WIKIPEDIA

Contraction hierarchies

In applied mathematics, the method of **contraction hierarchies** is a technique to speed up shortest-path routing by first creating precomputed "contracted" versions of the connection graph.[1] It can be regarded as a special case of "highway-node routing".

Contraction hierarchies can be used to generate shortest-path routes much more efficiently than Dijkstra's algorithm or previous highway-node routing approaches,[1] and is used in many advanced routing techniques.[2] It is publicly available in open source software to calculate routes from one place to another.[3][4][5][6][7]

Contents

The algorithm

In general, scalable map routing algorithms have two distinct phases: preprocessing of the original graph (which may take more than an hour to finish) and queries (less than a second). Contraction hierarchy (CH) is an extreme case of "hierarchy" approach, which generates a multi-layered node hierarchy in the preprocessing stage. In a CH, every node in the graph is represented as its own level of hierarchy. This can be achieved in many ways; one simple way is simply to label each node in order of some enumeration from 1 to N, where N is the number of nodes in the graph. More sophisticated approaches might consider the type of road (highway vs minor road, etc.).[8][9]

CHs have been extended to a three phase setup called CCH[10] that supports flexible edge weights. In this setup there is a slow (hours) preprocessing phase, a reasonably fast (about a second) customization phase and a very fast (milliseconds) query phase. In the preprocessing phase only the graph but not the edge weights are revealed to the algorithm. In the customization phase the weights are revealed and in the query phase shortest paths are computed. Changing the weights of the graph requires only rerunning the customization phase but not the preprocessing phase. It is therefore possible to exchange the edge weights within seconds. This setup enables live-traffic updates and routes can easily be adjusted to user specific preferences. An open-source CCH implementation is available.[7][11]

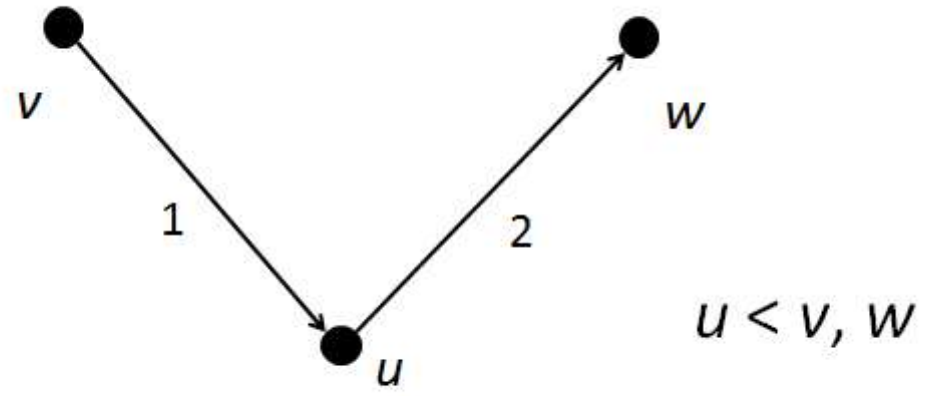
Preprocessing

Levels/order of nodes in CH can be arbitrary. The main point is that **shortcuts** are introduced when necessary. To understand when a shortcut is necessary, one has to understand the searching algorithm. The searching algorithm (bidirectional Dijkstra's algorithm) in this case is constrained so that it only relaxes edges that are connected to the nodes that are higher in CH than current node in one direction, and vice versa. With this constraint, the algorithm would not find certain shortest paths in the unprocessed network, and because of that we need to introduce new edges to the graph that

