Karlsruhe Institute of Technology

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

fluff text

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 studies suggest road networks have balanced separators on the order of $\mathcal{O}(n^{1/3})$ [DSW16], which is significantly smaller than the $\mathcal{O}(n^{1/2})$ worst-case bound for planar graphs [LT79]. This is noteworthy given that road networks are can be treated as nearly planar due to their geographic nature (few crossings from overpasses/tunnels).

In road networks, these separators enable the creation of effective node orders, 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 hierarchical structure. 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.

Road networks represent an intriguing subject for the study of graph separators. Classical results within graph theory primarily establish balanced separators characterized by asymptotic sizes such as $\Theta(1)$, common in structures like trees, or $\Omega(\sqrt{n})$, e.g. planar graphs and unit disk graphs. To the best of our knowledge, established graph classes consistently exhibiting separator sizes strictly between these $\Theta(1)$ and $\Omega(\sqrt{n})$ bounds are not prominently featured in the graph theory literature. This finding positions road networks within this sparsely populated intermediate complexity range, thereby highlighting the compelling nature of investigating their structural properties.

1.2 Contribution

todo

1.3 Outline

todo

2 Preliminaries

fluff text

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 $N : V \longrightarrow \mathcal{P}(V)$.

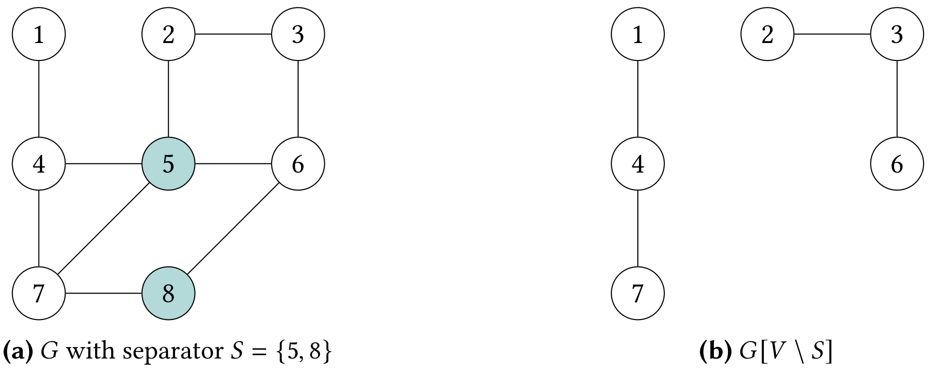
A graph *embedding* assigns each vertex $v \in V$ of a graph $G = (V, E)$ to a unique point p in a specific geometric space, such as the Euclidean plane $\mathbb{R}^2$ or the surface of a sphere, where edges are represented as straight line segments connecting the points corresponding to their incident vertices.

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.

Let $V_1, \ldots, V_k$ be the vertex sets corresponding to the connected components of the subgraph $G[V \setminus S]$. Most often, the removal of such a separator yields exactly two components ($k = 2$), as partitioning the graph into a larger number of components generally demands a larger separator. For a given constant $\alpha \in (0, 1)$, a separator S is termed α-balanced if the size of every resulting component V_i is bounded. Specifically, the condition $|V_i| \leq \alpha|V|$ must hold for all $i \in \{1, \ldots, k\}$. A simple illustration of a balanced separator 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 $\beta \in (0, 1)$.



(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.

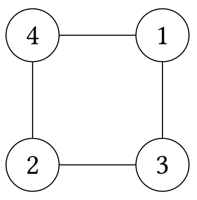
The concept of *recursive* α-balanced separators extends this idea by ensuring that the property of finding small, balanced separators persists in the resulting subgraphs. Specifically, after removing an α-balanced separator S from G, each induced subgraph $G[V_i]$ (for $i = 1, 2, \dots, k$) can itself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

To compute separators, various algorithms can be employed. In this thesis, we primarily utilize InertialFlowCutter [GHUW19]. This algorithm leverages geometric embeddings, often available for road networks, to compute high-quality node orderings efficiently. These node orderings serve as the basis for extracting separators from the graph using the method described below. However, InertialFlowCutter requires such a geometric embedding as input. In cases where a geometric embedding was not available, we utilized the KaHIP (Karlsruhe High Quality Partitioning) framework [SS13].

