Understanding Small Separators in Road Networks

Master's Thesis of

Samuel Born

At the Department of Informatics
Institute of Theoretical Informatics (ITI)

Reviewer: T.T.-Prof. Dr. Thomas Bläsius
Second reviewer: Dr. Torsten Ueckerdt
Advisors: Adrian Feilhauer
Michael Zündorf

January 1, 2025 – July 1, 2025

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself. I have not used any other than the aids that I have mentioned. I have marked all parts of the thesis that I have included from referenced literature, either in their original wording or paraphrasing their contents. I have followed the by-laws to implement scientific integrity at KIT.

Karlsruhe, July 1, 2025

. . . .

(Samuel Born)

Abstract

Zusammenfassung

Contents

1 Introduction **1**

1.1 Motivation 1

1.2 Contribution 1

1.3 Outline 1

2 Preliminaries **3**

2.1 Graph Theory 3

2.2 Graph Separators 3

2.3 Customizable Contraction Hierarchies 4

3 Experimental Analysis **9**

3.1 Planarity 9

4 Evaluation **15**

5 Conclusion **17**

5.1 Future Work 17

Bibliography **19**

1 Introduction

In this chapter, the motivation for this work is explained. We shortly introduce our contribution. Finally, a brief outline of the contents is given.

1.1 Motivation

In graph theory, a separator is a subset of vertices whose removal divides the graph into disconnected components of roughly equal size. The size of these separators significantly affects the performance of numerous algorithms, particularly those utilizing divide-and-conquer approaches. For instance, Tarjan's and Lipton's foundational work on planar graphs [LT77] demonstrates their utility in optimizing algorithms for many problems like e.g. maximum flow.

Empirical observations from the Customizable Contraction Hierarchies (CCH) paper indicate that road networks may possess remarkably small separators, on the order of $\mathcal{O}\left(n^{1/3}\right)$ [DSW16]. This finding contrasts with the $\mathcal{O}\left(n^{1/2}\right)$ separators typical of planar graphs [LT79], especially since road networks can be considered as nearly planar. In road networks, these separators enable the creation of effective node orderings, which are critical for the performance of search queries in advanced routing algorithms like CCH. This thesis seeks to uncover the properties responsible for the presence of such small separators in road networks. We aim to determine whether these separators stem from inherent graph characteristics, such as limited vertex degrees or sparsity, or from physical real-world features, such as borders, rivers, or a hierachical sturcutre. Gaining insight into these properties promises to advance our theoretical understanding and offers practical benefits, such as identifying new applications or generating comparable synthetic graphs. The exploration of small separators in road networks is particularly fascinating due to the rarity of other naturally occurring graph classes exhibiting separators smaller than those in planar graphs.

