necessary for the query.

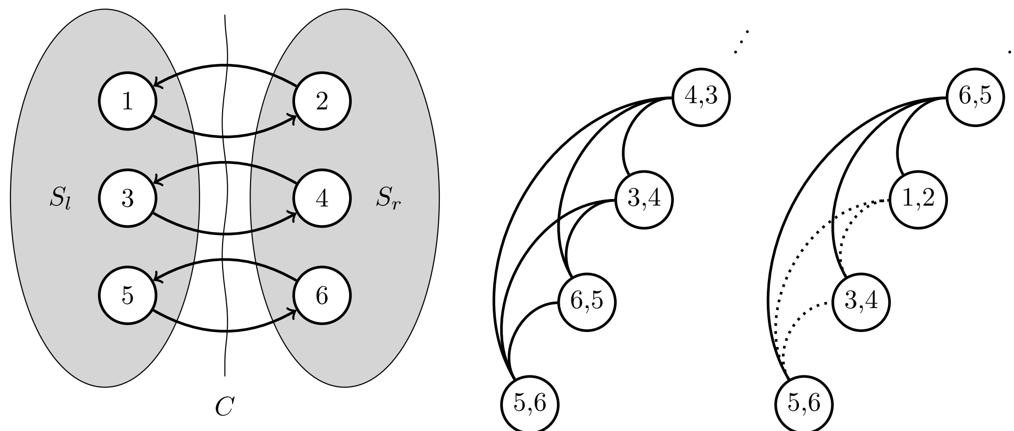

Figure 4 On the left is a visualization of a cut in G. In the middle is an arbitrary contraction order which results in no infinite edges after the first four contractions. On the right, the edges in the order are grouped which results in three infinite edges after the first four contractions (shown by the dotted edges).

Reordering Separator Vertices. In a nested dissection order, the vertices inside a separator can be ordered arbitrarily. We exploit this to generate more infinite shortcuts. Separator vertices in G_e correspond to cut edges in G. We order them by the side of their corresponding cut edges tail vertex. For example consider a cut in G with a left and a right side (Figure 4). Cut edges going from the left to the ride side (i.e. their respective vertices in G_e) will get the lower ranks, and cut edges from the right to the left will get the higher ranks. This way, shortcuts between the lower ranked vertices (left to right in the example) can never have a finite weight. Any directed path between them must use one of the higher ranked vertices (to go back from right to left). As shortcuts get the weight of the shortest path through lower ranked vertices, this will always be ∞ and these shortcuts can be removed later.

4 CCH on the Compact Model

Recall that all CCH phases do not work on the original directed graph $G = (V, E)$, but on the corresponding bidirected graph $G' = (V, E')$ that is obtained from G by adding all edges $\{(w, v) : (v, w) \in E, (w, v) \notin E\}$. The cost of each edge in $E' \setminus E$ is ∞, and thus the distance between any two vertices is the same in G and G'. Since most graph-theoretical results for undirected graphs carry over to bidirected graphs, CCH can use algorithmic tools for undirected graphs. In particular, CCH preprocessing exploits quotient graphs and CCH queries exploit elimination trees, which are both concepts for undirected graphs.

The compact model, however, is inherently directed. We cannot make a compact graph bidirected, since this would add edges that exit vertices at entry points and enter them at exit points. Therefore, in the compact model, all CCH phases have to work on the original (not necessarily bidirected) graph. This has undesirable consequences. First, we cannot use the efficient CCH preprocessing algorithm based on quotient graphs. Second, we have to use Dijkstra-based queries, since the faster elimination tree queries are also not applicable.

There is one additional issue. Recall that in the compact model, we generally cannot avoid self-loops and parallel edges and that each shortcut is no longer built from exactly two edges, but can also include one or more self-loops at the middle vertex. Standard CH (on the compact model) deals with this by using the witness searches to determine shortcut costs.