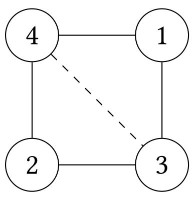
When using InertialFlowCutter, the resulting node ordering is interpreted as an elimination order for the vertices of the graph $G = (V, E)$. Based on this order, a chordal supergraph $G_C = (V, E \cup F)$ is constructed, where F represents the fill-in edges. The chordal supergraph is constructed by processing vertices v according to their rank. Fill-in edges are added such that for each vertex v, all its neighbors with a rank greater than $rank(v)$ form a clique. An efficient implementation connects the neighbor u_{min} with the minimum rank among those where $rank(u_{min}) > rank(v)$ to all other neighbors w also satisfying $rank(w) > rank(v)$. This suffices because the responsibility for adding edges between the remaining pairs of these higher-ranked neighbors w is effectively delegated to u_{min}.

Afterwards, a tree structure T can be constructed. Each node $v \in V$ selects its parent in the tree as the neighbor u that appears earliest in the elimination order among all neighbors w in G_C with $\text{rank}(w) > \text{rank}(v)$. If a node has no neighbors later in the order, it becomes the root.

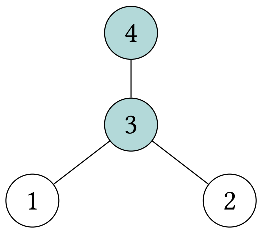
Separators in the original graph G can be derived from this tree structure using a traversal algorithm. The fundamental idea is to identify paths representing non-branching segments of the tree. Starting from a node r (representing the current subgraph), the traversal follows a path $P = (v_1 = r, v_2, \dots, v_k)$ downwards, where each node v_i $(1 \leq i < k)$ has exactly one child v_{i+1} in the tree. The path ends at node v_k, which is the first node encountered that does not have exactly one child (i.e., it has zero or multiple children). The set of vertices on this path, $S = \{v_1, v_2, \dots, v_k\}$, forms a separator. Its size is k, the number of nodes on the path. The traversal algorithm continues recursively into these subtrees. An overview of this process is illustrated in Figure 2.3.



(a) Original graph G



(b) Chordal supergraph based on the order



(c) Elimination tree T, Separator {4,3} highlighted

Figure 2.2: Example Process of deriving a separator from a node order. Node labels in indicate their rank in the node order.

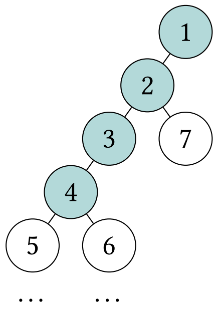


Figure 2.3: Separator identification with a significance threshold. The path extends downwards from node 1. At node 2, child 7 represents an insignificant subgraph (below the threshold), so the traversal continues via the single significant child path towards node 3. Node 4 is the first node encountered with two children (5 and 6) that both represent significant subgraphs. Therefore, the process stops here, and the identified separator is S = {1, 2, 3, 4}.

To ensure separators yield balanced partitions, the extraction process is refined using a 'significance threshold' based on relative subgraph size. Child nodes are only considered 'significant' if the subgraph they represent exceeds this threshold (e.g., contains at least 5% of the nodes in the parent's subgraph). When tracing a potential separator path $P = (v_1, \ldots, v_k)$ downwards, the path extends from v_i to v_{i+1} only if v_{i+1} is the *single* significant child of v_i. The path $S = \{v_1, \ldots, v_k\}$ is finalized as the separator upon reaching the first node v_k that possesses *two or more* significant children. This ensures separators correspond to meaningful branches in the tree structure concerning substantial graph parts. Figure 2.2 illustrates an example of this process.

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. 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.

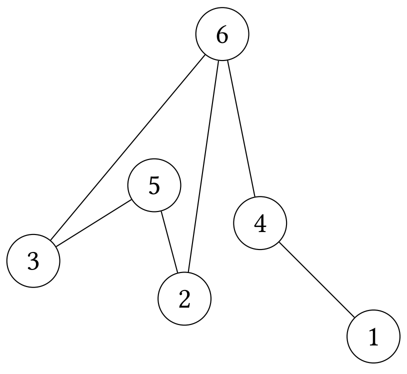
However, edge weights in real-world networks, particularly road networks, are often dynamic due to factors like traffic congestion. Standard two-phase approaches typically require re-running the entire, often time-consuming, precomputation phase whenever edge weights change. To address this limitation, three-phase approaches have been developed [DGPW11], 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 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.