1.2 Contribution

todo

1.3 Outline

todo

2 Preliminaries

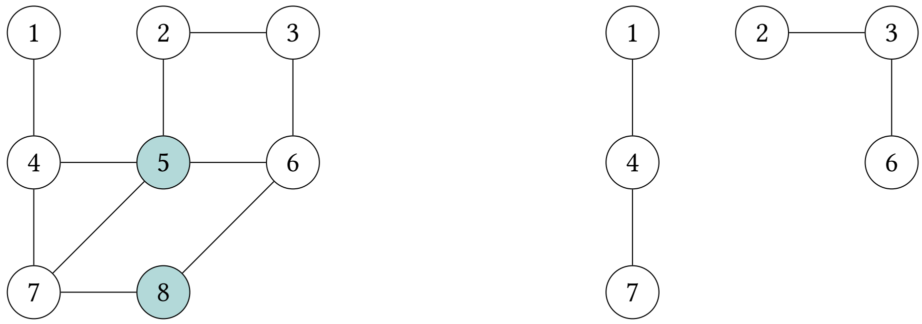
2.1 Graph Theory

Road networks can be modeled as graphs. A graph G is formally defined as a pair (V, E), where V represents a finite set of vertices (or nodes) and E represents a set of edges connecting pairs of vertices. In many applications, particularly route planning, graphs are augmented with a weight function $w : E \longrightarrow \mathbb{R}^+$, assigning a positive real value such as distance or travel time to each edge. However, for the purpose of this thesis, the topological strucuture of the graph is of primary interest, and we will not focus on edge weights. We will also only consider simple graphs, meaning graphs without multiple edges between the same pair of vertices and without edges connecting a vertex to itself (loops). Furthermore, as the concept of separators primarily applies to connectivity, we will consider undirected graphs, where edges represent symmetric relationships. An edge connecting vertices u and v in an undirected graph is denoted as the set $\{u, v\}$. The neighborhood of a vertex v is defined as the set of vertices adjacent to v, denoted as $N : V \longrightarrow \mathcal{P}(V)$. A *geometric graph* associates each vertex $v \in V$ of a graph $G = (V, E)$ with a unique point p in a specific geometric space, such as the Euclidean plane $\mathbb{R}^2$ or the surface of a sphere.

2.2 Graph Separators

A vertex separator (or simply separator) of a graph $G = (V, E)$ is a subset of vertices $S \subseteq V$ whose removal disconnects the graph into two or more components. More formally, the subgraph induced by $V \setminus S$, denoted $G[V \setminus S]$, is disconnected. For algorithmic applications, particularly divide-and-conquer strategies, balanced separators are crucial. A separator S is called α-balanced, for an $\alpha \in (0, 1)$, if removing S partitions the remaining vertices $V \setminus S$ into disjoint sets $V_1, V_2, \ldots, V_k$ such that no vertex in V_i is adjacent to a vertex in V_j for $i \neq j$, and the size of each resulting component V_i is bounded: $|V_i| \leq \alpha \cdot |V|$. A simple illustration of a balcseparator is shown in Figure 2.1. A common requirement is 2/3-balancedness, meaning each component contains at most 2/3 of the original graph's vertices. Balancedness ensures that recursive applications of the separator lead to subproblems of substantially smaller size, which is essential for the efficiency of algorithms based on this technique. Furthermore, minimizing the size of the separator S itself is critical for algorithmic performance. The size of the separator is typically evaluated asymptotically as a function of the number of vertices $n = |V|$ e.g. n^β for $\beta \in (0, 1)$.

The concept of *recursive* α-balanced separators extends this idea by ensuring that the property of finding small, balanced separators persists in the resulting subgraphs. Specifically, after removing an α-balanced separator S from G, each induced subgraph $G[V_i]$ (for $i = 1, 2, \ldots, k$) can itself be partitioned using another α-balanced separator of small size.



(a) G with separator $S = \{5, 8\}$ **(b)** $G[V \setminus S]$

Figure 2.1: Example of a well balanced separator in a graph. The vertices 5 and 8 form a balanced separator that disconnects the graph into two components.

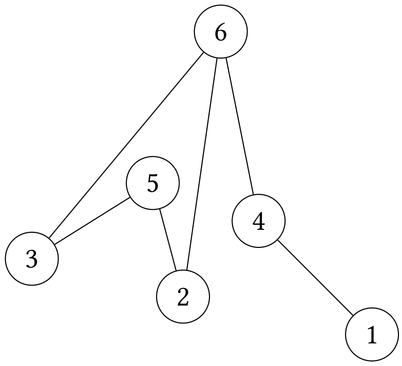
2.3 Customizable Contraction Hierarchies

Efficiently computing shortest paths in large graphs, such as continental road networks, is a fundamental problem. While Dijkstra's algorithm provides exact solutions for single-source shortest paths, its performance can be insufficient for real-time applications on large datasets. For instance, executing Dijkstra's algorithm on a graph representing the European road network can take over a second, primarily due to memory access latency rather than computational complexity alone. To accelerate query performance, many algorithms employ a two-phase approach: an initial precomputation phase followed by a query phase. This precomputation step processes the graph structure and edge weights to generate auxiliary data structures that enable faster subsequent queries.

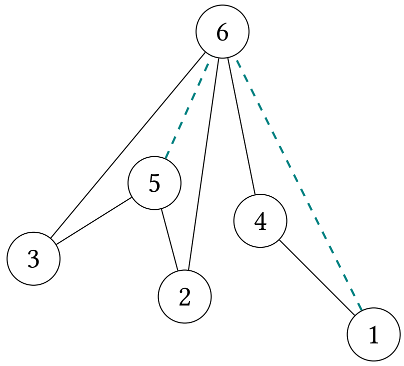
However, edge weights in real-world networks, particularly road networks, are often dynamic due to factors like traffic congestion. Standard two-phase approaches typically require re-running the entire, often time-consuming, precomputation phase whenever edge weights change. To address this limitation, three-phase approaches have been developed, separating the process into precomputation, customization, and query phases. The initial precomputation relies only on the graph's topology (nodes and edges), which is assumed to be relatively static. The second phase, customization, quickly incorporates the current edge weights into the precomputed structures. Finally, the query phase uses the customized data structures to answer shortest path requests rapidly.