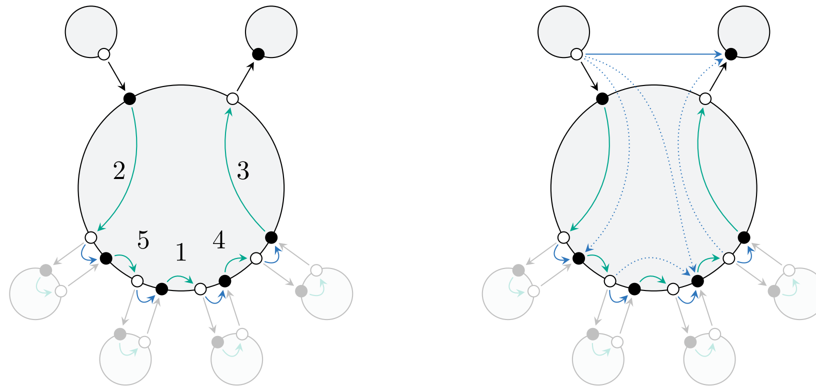

Figure 5 Creation of phantom shortcuts. We are about to contract the vertex in the center. Its lower-ranked neighbors (light-colored) are already contracted. Original edges are shown in black, turn edges are shown in green, and shortcut edges are shown in blue. Left: The vertex to be contracted and its neighbors before the contraction. The order on the turns is given by the numbers. Right: The shortcuts added while contracting the turns. Phantom shortcuts are drawn dotted.

During CCH customization, however, there is no notion of graph searches at all. We enumerate triangles and perform one basic operation for each triangle: adding up the costs of two edges to update the cost of the third edge. Hence, to determine the cost of a shortcut s containing self-loops, we must insert *phantom shortcuts*. These are used during customization to incrementally compute the cost of s by repeatedly combining two of its constituent edges.

Preprocessing. Given a nested dissection order on the vertices, we contract them in this order. When contracting a vertex v, we have to add a shortcut between each exit point $u|i$ with $u \neq v$ and $(u|i, v) \in E$ and each entry point $w|l$ with $w \neq v$ and $(v, w|l) \in E$. In addition, as already mentioned, we must add phantom shortcuts, so that the customization phase is able to compute the costs of shortcuts built from more than two edges incrementally.

Our approach is as follows. To contract a vertex v, we pick an order on the turns at v and contract them in this order. Consider a turn (j, k) at v. For each edge $(u|i, v|j)$ entering v at entry point j and each edge $(v|k, w|l)$ leaving v at exit point k, we add a shortcut $(u|i, w|l)$. Note that these shortcuts are concatenations of two edges, and thus their costs can be customized. If $u = v$ or $w = v$, then the shortcut is a phantom shortcut.

Note that this approach adds shortcuts that are not necessary. A shortcut $(u|i, w|l)$ is superfluous if $u = v$ and all turns entering exit point i are already contracted, or $w = v$ and all turns leaving entry point l are already contracted. To decide whether a shortcut is necessary, we maintain the number $t(\cdot)$ of uncontracted turns that enter or leave each exit or entry point of v, respectively. Whenever we contract a turn (j, k), we decrement both $t(j)$ and $t(k)$. A shortcut $(u|i, w|l)$ is only inserted if $u \neq v$ or $t(i) \neq 0$, and $w \neq v$ or $t(l) \neq 0$. Figure 5 illustrates the creation of phantom shortcuts.

Different turn orders can lead to slightly different numbers of phantom shortcuts. We thus tested some turn orders on various benchmark instances, however, the impact on the performance of all phases was limited. Therefore, any turn order that is easy to implement can be picked, in particular, the order in which the turns are stored in memory.

Customization. We recontract each turn, this time determining shortcut costs. Since we have the CCH topology in place, all we need to do to recontract a turn is to enumerate all triangles spanned by this turn and perform one minimum operation for each triangle.