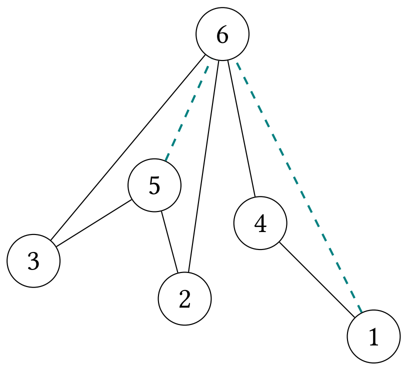
Precomputation The CCH precomputation phase introduces shortcut edges based on a given vertex order. These shortcuts effectively bypass sections of the graph, allowing algorithms to skip over entire subgraphs, unless the target node resides within such a subgraph. Furthermore, the specific process of inserting shortcuts based on the contraction order guarantees that any shortest path in the original graph corresponds to an 'up-down' path in the hierarchy defined by the vertex ranks [GSSD08]. An 'up-down' path consists of a sequence of edges leading to vertices with increasing ranks (the 'up' segment), followed by a sequence of edges leading to vertices with decreasing ranks (the 'down' segment). This property enables efficient bidirectional search by restricting exploration to higher-ranked neighbors.

This order is defined by a bijection $\pi : \{1, \dots, n\} \rightarrow V$, where $n = |V|$. We will call the inverse of this order rank : $V \rightarrow \{1, \dots, n\}$, which assigns each vertex its position in the order. The core process involves iteratively contracting vertices in order of their rank, starting from rank 1 up to n. Contracting a vertex v_i involves removing it and its incident edges from the current graph representation. For every pair of (higher ranked) neighbors $u, w \in N(v_i)$, a shortcut edge (u, w) is introduced. Resulting multi-edges are simplified. We call the resulting graph $G_C = (V, E_C)$, where $E_C = E \cup F$ and F represents the set of shortcut edges. The contraction process is illustrated in Figure 2.4.

A primary objective when selecting the vertex order is to minimize the number of shortcut edges introduced during the contraction process. Minimizing shortcuts is beneficial for both storage and query efficiency [DSW16]. However, solely minimizing the number of added shortcuts may not be sufficient in all cases. Different heuristics for selecting the contraction order exist.



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



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

Figure 2.4: Example of the CCH precomputation step. Nodes are named and positioned based on their rank.

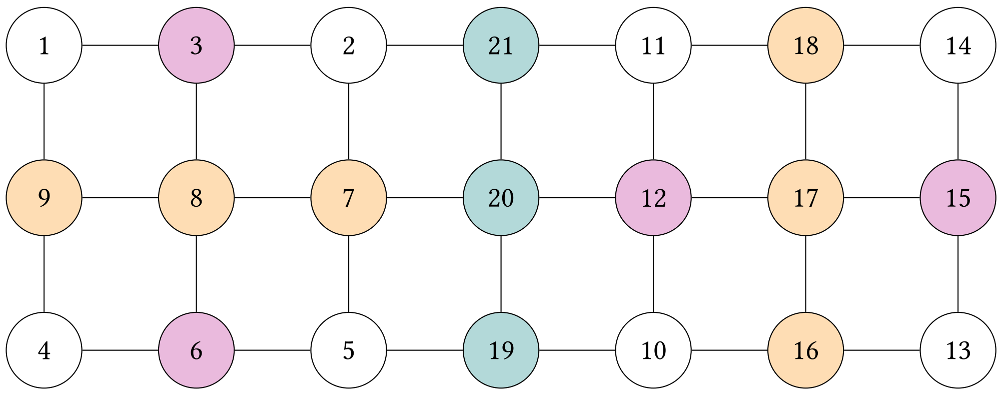
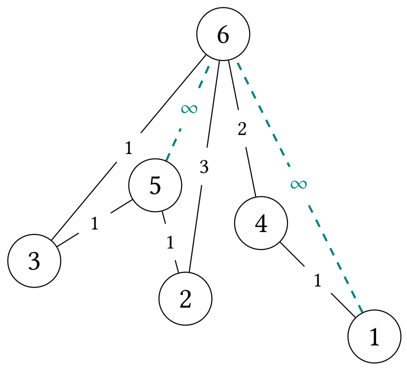


Figure 2.5: Example of a Nested Dissection. The top level separator is shown in teal, the second level in orange and the third level in purple. The nodes are named according to their rank in the resulting order.