Customizable Contraction Hierarchies (CCH) represent a prominent and effective three-phase route planning technique [DSW16]. CCH enables very fast customization, allowing adaptation to frequently changing edge weights, making it suitable for dynamic scenarios. The core idea underpinning CCH involves strategically inserting shortcut edges into the graph, analogous to the concept used in the original Contraction Hierarchies (CH) algorithm [GSSD08]. These shortcuts bypass sequences of original edges, effectively contracting the graph and speeding up queries. The efficiency of the CCH precomputation, particularly the node ordering it employs, can leverage the existence of small separators. We will now give a quick overview of the CCH algorithm.

Precomputation The CCH precomputation phase processes a graph $G = (V, E)$ based on a given vertex order [DSW16]. This order is defined by a bijection $\pi : \{1, \dots, n\} \to V$, where $n = |V|$. We will call the inverse π^{-1} rank, defined by the function $\text{rank} : V \to \{1, \dots, n\}$,



(a) Input graph. Already converted to be undirected and simple.



(b) Graph after precomputation, new shortcut edges are shown in teal.

Figure 2.2: Example of the CCH precomputation step. Nodes are named and positioned based on their rank.

where $\mathrm{rank}(v) = \pi^{-1}(v)$ assigns each vertex its position in the order. The determination of π is discussed separately. The core process involves iteratively contracting vertices $v_1, v_2, \ldots, v_n$, where $v_i = \pi(i)$. Contracting vertex v_i removes it from the current graph. For every pair of neighbors $u, w \in N(v_i)$, a shortcut edge (u, w). Resulting multi-edges between any pair (u, w) are simplified. The contraction process is illustrated in Figure 2.2.

Customization The customization phase integrates an edge weight function into the precomputed CCH supergraph $G_C = (V, E_C)$. During customization, edges in the supergraph E_C that correspond to edges in the original graph topology E are assigned weights according to the current metric being considered. Edges present only in E_C but not in E, which were introduced as shortcuts during the contraction process, are initially assigned an infinite weight. The primary objective of this phase is to establish the triangle inequality for the current edge weights within the CCH graph G_C.

To achieve this, the concept of a lower triangle is employed. Given an edge $\{x, y\} \in E_C$, a lower triangle is formed by the vertices $\{x, y, z\}$ if the edges $\{z, x\}$ and $\{z, y\}$ also exist in E_C, and $rank(z) < \min\{rank(x), rank(y)\}$. The customization algorithm iterates through the vertices of the graph in ascending order of their precomputed rank. For each vertex x, it considers all upward edges $\{x, y\}$ in the graph, where y is a neighbor of x and $rank(y) > rank(x)$. For every edge we determine all lower triangles $\{x, y, z\}$ that can be formed with x as the lowest ranked vertex. If the path through z offers a shorter connection, the weight of the edge $\{x, y\}$ is updated to this smaller value: $w(x, y) \leftarrow \min\{w(x, y), w(x, z) + w(z, y)\}$. The detailed procedure is outlined in the pseudocode presented in Algorithm 2.1. Figure 2.2 illustrates the customization process.