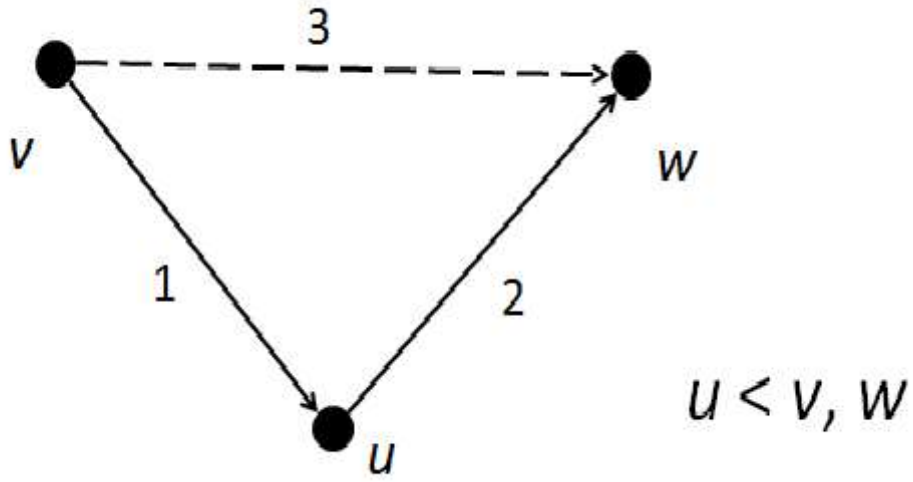
represent existing shortest paths that the algorithm doesn't take into consideration. Not all shortest paths need to be restored as new shortcut edges: it's enough to take in consideration adjacent nodes of some node that are higher in CH (since the sub-path of some shortest path is itself a shortest path). Algorithmically:

- Add order/levels to the graph nodes
- For each node, respecting order, get its adjacent nodes of higher order and find if shortest path between each pair of them goes through current node and if it is, add shortcut edge

Let's say that we take in consideration just 2 adjacent nodes (from, in general, n of them):



From this picture, if shortest path from v to w goes through node u that is lower in CH, a new edge has to be added the CH graph so that the shortest paths that the searching algorithm takes into consideration are preserved.



The weight of the new edge is equal to the path distance from v to w.

When preprocessing of the original graph is done, we have a CH graph which consists of the original graph with node ordering added and with shortcut edges introduced.

Querying

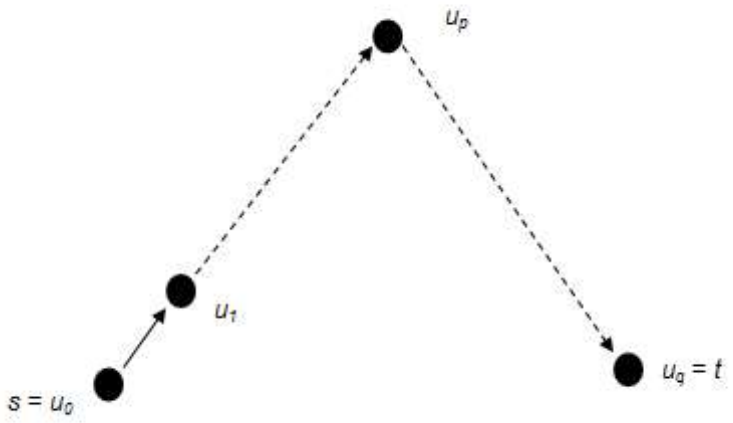
For the searching algorithm, bidirectional Dijkstra's algorithm is used. It is classical Dijkstra's algorithm with some modifications. The algorithm searches from the starting node in one direction and from the ending node in other direction (this is classical bidirectional Dijkstra's algorithm), but it relaxes edges that are directed toward higher hierarchy nodes in one direction (essentially it expands towards higher hierarchy nodes) and edges that are directed toward lower hierarchy nodes in the other direction. If the shortest path exists, those two searches will meet at some node v. A shortest path from s to t consists of paths from s to v and from v to t.

The shortest paths found by this algorithm have the particular form:

$$\langle s = u_0, u_1, \ldots, u_p = v, \ldots, v_q = t \rangle, \quad p, q \in \mathbb{N}$$