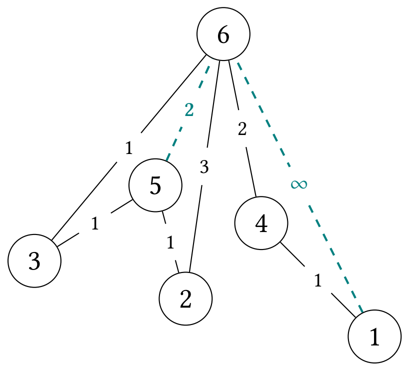
Nested Dissection One method for computing good vertex orders are Nested Dissections. The process begins by identifying a small, balanced separator in the graph. Nodes within this separator are conceptually removed, partitioning the graph into smaller components. These separator nodes are designated 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.

Customization Customization assigns the current metric's weights to the original edges within the CCH supergraph G_C and initializes shortcut edge weights to infinity. Following this initialization, edge weights are systematically updated to ensure the triangle inequality holds throughout 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, 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



(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 2.6: Example of the CCH customization step.

considers all upward edges $\{x, y\}$ in the graph, where y is a neighbor of x and $rank(y) > rank(x)$. For every such edge $\{x, y\}$ we determine all lower triangles $\{x, y, z\}$. 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. An illustration of the customization process is provided in Figure 2.4.

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].

Algorithm 2.1: CCH Customization

```
Input:  $G_C = (V, E_C)$, node ordering $\pi$, edge weights $w$
Output: Customized CCH graph
1 forall $x$ in $V$ in ascending order of rank do
2     forall upward edges $\{x, y\}$ in $E_C$ do
3         forall lower triangles $\{x, y, z\}$ associated with $\{x, y\}$ do
4             $w(x, y) \longleftarrow \min\{w(x, y), w(x, z) + w(z, y)\}$
```

Query To answer a shortest path query between a source node s and a target node t, the algorithm utilizes an structure known as the elimination tree. The elimination tree is defined on the nodes of G_C. 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 neighbors with a rank strictly greater than the rank of v. Figure 2.7 illustrates the elimination tree for the example graph shown in Figure 2.4. 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 until the root of the elimination tree is reached. 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 G_C. Subsequently, u_s is updated to become its parent node in the elimination tree. Otherwise (if u_t has a rank less than or equal to that of

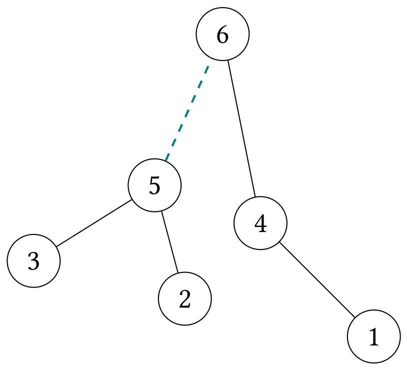


Figure 2.7: Elimination tree for the example graph in Figure 2.4.

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 correctness of this query algorithm for computing shortest path distances has been established; a detailed proof, which is beyond the scope of this thesis, can be found in [DSW16].

Complexity 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 contains approximately n^β nodes. 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. Let α denote the balance factor. The sub-components at recursion level i consequently have size at most $\alpha^i \cdot n$. Analyzing the total bound of the search space involves summing these separator sizes across the finite levels i of the recursion. This sum can be bounded by approximating it with the corresponding infinite geometric series [BCRW16]:

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

This analysis demonstrates that the total search space size explored during a CCH query is bounded by $\mathcal{O}(n^\beta)$, under the assumption that small separators can be found recursively. Note that we do not have worst case guarantees for the size of the separators in real road networks, but we can expect them to be small in practice. Thus, the performance of the CCH algorithm is directly linked to the ability to find small separators.

3 Approach

Add a short introduction to the chapter.

3.1 Empirical Analysis of Separator Scaling in Road Networks

To empirically investigate the relationship between graph size and separator size in road networks, we analyze data derived from a road network graph of Europe [PTV09]. A crucial first step in our research, addressed by this analysis, is to empirically validate whether road networks exhibit separators scaling near $\mathcal{O}(n^{1/3})$, as suggested by prior work [DSW16], as opposed to an alternative function, like $\mathcal{O}(n^{1/2})$, potentially appearing smaller due to a low constant factor. Figure Figure 3.1 plots the size of separators against the size of the corresponding subgraphs from which they were computed. Each data point (x, y) in this figure signifies that a subgraph containing x nodes possesses a separator of size y. The data encompasses separators computed recursively, starting from the separator of the original graph and proceeding within the resulting subgraphs.