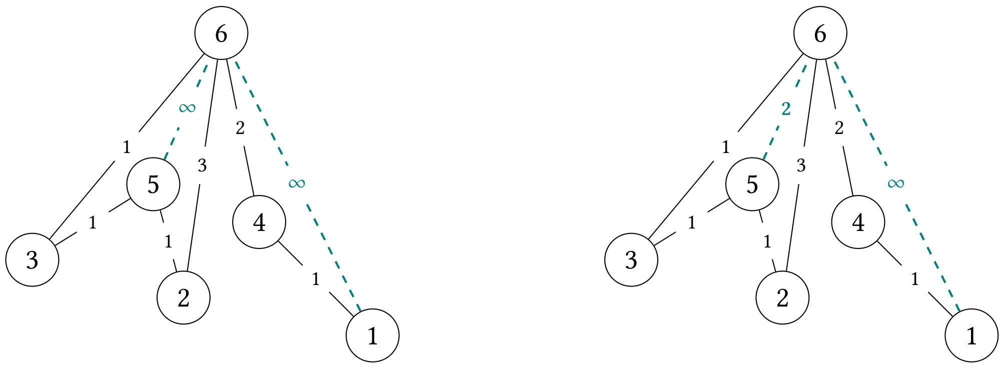
Note that the outlined algorithm only considers undirected edge weights, the algorithm can be extendend to directed edge weights. Details can be found in [DSW16].

Query To answer a shortest path query between a source node s and a target node t, the algorithm utilizes an implicit structure known as the elimination tree. This tree is defined on the nodes of the preprocessed CCH graph. Specifically, the parent of a node v in the elimination tree is the neighbor p of v in the CCH graph that has the lowest rank among all

Algorithm 2.1: CCH Customization

Input: $G_C = (V, E_C)$, node ordering π, edge weights w
Output: Customized CCH graph

forall x in V in ascending order of rank **do**
  forall upward edges $\{x, y\}$ in E_C **do**
    forall lower triangles $\{x, y, z\}$ associated with $\{x, y\}$ **do**
      $w(x, y) \longleftarrow \min\{w(x, y), w(x, z) + w(z, y)\}$



(a) Graph after precomputation. Weights are added to the edges. Shortcuts get weight ∞.

(b) Graph after customization. The shortcut edge $\{5, 6\}$ is updated to weight 2.

Figure 2.3: Example of the CCH customization step.

neighbors with a rank strictly greater than the rank of v. Figure 2.4 illustrates the elimination tree for the example graph shown in Figure 2.2. The query algorithm performs a bidirectional search upwards in this elimination tree, starting from s and t.

The core query process operates iteratively. Let u_s and u_t be the current nodes in the upward search originating from s and t, respectively; initially, $u_s = s$ and $u_t = t$. The algorithm proceeds as long as the search space needs exploration, effectively moving u_s and u_t towards the root of the elimination tree. In each step, the ranks of the current nodes u_s and u_t are compared. If u_s has a smaller rank than u_t, the algorithm relaxes all outgoing edges $\{u_s, v_i\}$ present in the (original, not the elimination tree) CCH graph. Subsequently, u_s is

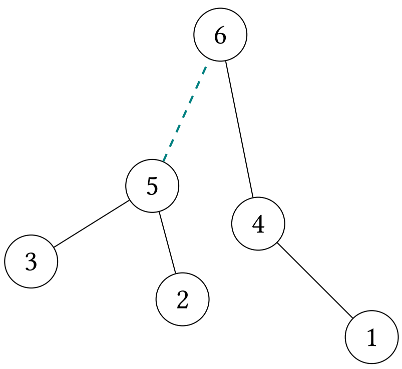


Figure 2.4: Elimination tree for the example graph in Figure 2.2.

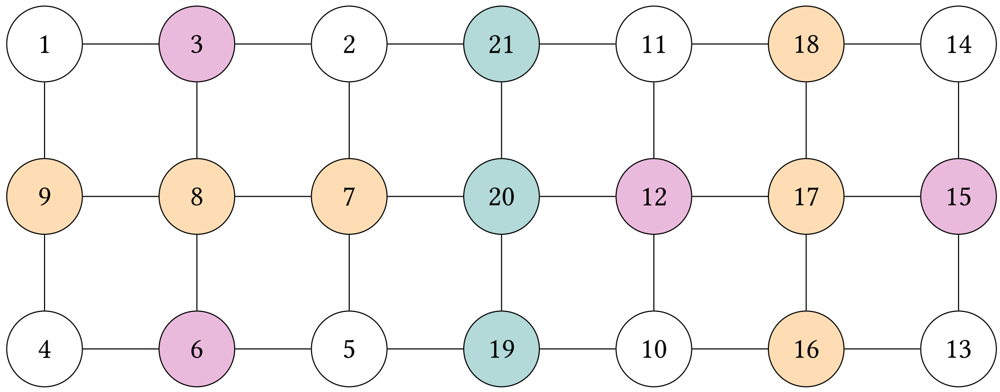


Figure 2.5: Example of a Nested Dissection. The top level separator is shown in teal, the second level in orange and the third level in purple. The nodes are named according to their rank in the resulting order.