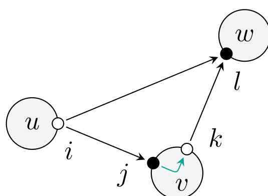

Figure 6 A triangle spanned by the turn (j, k) at v. Note that $(u|i, w|l)$ is the shortcut edge, and $(u|i, v|j)$ and $(v|k, w|l)$ are the supporting edges of the triangle.

Consider a turn (j, k) at a vertex v and a triangle consisting of three edges $(u|i, v|j)$, $(v|k, w|l)$ and $(u|i, w|l)$; see Figure 6. We call $(u|i, w|l)$ the *shortcut edge* and the other the *supporting edges* of the triangle. Also, we say that the turn *spans* the triangle.

We recontract the vertices in the given nested dissection order, and within each vertex, we recontract the turns in the same order as during preprocessing. If we pick the order in which the turns are stored in memory, we do not have to store the turn order for each vertex explicitly. For each turn at a vertex v, we enumerate the triangles spanned by the turn where v is the lowest-ranked vertex, and for each triangle, we add the costs of the two supporting edges and the turn between them, and update the cost of the shortcut edge if needed.

We now show how to efficiently enumerate all triangles spanned by a turn (j, k) where the shortcut edge does not point downwards. The other case is symmetric. We maintain a $|V| \times \Delta$ array W, where Δ is the maximum indegree of the original graph. All values in the array are initialized to ∞. First, we loop over all non-downward edges $e_2 = (v|k, w|l)$ leaving v at k and set $W[w, l] = T_v(j, k) + \ell(e_2)$. Then, we loop through all non-downward edges $e_1 = (u|i, v|j)$ entering v at j. For each such e_1, we loop through all non-downward edges $e = (u|i, w'|l')$ leaving u at i. If $\ell(e_1) + W[w', l'] < \ell(e)$, then we set $\ell(e) = \ell(e_1) + W[w', l']$. Finally, we loop over all edges e_2 again and reset $W[w, l]$ to ∞.

Interestingly, a nonturn version of this customization algorithm outperforms the original customization by Dibbelt et al. [14] by a factor of four, and is twice as fast as the engineered customization by Buchhold et al. [8]. The drawback is the increase in space consumption.

Queries. Dijkstra-based queries work as in standard CH on the compact model, however, they do not need to follow phantom shortcuts. Elimination tree queries are not applicable, since elimination trees are defined only for undirected graphs (and their bidirected counterparts).

5 Experiments

In this section, we present our experimental evaluation. Our benchmark machine runs openSUSE Leap 15.1 (kernel 4.12.14), and has 192 GiB of DDR4-2666 RAM and two Intel Xeon Gold 6144 CPUs, each of which has eight cores clocked at 3.5 Ghz and 8 × 64 KiB of L1, 8 × 1 MiB of L2, and 24.75 MiB of shared L3 cache. Hyperthreading was disabled and parallel experiments use 16 threads. Our code is written in C++ and compiled with GCC 8.2.1 using optimization level 3.

We implement our algorithms on top of the existing open-source libraries. For CCH, we use the implementation from RoutingKit[1]. We extend it by implementing customization for directed hierarchies and the removal of infinite edges. For the computation of contraction

[1] `https://github.com/RoutingKit/RoutingKit`

Table 1 Road networks used for the evaluation our algorithms. The turns column contains the number of allowed turns. It corresponds to the number of edges in the edge-based model. The number of vertices in the edge-based model is equal to the number of edges in the original graph.

	Source	Vertices [$\cdot 10^3$]	Edges [$\cdot 10^3$]	Turns [$\cdot 10^3$]	Turn data
Chicago	TransportationNetworks	13.0	39.0	135.3	100 s U-Turns
London	PTV	37.0	85.5	137.2	Costs, Restrictions
Stuttgart	PTV	109.5	252.1	394.2	Costs, Restrictions
Europe	DIMACS	17 350.0	39 936.5	106 371.3	100 s U-Turns

orders, we use InertialFlowCutter[2] [21] and implement the computation of cut-based orders and the reordering of separator vertices. We publish our extensions to these projects as pull requests on Github[3][4]. RoutingKit includes an implementation of InertialFlow [26] for the computation of contraction orders. We perform experiments with both InertialFlow and InertialFlowCutter. As InertialFlow is outperformed by InertialFlowCutter, our evaluation focuses on contraction orders obtained by InertialFlowCutter.