Initial observations reveal outliers, particularly for very large subgraphs corresponding to continental or country scales. Specifically, analysis of the top-level separator structure for the Europe graph shows that the Scandinavian peninsula can be disconnected via separators significantly smaller than the general trend would suggest, due to specific geographic bottlenecks. This can be seen in Figure 3.2. Such outliers at the largest scales appear to be heavily influenced by macroscopic geographic features rather than intrinsic network structure representative of typical road networks. Consequently, these data points may not accurately reflect the general separator properties inherent in the finer structure of the road network graph. To mitigate the influence of these large-scale geographical artifacts and focus on more representative structural properties, our analysis primarily considers subgraphs with fewer than 10,000,000 nodes.

For enhanced visibility, particularly concerning the numerous data points corresponding to smaller subgraphs, and to avoid overrepresentation of larger subgraphs, we will also present data a log-log scale. This logarithmic scaling offers the additional advantage that a polynomial relationship between separator size y and subgraph size x, such as $y \propto x^c$, manifests as a linear trend in the log-log plot, facilitating the identification of potential power-law dependencies. Furthermore, to improve the interpretability of the visualization and emphasize the underlying trend over individual fluctuations or outliers present at various scales, the data points are aggregated into bins. Figure Figure 3.3 presents this binned data on a log-log scale.

Add statistical analysis

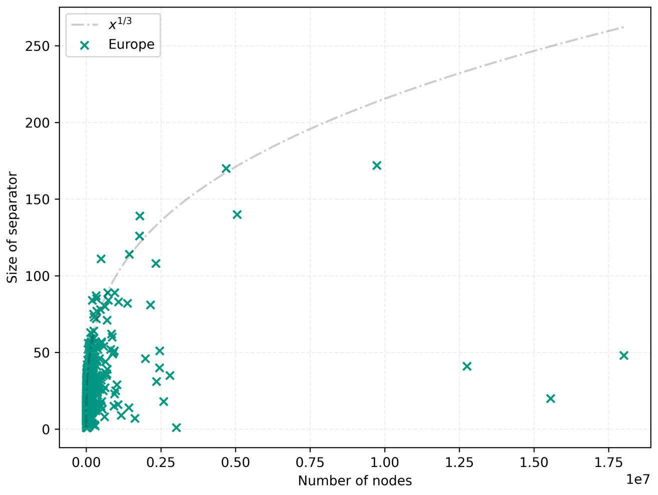


Figure 3.1: Empirical separator size versus subgraph size for the Europe road network. Each point represents a subgraph and its corresponding separator size.

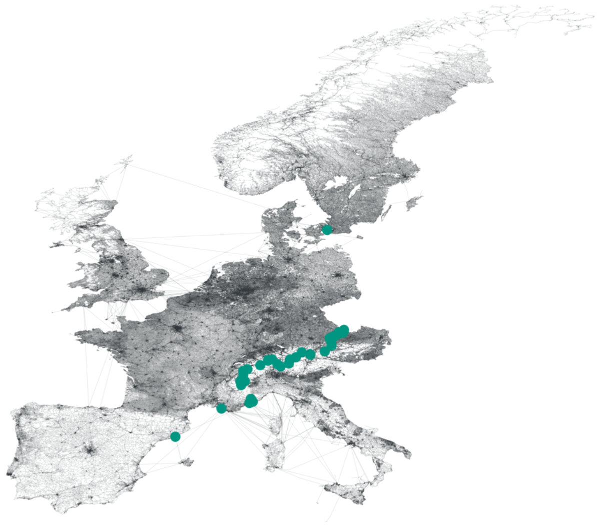


Figure 3.2: Illustration of a geographically influenced outlier at the continental scale: removal of a few nodes disconnects the whole Scandinavian peninsula.