updated to become its parent node in the elimination tree. Otherwise (if u_t has a rank less than or equal to that of u_s), the algorithm relaxes all outgoing edges $\{u_t, v_i\}$ existing in the CCH graph. Following the relaxation step, u_t is updated to its parent in the elimination tree. This process continues, effectively exploring paths upwards towards higher-ranked nodes. The algorithm ensures that the necessary parts of the search graph are explored by ascending the elimination tree structure. It has been proven that this query algorithm correctly computes the shortest path distance, although a detailed proof is beyond the scope of this thesis.

Nested dissection One method for generating the node ordering required for CCH are Nested Dissections, which rely on graph separators. The process begins by identifying a small, balanced separator in the graph. Nodes within this separator are conceptually removed, partitioning the graph into smaller components. These separator nodes are designated as high-rank nodes in the hierarchy and are consequently placed towards the end of the final node ordering. This procedure is then applied recursively to the remaining components. Figure 2.5 provides a visual representation of this recursive partitioning strategy.

The size of the separators found significantly impacts the efficiency of CCH queries. CCH queries restrict exploration to edges leading towards higher-ranked nodes (upward edges). Consider the separator identified at the highest level of the recursion, which might contain approximately n^β nodes, where n^β denotes the separator size. When a query initiates within a component defined by this separator, nodes located in other components cannot be reached without traversing downwards through a separator node, violating the upward search constraint. This containment effect applies recursively within the sub-components generated during the nested dissection. The sub components at recursion level i have at most $\alpha^i \cdot n$ nodes, where α is the balance factor of the separator and thus have separtors of size $(\alpha^i \cdot n)^\beta$. This leads to a complete search space of size:

$$\begin{aligned}
&\sum_{i=0}^{\infty}(\alpha^i \cdot n)^\beta \\
&= n^\beta \cdot \sum_{i=0}^{\infty} \alpha^{i \cdot \beta} \\
&= n^\beta \cdot \frac{1}{1-\alpha^\beta} && \text{Geometric series, since } \alpha \in (0,1) \\
&\in \mathcal{O}\left(n^\beta\right)
\end{aligned}$$

Thus, the performance of the CCH algorithm is directly linked to the ability to find small separators.

3 Experimental Analysis

3.1 Planarity

Road networks can be modeled as graphs that are nearly planar, meaning they can be embedded in the plane with only a small number of edge crossings. It is a well-known result in graph theory that planar graphs admit $\frac{2}{3}$-balanced separators of size $\mathcal{O}(\sqrt{n})$, where n denotes the number of vertices.

A relevant inquiry is whether the near-planarity of road networks is a critical feature that influences their structural properties, or if the occasional non-planar elements are merely incidental and do not substantially affect the network's overall characteristics. This prompts the question of how the separator sizes of road networks are affected when they are transformed into strictly planar graphs, for instance, by altering edges to eliminate crossings.

To study road networks as planar graphs, we represent roads as linear segments between points. At each intersection of these segments, a new vertex is introduced, and the original edges are replaced accordingly. This process transforms the graph into a planar form by eliminating crossings. For efficient execution, we utilize a spatial index that stores the bounding boxes of all edges. Under the assumption of short edges, this structure enables rapid identification of potential intersections by querying overlapping bounding boxes, followed by verification of actual crossings. While other algorithms exist, such as the Bentley-Ottmann algorithm [BO79], which is designed for general segment crossings, or a linear-time algorithm (e.g., as described in [EGS10]), which is tailored for graph structures with a sublinear number of edge crossings, we opted for this spatial index-based approach due to its simplicity and ease of implementation, especially since performance is not a critical concern in this context. Given that a single edge may intersect multiple times, we sort the intersection points along each edge and introduce new edges accordingly. Pseudo-code for this planarization algorithm is provided in Algorithm 3.1.