Inputs and Methodology. We perform experiments on several graphs with synthetic and real turn cost data. See Table 1 for an overview. We use three city-sized instances of the road networks of Chicago [16], London and Stuttgart. The London and Stuttgart instances were provided by PTV[5] with real turn restrictions and cost data. Our biggest benchmark instance is a graph of the road network of Western Europe made publicly available for the Ninth DIMACS implementation Challenge [13] with synthetic turn costs. To generate synthetic turn costs, we assign a travel time of 100 s to all U-turns. This number does not model a realistic time but a heavy penalty. All other turns are free. This model has been suggested in [11] and found to approximate real-world turn cost effects on the routing sufficiently well.

We perform experiments on the biggest strongly connected component of edge-based model representation of each graph and the induced subgraph on the original graph. Preprocessing running times are averages over 10 runs, customization running times averages over 100 runs. We utilize parallelization only for the preprocessing. All other phases are run sequentially. For the queries, we perform 1 000 000 point-to-point queries where both source and target are edges drawn uniformly at random. In the edge-based model, these edges correspond to vertices, which we select as source and target. For the original and compact graph, we use the head vertices of these edges.

Edge-based model. We evaluate the impact of different contraction orders on the performance of the different phases and the size of the augmented graph. Preprocessing includes both computing the order and the contraction but is dominated by the ordering. Table 2 depicts the results. Incorporating turns has a significant impact on the running time of all phases of CCH. The number of edges in the hierarchy grows at least by a factor of four to up to more than an order of magnitude. The derived order performs the worst on all instances.

[2] `https://github.com/kit-algo/InertialFlowCutter`
[3] `https://github.com/RoutingKit/RoutingKit/pull/77`
[4] `https://github.com/kit-algo/InertialFlowCutter/pull/6`
[5] `https://ptvgroup.com`

Table 2 Performance results for different contraction orders on each graph. We report the number of edges in the augmented graph and running times for preprocessing, customization, and queries. *Orig.* denotes the baseline on the nonturn/compact graph. The other three orders are for the edge-based model. *Deri.* indicates the derived order, *Edge* the order computed on the expanded graph, *Cut* the order obtained by ordering edges in the original graph.

		CCH Edges [$\cdot 10^3$]	Prepro. [s]	Custom. [ms]	Query [μs]
Chicago	Orig.	118	0.2	6	18
	Deri.	1 439	0.2	155	150
	Edge	819	1.1	50	60
	Cut	852	0.2	51	57
London	Orig.	182	0.3	7	20
	Deri.	1 199	0.3	85	111
	Edge	767	1.1	37	52
	Cut	840	0.3	40	51
Stuttgart	Orig.	362	0.5	11	16
	Deri.	2 145	0.6	94	79
	Edge	1 607	2.4	58	41
	Cut	1 680	0.9	60	37
Europe	Orig.	53 521	182.3	2 349	187
	Deri.	414 615	202.1	29 787	1 561
	Edge	311 213	2 321.1	14 787	524
	Cut	331 794	256.3	14 751	577

Table 3 Performance impact of different optimizations on each graph. We report the number of triangles enumerated during the customization as well as customization and query running times. All configurations use a cut-based contraction order. Directed hierarchies imply the removal of infinite shortcuts and reordering separator vertices builds on both directed hierarchies and the removal of infinite shortcuts.

