Efficient Route Planning

Paul Müller
Universität Konstanz
paul.mueller@uni-konstanz.de

***Abstract*—This project was developed as part of the course Efficient Route Planning Techniques at the University of Konstanz during the summer term, under Sabine Storandt. The system implements and visualizes different pathfinding algorithms.**

I. Introduction

This application provides an interactive visualization platform for comparing pathfinding algorithms on real road networks. The system implements Dijkstra's algorithm, Contraction Hierarchies (CH), and Customizable Contraction Hierarchies (CCH) with a web-based interface for performance analysis.

II. Implementation

A. Backend Architecture

The backend implements several algorithmic optimizations for efficient preprocessing and querying. The Contraction Hierarchies implementation employs parallel batch processing during vertex contraction, where independent vertices are identified and contracted simultaneously to reduce preprocessing time. The priority function extends beyond simple edge difference calculations:

$$p(v) = s(v)\left(1 + \frac{1}{d(v)+1}\right) - d(v) + 0.5c(v) \qquad (1)$$

Here,

- $s(v)$ denotes the number of shortcuts introduced by contracting vertex v,
- $d(v)$ is the degree of v, and
- $c(v)$ counts the already contracted neighbors of v.

This formulation effectively prioritizes low-degree vertices, particularly degree-1 nodes which contribute minimal shortcuts while reducing graph complexity.

Customizable Contraction Hierarchies utilize nested dissection ordering through recursive separator decomposition implemented via KaHIP integration. The preprocessing pipeline recursively partitions graphs into balanced components with minimal separators, generating vertex orderings optimized for contraction efficiency.

Witness search optimization employs bidirectional Dijkstra terminating early when potential witness paths exceed shortcut costs, significantly reducing preprocessing overhead compared to full shortest path computations.

B. Frontend Visualization

The frontend uses Vue.js together with deck.gl for interactive map visualizations powered by WebGL. It shows three-dimensional arcs to highlight shortcut routes, making it easy to distinguish them from the regular road segments and visualize the hierarchy of the road network. The map is built on MapLibre GL, which efficiently handles geographic data while allowing real-time layer control and displaying performance metrics.

When a user performs a search, a geocoding service converts the query into geographic coordinates. These coordinates are then mapped to the nearest point on the road network, ensuring the search location is correctly placed on the graph for pathfinding.

C. System Integration

A RESTful Go API provides communication between frontend and backend components, handling graph data serialization and query execution with measured performance metrics. The modular architecture maintains separation between algorithmic implementations and visualization layers.

Code quality is maintained through comprehensive unit testing and clean architectural patterns. Documentation is generated using pkgsite and accessible via pkgsite open . for complete API reference.

III. Experimental Results

Performance evaluation used OpenStreetMap datasets ranging from small urban networks to larger regional graphs. Contraction Hierarchies achieved substantial search space reduction, exploring 91-99% fewer nodes compared to Dijkstra's algorithm across all test cases.

Across all tested graph sizes, query times for Dijkstra's algorithm, Contraction Hierarchies (CH), and Customizable Contraction Hierarchies (CCH) were practically identical, with Dijkstra being only marginally slower. Interestingly, this is somewhat unexpected, as Dijkstra is usually slower in practice. Since all queries completed within just a few milliseconds, performance differences might be negligible at this scale, suggesting that substantially larger graph instances would be needed to fully observe the theoretical advantages of hierarchical methods.

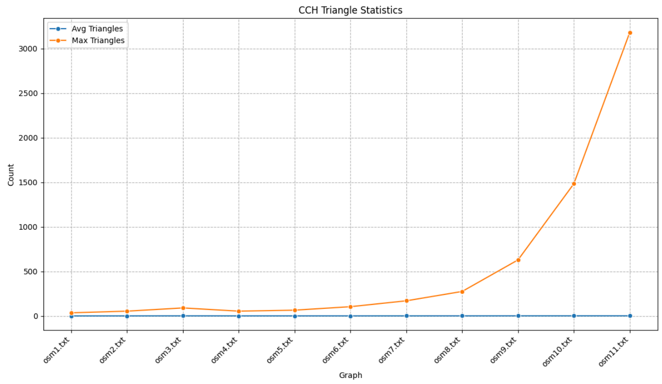


Fig. 1. Average and maximum number of triangles (witness paths) encountered during CCH preprocessing.

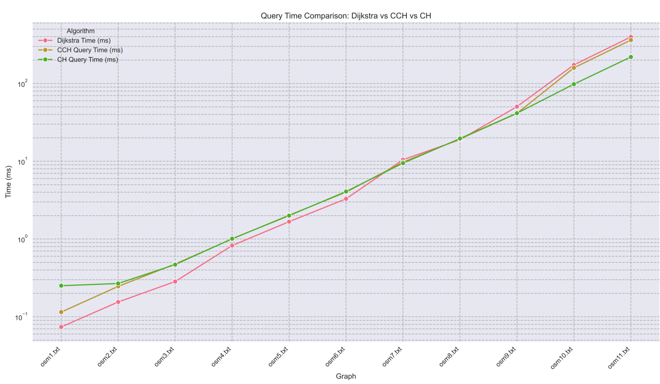


Fig. 2. Comparison of query performance (time) for Dijkstra's algorithm, CCH, and CH across various road network graphs. 100 random source-target node pairs were selected to query each.