We applied this planarization method to real-world road networks. The Karlsruhe network, with approximately 120,000 nodes and 150,000 edges, revealed around 2,500 intersections, while the Germany network, comprising about 5.8 million nodes and 7.2 million edges, exhibited approximately 100,000 intersections. These figures slightly exceed the $\mathcal{O}(\sqrt{n})$ intersection counts reported in prior studies but remain within a similar magnitude [EGS10]. These differences could be explained by the unoptimal linear assumption of edges and might be mitigated by using a more modeled road network like OpenStreetMap.

Analysis of separator sizes showed minimal variation post-planarization. We identified $\frac{2}{3}$-balanced separators of size approximately $\mathcal{O}(n^(1/3))$, aligning with the values from non planar graphs. A comparison of the separator sizes in the planar and non-planar versions of the Germany network is depicted in Figure 3.1.

Our findings indicate that separators in non-planar road networks closely resemble those in their planarized versions. Frequently, non-planar separators are also separators in the planarized graph or can be adapted to planar ones with the addition of only a few nodes. This can be seen in Figure 3.4, which depicts a non-planar separator extended to be a separator in the planarized Karlsruhe network.

explain they can get larger and smaller

also say that separators can increase: simple example two disjoint components that get connected

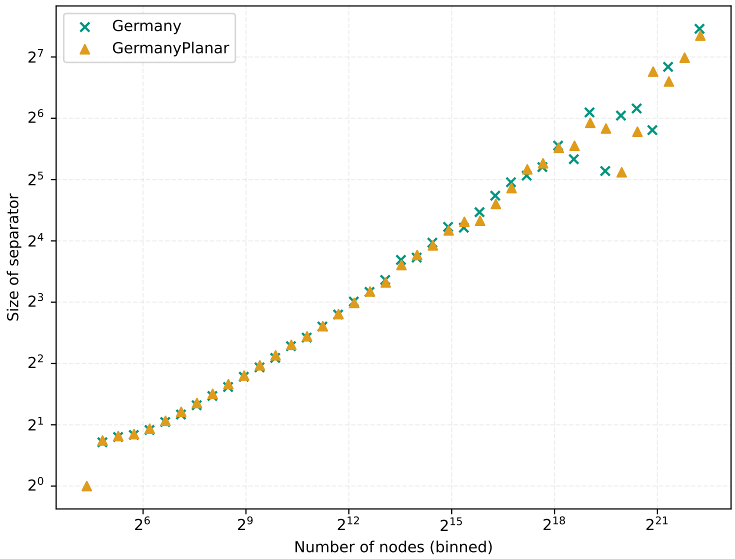
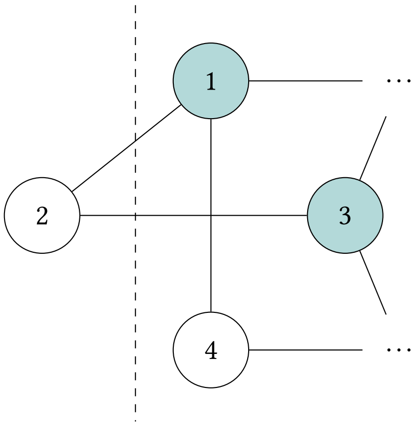


Figure 3.1: Comparison of separator sizes in the German road network: planar vs. non-planar.

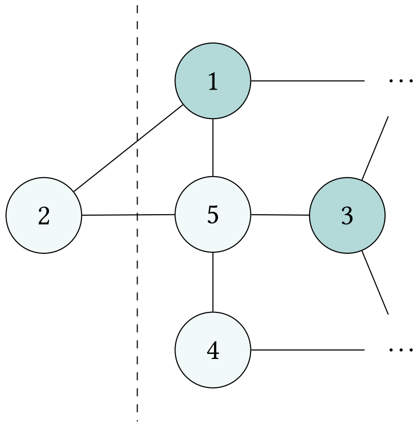
(a) Separator in non-planar graph

(b) Better separator in planarized graph

Figure 3.2: Example of a separator, where a better separator can be found in the planarized graph.



(a) Separator in non-planar graph



(b) The separator of the original graph is not a separator in the planarized graph.

Figure 3.3: Example where the separator of the original graph is not a separator in the planarized graph.