$$u_i < u_{i+1}, \quad i \in \mathbb{N}, i < p$$

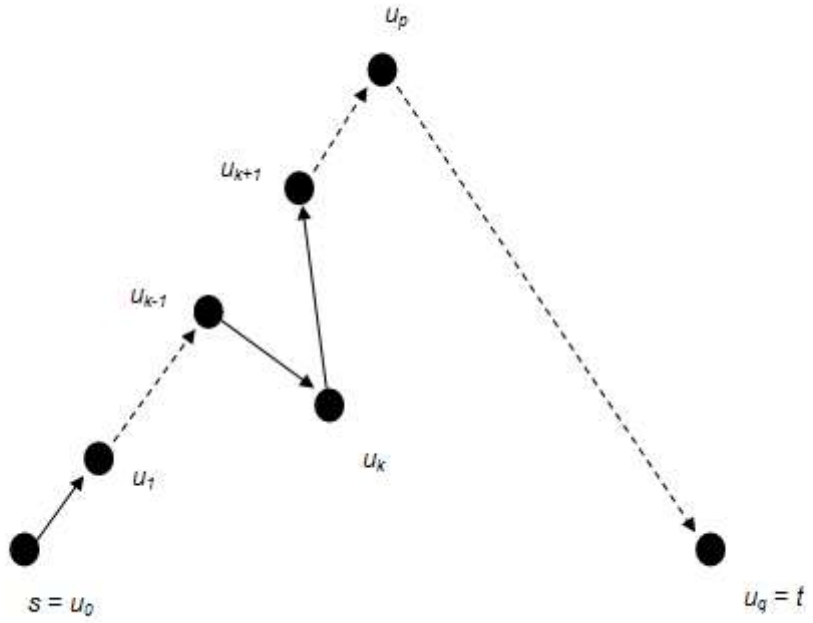
$$u_j > u_{j+1}, \quad j \in \mathbb{N}, p \leq j < q$$



A path found by a query is the shortest path because of the preprocessing stage. In the preprocessing stage we transformed graph introducing shortcut edges, which represents the shortest paths that algorithm does not take into consideration.

In order to return the final result, the shortcut edges need to be postprocessed to give the actual paths they represent in the original graph.

In order to demonstrate that this algorithm retrieves shortest paths, consider it by contradiction: let's assume (for forward search, identical thing stands for backward search) that there exists a path that is shorter than the one we found with this algorithm:



Let's say that at some point there exists a subpath that is shorter than path. Because algorithm expands towards nodes that have higher order, order of u_k node must be lower than order of u_{k-1} and u_{k+1} nodes. Because of that fact, in preprocessing stage that path would be represented as shortcut edge with equal length, and therefore no such path that is shorter than the one algorithm found can exists.

References

1. Robert Geisberger (1 July 2008). *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks* (http://algo2.iti.kit.edu/documents/routeplanning/geisberger_dipl.pdf) (PDF) (Thesis). Institut für Theoretische Informatik Universität Karlsruhe. Retrieved 2010-12-27.
2. Delling, D.; Sanders, P.; Schultes, D.; Wagner, D. (2009). "Engineering route planning algorithms". *Algorithmics of Large and Complex Networks: Design, Analysis, and Simulation*. Springer. pp. 117–139. doi:10.1007/978-3-642-02094-0_7 (https://doi.org/10.1007%2F978-3-642-02094-0_7)..
3. "OSRM – Open Source Routing Machine" (http://project-osrm.org/).
4. "Wiki – OpenTripPlanner" (http://www.opentripplanner.org).
5. "Web – GraphHopper" (http://graphhopper.com).
6. "GitHub – Tempus" (https://github.com/ifsttar/Tempus).
7. "GitHub – RoutingKit" (https://github.com/RoutingKit/RoutingKit).
8. Hannah Bast (2012). "Efficient Route Planning, Lectures" (http://ad-wiki.informatik.uni-freiburg.de/teaching/EfficientRoutePlanningSS2012).
9. Ivan Škugor (2012). "Contraction hierarchies" (http://into-the-zen.blogspot.co.uk/2011/01/contraction-hierarchies-part-1.html). Retrieved 10 February 2013. (CC-BY)
10. Dibbelt, J.; Strasser, B.; Wagner, D. (2015). "Customizable Contraction Hierarchies" (https://dl.acm.org/citation.cfm?doid=2886843). *Journal of Experimental Algorithmics*. ACM. **21** (1): 1.5:1–1.5:49. doi:10.1145/2886843 (https://doi.org/10.1145%2F2886843). Retrieved September 9, 2016..
11. "GitHub – GraphHopper Routing Engine" (https://github.com/graphhopper/graphhopper).