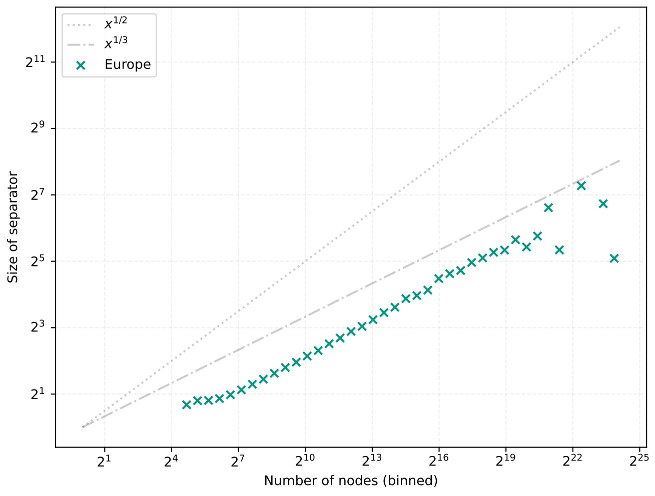


Figure 3.3: Log-log plot of binned separator size against subgraph size (limited to subgraphs < 10M nodes) for the Europe road network.

3.2 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,

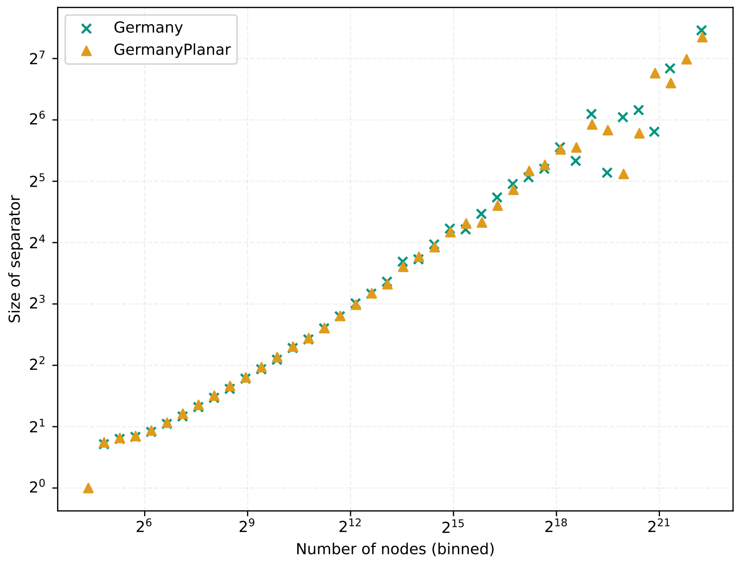
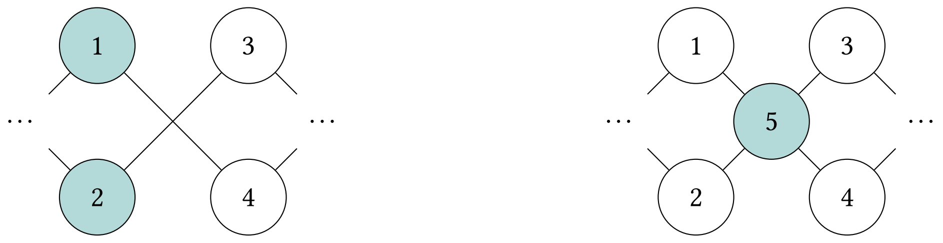


Figure 3.4: Comparison of separator sizes in the German road network: planar vs. non-planar.



(a) Separator in non-planar graph **(b)** Better separator in planarized graph

Figure 3.5: Example of a separator, where a better separator can be found in the planarized graph.

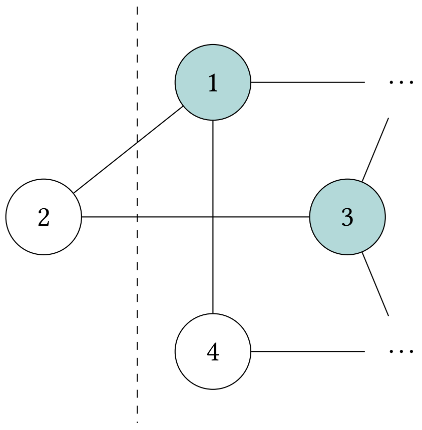
exhibited approximately 100,000 intersections. These figures slightly exceed the $\mathcal{O}(\sqrt{n})$ intersection counts reported in prior studies but remain within a similar magnitude [EGS10]. These differences could be explained by the unoptimal linear assumption of edges and might be mitigated 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 3.4.