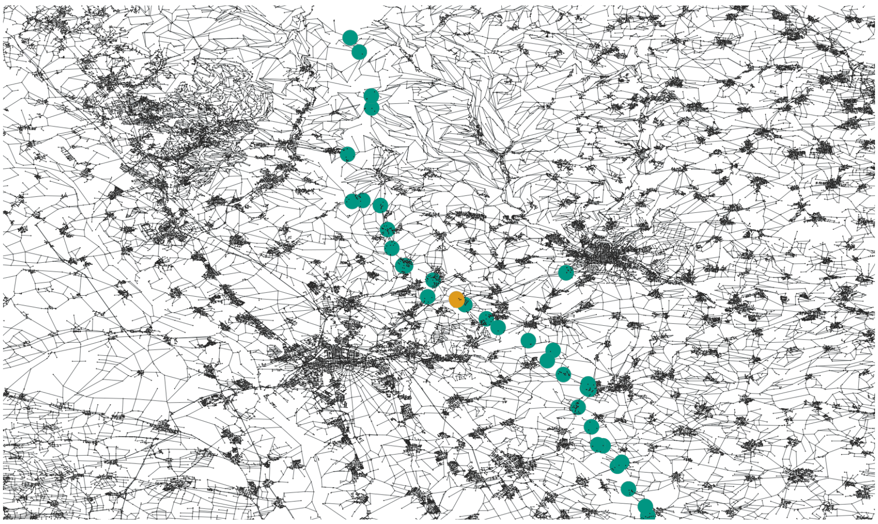


Figure 3.4: Example visualization of one possible top-level separator for Karlsruhe. Teal points represent the separator of the original graph, while orange points denote the additional nodes required to make it separator for the planarized version of Karlsruhe. Separators where computed with KaHIP.

These findings highlight that the near-planar structure of road networks has minimal impact on separator size, suggesting that such networks can typically be analyzed as planar graphs.

Algorithm 3.1: Simple planarization algorithm

Input: Non-planar graph $G = (V, E, pos)$.
Output: Planarized version of G.

```
spatial_index ⟵ load(bounding_boxes(E))
crossings ⟵ {}
forall e in E do
    forall candidates c in spatial_index.query(e) do
        if c intersects e then
            crossings[e].append(c)
            crossings[c].append(e)
forall (e, crossed) in crossings do
    G.remove(e)
    vertices ⟵ get_intersection_vertices(e, crossed)
    sort vertices along e
    add_new_edges(e, vertices)
```

4 Evaluation

5 Conclusion

5.1 Future Work

Bibliography

[BO79] Bentley and Ottmann. “Algorithms for Reporting and Counting Geometric Intersections”. In: *IEEE Transactions on Computers* Volume C-28 (Sept. 1979), pp. 643–647. ISSN: 0018-9340. DOI: *10.1109/TC.1979.1675432*.

[DSW16] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. “Customizable Contraction Hierarchies”. In: *ACM Journal of Experimental Algorithmics* Volume 21 (Nov. 4, 2016), pp. 1–49. ISSN: 1084-6654, 1084-6654. DOI: *10.1145/2886843*.

[EGS10] David Eppstein, Michael T. Goodrich, and Darren Strash. “Linear-Time Algorithms for Geometric Graphs with Sublinearly Many Edge Crossings”. In: *SIAM Journal on Computing* Volume 39 (Jan. 2010), pp. 3814–3829. ISSN: 0097-5397, 1095-7111. DOI: *10.1137/090759112*.

[GSSD08] Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. “Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks”. In: *Experimental Algorithms*. Edited by Catherine C. McGeoch. Vol. 5038. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 319–333. ISBN: 978-3-540-68548-7 978-3-540-68552-4. DOI: *10.1007/978-3-540-68552-4_24*.

[LT77] Richard J. Lipton and Robert Endre Tarjan. “Applications of a planar separator theorem”. In: *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*. 18th Annual Symposium on Foundations of Computer Science (sfcs 1977). Providence, RI, USA: IEEE, Sept. 1977, pp. 162–170. DOI: *10.1109/SFCS.1977.6*.

[LT79] Richard J. Lipton and Robert Endre Tarjan. “A Separator Theorem for Planar Graphs”. In: *SIAM Journal on Applied Mathematics* Volume 36 (Apr. 1979), pp. 177–189. ISSN: 0036-1399, 1095-712X. DOI: *10.1137/0136016*.