		Triangles [$\cdot 10^6$]	Custom. [ms]	Query [μs]
Chicago	None	21.6	51	57
	Infinity	19.6	48	56
	Directed	13.3	28	41
	Reorder	8.2	20	31
London	None	12.9	40	51
	Infinity	11.0	36	51
	Directed	7.7	23	40
	Reorder	4.8	18	30
Stuttgart	None	11.4	60	37
	Infinity	8.5	53	37
	Directed	6.2	36	30
	Reorder	4.4	32	22
Europe	None	3 955.7	14 751	577
	Infinity	3 413.6	13 942	582
	Directed	2 319.7	9 590	407
	Reorder	1 514.2	8 180	306

On Chicago, the customization slows down by a factor of 25. On the other instances, the slowdown is about an order of magnitude. The slowdown for queries is not as strong but still significant (by a factor of 5 to 8). Only the preprocessing stays comparatively fast as it is dominated by the order computation, which can run on the unmodified original graph. We conclude that this approach is not feasible.

With the edge-based order, we achieve a better order at the cost of additional preprocessing time. The slowdown compared to a nonturn CCH is reduced to a factor of five for the customization phase, for queries to 2.5 to 3. However, preprocessing takes up to an order of magnitude longer. Orders computed by InertialFlow are generally worse than InertialFlowCutter orders (the customization is a factor 1.3 to 1.5 slower) and on graphs of the edge-based model this difference becomes even more pronounced (factor 1.3 to 2.8). Consequentially, we focus on InertialFlowCutter orders.

Cut orders achieve the best trade-off between the running times of the different phases. Customization and query performance is roughly the same as with an edge-based order. The preprocessing slowdown is well below a factor of two for all graphs. InertialFlowCutter has certain optimizations which find optimal vertex orders for certain subclasses of graphs. We did not implement these optimizations for cut-based orders. We expect that implementing them would close the gap in quality between edge-based and cut-based orders.

In Table 3, we report performance results depending on the additional optimizations applied. All configurations use cut-based orders. We also report the number of triangles enumerated during the customization as the triangle enumeration dominates the customization running time. The impact of the optimizations is similar across all instances. All optimizations combined roughly achieve a speedup of two on both customization and queries. Removing undirected infinite shortcuts alone yields only small improvements. Combining this with directed hierarchies and removing all directed infinite shortcuts has a much bigger impact. This impact can be further amplified by reordering separator vertices, which produces even more infinite shortcuts. Note that the work per triangle is different for directed hierarchies. For undirected hierarchies, each triangle will be enumerated once and both directed triangles will be relaxed at once. For directed hierarchies, however, both directions will be enumerated separately. Thus, for undirected hierarchies, the number of relaxation operations is twice the number of enumerated triangles and the reduction achieved by directed hierarchies even greater. It is noteworthy that even though our optimizations primarily aim for the customization running time, we also achieve a significant speedup for query running times. The removal of infinite edges also reduces the number of edges in the query search space.

Compact Model. We also evaluate the performance of CCH with the compact model. The implementation is considerable more complex than our optimizations for the edge-based model and sadly does not deliver competitive performance. As we cannot use the efficient quotient graph based contraction routine, preprocessing slows down by an order of magnitude as previously observed in [14]. For the Europe instance, the augmented graphs in the compact model and in the edge-based model contain a similar number of edges. The number of triangles, however, increases by a factor of 43. This leads to a slowdown of the customization by a factor of 34. Queries are even worse. The running time increases by a factor of 53. The reason for this slowdown are vertices with high degrees (several thousand edges) in high-level separators. This happens because we get shortcuts between almost all pairs of entry and exit nodes of separator vertices. When an entry node is popped from the queue, all outgoing edges of that vertex are relaxed. This leads to a tremendous amount of edge relaxations and the observed slowdown. On Stuttgart and London, the slowdowns are around factor 20.