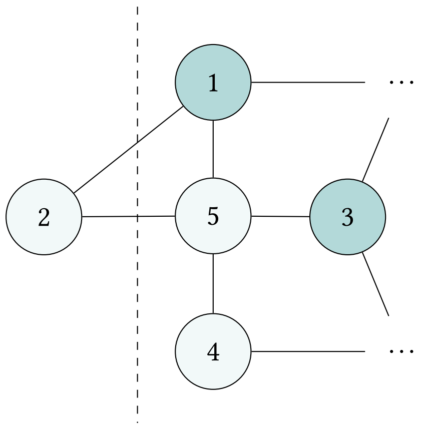
Our findings indicate that separators in non-planar road networks closely resemble those in their planarized versions. Frequently, non-planar separators are also separators in the planarized graph or can be adapted to planar ones with the addition of only a few nodes. This can be seen in Figure 3.7, 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



(a) Separator in non-planar graph



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

Figure 3.6: Example where the separator of the original graph is not a separator in the planarized graph.

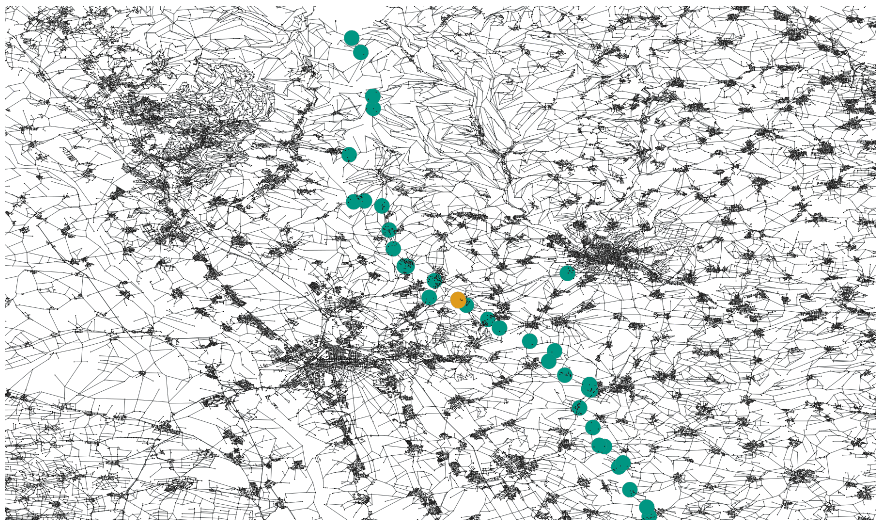


Figure 3.7: Example visualization of one possible top-level separator for Karlsruhe. Teal points represent the separator of the original graph, while orange points denote the additional nodes required to make it separator for the planarized version of Karlsruhe. Separators where computed with KaHIP.

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

[BCRW16] Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner. “Search-space size in contraction hierarchies”. In: *Theoretical Computer Science* Volume 645 (Sept. 2016), pp. 112–127. ISSN: 03043975. DOI: *10.1016/j.tcs.2016.07.003*.

[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*.

[DGPW11] Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck. “Customizable Route Planning”. In: *Experimental Algorithms*. Edited by Panos M. Pardalos and Steffen Rebennack. Berlin, Heidelberg: Springer, 2011, pp. 376–387. ISBN: 978-3-642-20662-7. DOI: *10.1007/978-3-642-20662-7_32*.

[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*.

[GHUW19] Lars Gottesbüren, Michael Hamann, Tim Niklas Uhl, and Dorothea Wagner. “Faster and Better Nested Dissection Orders for Customizable Contraction Hierarchies”. In: *Algorithms* Volume 12 (Sept. 16, 2019), p. 196. ISSN: 1999-4893. arXiv: *1906.11811[cs]*.

[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. Berlin, Heidelberg: Springer, 2008, pp. 319–333. ISBN: 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*.

[PTV09] PTV Group. *DIMACS-Europe Graph for the 9th DIMACS Implementation Challenge. Visit www.iti.kit.edu/resources/roadgraphs.php for details on how to acquire this graph.* 2009.

[SS13] Peter Sanders and Christian Schulz. “Think Locally, Act Globally: Highly Balanced Graph Partitioning”. In: *Experimental Algorithms*. Edited by Vincenzo Bonifaci, Camil Demetrescu, and Alberto Marchetti-Spaccamela. Berlin, Heidelberg: Springer, 2013, pp. 164–175. ISBN: 978-3-642-38527-8. DOI: *10.1007/978-3-642-38527-8_16*.