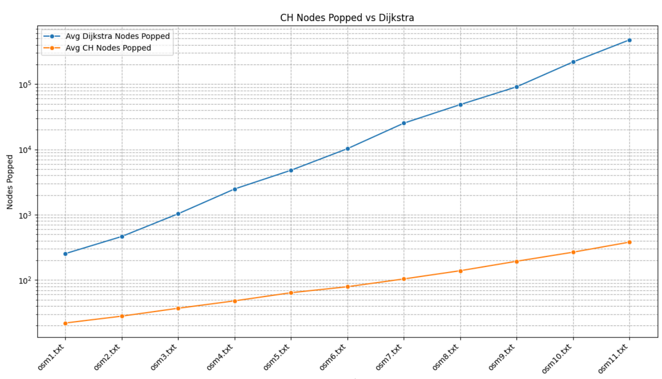


Fig. 3. Reduction in explored nodes by Contraction Hierarchies (CH) compared to Dijkstra's algorithm during queries.

IV. Figure Appendix

The appendix includes the measurements used to generate the plots Additionally there are 5 example queries using osm5.txt. Each picture shows the shortest path found by CH and Dijkstra, plotted together since the paths are the same CH just uses bidirectional search. In each pair, the second image shows CH's shortcuts as orange arcs.

TABLE I
CCH Preprocessing Experiment Results

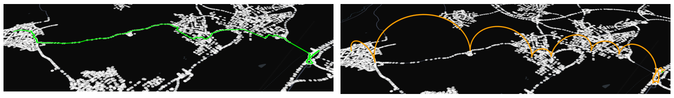
Graph	Preprocessing Time (ms)	Shortcuts Added	Avg Triangles	Max Triangles
osm1.txt	3001	446	1.66	36
osm2.txt	7003	1001	1.93	55
osm3.txt	15005	2253	2.38	91
osm4.txt	38011	5504	2.07	55
osm5.txt	75015	10982	2.05	66
osm6.txt	153029	22100	2.03	105
osm7.txt	379083	56761	2.14	171
osm8.txt	767190	114426	2.21	276
osm9.txt	1532432	235095	2.51	630
osm10.txt	4533281	589670	2.62	1485
osm11.txt	8266947	1376429	2.73	3182

TABLE II
Query Experiment Results

Graph	Avg Dijkstra Time (ms)	Avg CH Dijkstra Time (ms)	Avg Dijkstra Nodes Popped	Avg CH Nodes Popped	Mismatches
osm1.txt	0.172	0.251	254	22	0
osm2.txt	0.165	0.268	465	28	0
osm3.txt	0.333	0.469	1036	37	0
osm4.txt	0.784	1.009	2482	48	0
osm5.txt	1.601	2.000	4803	64	0
osm6.txt	3.631	4.077	10278	79	0
osm7.txt	10.030	9.465	25262	104	0
osm8.txt	21.247	19.590	48651	139	0
osm9.txt	45.201	41.511	91061	194	0
osm10.txt	107.234	98.176	218743	267	0
osm11.txt	241.892	219.334	476291	381	0

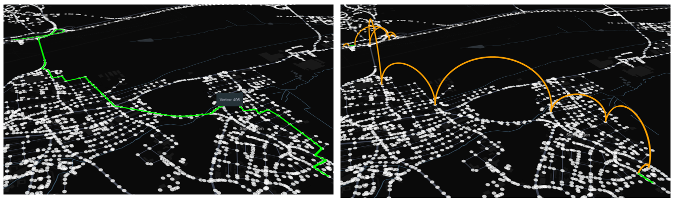
TABLE III
CCH Customization Experiment Results

Graph	Original Customization Time (ms)	Avg Random Customization Time (ms)
osm1.txt	0.486	0.461
osm2.txt	1.105	1.085
osm3.txt	2.508	2.601
osm4.txt	6.275	6.329
osm5.txt	12.638	13.105
osm6.txt	29.220	29.383
osm7.txt	92.977	94.169
osm8.txt	234.414	221.799
osm9.txt	526.632	529.690
osm10.txt	1489.694	1489.950
osm11.txt	3427.183	3441.274



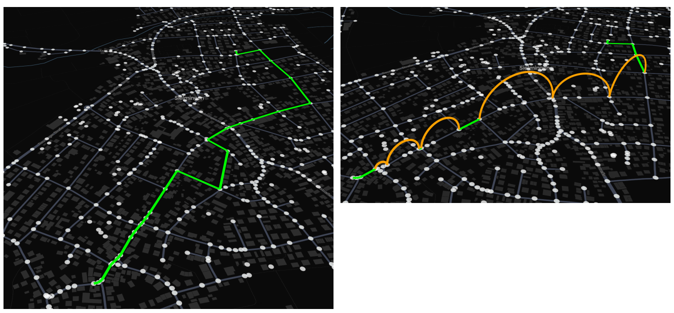
a) Unpacked 1 b) Not unpacked 1

Fig. 4. Comparison of unpacked vs not unpacked sample 1



a) Unpacked 2 b) Not unpacked 2

Fig. 5. Comparison of unpacked vs not unpacked sample 2



a) Unpacked 3 b) Not unpacked 3

Fig. 6. Comparison of unpacked vs not unpacked sample 3

a) Unpacked 4 b) Not unpacked 4

Fig. 7. Comparison of unpacked vs not unpacked sample 4