Comparison with related work. Table 4 summarizes our results and depicts them in comparison to running times achieved by competing approaches as reported in [11]. The experiments were performed on the publicly available Europe instance which is the only instance also considered in related work. Our experiments were conducted on a newer machine. Thus, the absolute numbers are not perfectly comparable. Using the comparison methodology from [3], the numbers from [11] should scaled down by a factor of 0.79. We observe that incorporating turns has a strong impact on all algorithms except CRP. Dijkstra becomes at least 2.5 times slower. CH queries remain comparatively fast (at least on the edge-based model), but preprocessing slows down by more than an order of magnitude. The CRP nonturn variant is realized as free turns in the compact model which explains why incorporating turns leaves the performance unaffected. While CCH achieves faster running times than CRP in all phases on nonturn graphs, without our modifications, it is outperformed by CRP on graphs with turns. However, when using cut-based orders and all optimizations, CCH again outperforms CRP. CCH with the compact model is outperformed by the optimized edge-based variant in all phases. Note that both the CRP and CCH customization times can be further decreased through parallelization and by two related techniques known as microcode [12] (for CRP) and triangle preprocessing [14] (for CCH).

Table 4 Performance of Dijkstra, CH, CRP and CCH in the compact model, in the edge-based model as is and with our optimizations (Edge-based*) on Europe with and without turns. Preprocessing was executed in parallel, customization and query sequentially. For CH and CRP we list unscaled results as reported in [11].

		No turns			Turns			
		Prepro. [s]	Custom. [s]	Queries [ms]	Repr.	Prepro. [s]	Custom. [s]	Queries [ms]
Dijkstra		-	-	1 061.52	Edge-based	-	-	2 674.72
					Compact	-	-	12 699.32
CH	[11]	109	-	0.11	Edge-based	1 392	-	0.19
					Compact	1 753	-	2.27
CRP	[11]	654	10.55	1.65	Compact	654	11.12	1.67
CCH		182	2.35	0.19	Edge-based	2 321	14.79	0.52
					Edge-based*	256	8.18	0.31
					Compact	2 542	281.56	16.51

However, both techniques require significantly more space, and we choose not to use them to keep the space requirement low.

6 Conclusion

We incorporated turn costs and restrictions into CCH. We presented several straightforward yet effective optimizations that bring preprocessing and customization times on the expanded graph close to those achieved on the simplified graph. Preprocessing now takes similar time on the simplified and expanded graph, and customization on the expanded graph is only roughly three times slower (down from up to an order of magnitude, e.g., on Chicago).

Adapting CCH to the compact model was much harder. We observed that CCH and the compact model do not match well. CCH relies heavily on concepts for undirected graphs, whereas the compact model is inherently directed. Moreover, shortcuts built from more than two edges are an issue for CCH customization, where there is no notion of graph searches. Consequently, our experiments showed that the CCH implementation tailored to expanded graphs significantly outperforms the one for compact graphs.

References

1 Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck. A hub-based labeling algorithm for shortest paths in road networks. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 230–241. Springer, 2011. doi:10.1007/978-3-642-20662-7_20.

2 Julian Arz, Dennis Luxen, and Peter Sanders. Transit node routing reconsidered. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 55–66. Springer, 2013. doi:10.1007/978-3-642-38527-8_7.

3 Hannah Bast, Daniel Delling, Andrew V. Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F. Werneck. Route planning in transportation networks. In Lasse Kliemann and Peter Sanders, editors, *Algorithm Engineering:*

Selected Results and Surveys, volume 9220 of *Lecture Notes in Computer Science*, pages 19–80. Springer, 2016. doi:10.1007/978-3-319-49487-6_2.

4 Holger Bast, Stefan Funke, Peter Sanders, and Dominik Schultes. Fast routing in road networks with transit nodes. *Science*, 316(5824):566, 2007. doi:10.1126/science.1137521.

5 Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. Search-space size in contraction hierarchies. *Theoretical Computer Science*, 645:112–127, 2016. doi:10.1016/j.tcs.2016.07.003.

6 Joschka Bischoff and Michal Maciejewski. Simulation of city-wide replacement of private cars with autonomous taxis in Berlin. In Ansar-Ul-Haque Yasar and Jesús Fraile-Ardanuy, editors, *Proceedings of the 7th International Conference on Ambient Systems, Networks and Technologies (ANT'16)*, volume 83 of *Procedia Computer Science*, pages 237–244. Elsevier, 2016. doi:10.1016/j.procs.2016.04.121.

7 Joschka Bischoff, Michal Maciejewski, and Kai Nagel. City-wide shared taxis: A simulation study in berlin. In *20th IEEE International Conference on Intelligent Transportation Systems (ITSC'17)*, pages 275–280. IEEE Computer Society, 2017. doi:10.1109/ITSC.2017.8317926.

8 Valentin Buchhold, Peter Sanders, and Dorothea Wagner. Real-time traffic assignment using engineered customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 24(2):2.4:1–2.4:28, 2019. doi:10.1145/3362693.

9 Tom Caldwell. On finding minimum routes in a network with turn penalties. *Communications of the ACM*, 4(2):107–108, 1961. doi:10.1145/366105.366184.

10 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 376–387. Springer, 2011. doi:10.1007/978-3-642-20662-7_32.

11 Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. Customizable route planning in road networks. *Transportation Science*, 51(2):566–591, 2017. doi:10.1287/trsc.2014.0579.

12 Daniel Delling and Renato F. Werneck. Faster customization of road networks. In Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela, editors, *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *Lecture Notes in Computer Science*, pages 30–42. Springer, 2013. doi:10.1007/978-3-642-38527-8_5.

13 Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*. American Mathematical Society, 2009.

14 Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Customizable contraction hierarchies. *ACM Journal of Experimental Algorithmics*, 21(1):1.5:1–1.5:49, 2016. doi:10.1145/2886843.

15 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

16 Transportation Networks for Research Core Team. Transportation networks for research. URL: https://github.com/bstabler/TransportationNetworks.

17 Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, 2012. doi:10.1287/trsc.1110.0401.

18 Robert Geisberger and Christian Vetter. Efficient routing in road networks with turn costs. In Panos M. Pardalos and Steffen Rebennack, editors, *Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11)*, volume 6630 of *Lecture Notes in Computer Science*, pages 100–111. Springer, 2011. doi:10.1007/978-3-642-20662-7_9.

19 Alan George. Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363, 1973. doi:10.1137/0710032.

20 Andrew V. Goldberg and Chris Harrelson. Computing the shortest path: A* search meets graph theory. In *Proceedings of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'05)*, pages 156–165. SIAM, 2005.

21 Lars Gottesbüren, Michael Hamann, Tim Niklas Uhl, and Dorothea Wagner. Faster and better nested dissection orders for customizable contraction hierarchies. *Algorithms*, 12(9):1–20, 2019. doi:10.3390/a12090196.

22 Ron Gutman. Reach-based routing: A new approach to shortest path algorithms optimized for road networks. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04)*. SIAM, 2004.

23 Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling. Fast point-to-point shortest path computations with arc-flags. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 41–72. American Mathematical Society, 2009.

24 Ulrich Lauther. An experimental evaluation of point-to-point shortest path calculation on road networks with precalculated edge-flags. In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 19–39. American Mathematical Society, 2009.

25 Peter Sanders, Kurt Mehlhorn, Martin Dietzfelbinger, and Roman Dementiev. *Sequential and Parallel Algorithms and Data Structures – The Basic Toolbox*. Springer, 2019. doi:10.1007/978-3-030-25209-0.

26 Aaron Schild and Christian Sommer. On balanced separators in road networks. In Evripidis Bampis, editor, *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, volume 9125 of *Lecture Notes in Computer Science*, pages 286–297. Springer, 2015. doi:10.1007/978-3-319-20086-6_22.

27 Arne Schneck and Klaus Nökel. Accelerating traffic assignment with customizable contraction hierarchies. *Transportation Research Record*, 2674(1):188–196, 2020. doi:10.1177/0361198119898455.

28 Stephan Winter. Modeling costs of turns in route planning. *GeoInformatica*, 6(4):345–361, 2002. doi:10.1023/A:1020853410145.