Distance Problems in Networks - Theory and Praxis

Contraction Hierarchies

Mykola Protsenko

October 12, 2010

Abstract

Contraction Hierarchies (CH) is another hierarchical approach to a shortest path/shortest distance problem. The main idea is to arrange all nodes according to their importance and then iteratively contract them in this order, adding shortcuts to preserve shortest distances. The queries are processed using modified bidirectional Dijkstra algorithm: the forward search uses only edges going to more important nodes and the backward search the ones going to less important nodes.

Contents

1 Introduction

The preprocessing step of Contraction Hierarchies routing algorithm consists of two parts:

- node ordering
- node contraction

After the first one we have all nodes sorted according to their 'importance'. Then the hierarchy is constructed by contracting the nodes in this order. The contraction is performed by deleting the node from the graph and adding some shortcuts to preserve shortest distances in the remaining overlay graph.

The queries are processed with the modified version of bidirectional Dijkstra which attempts to avoid less significant nodes by using the shortcuts added in the preprocessing step. To achieve this the forward search uses only the edges going to more important nodes and the backward search only those going to less important nodes so that the both search scopes eventually meet in the node with the highest priority on a shortest path between source node and target node.

In next sections we describe 3 important issues of Contraction Hierarchies: the node ordering, the contraction of nodes and the query processing. In the last section some experimental results are shown.

2 Node Ordering

To arrange nodes in order of their importance we maintain a priority queue which minimum element should be contracted next. The priority, or in other words, the importance of a node is computed as a linear combination of several terms describing how contracting one specific node will affect such important parameters as size of a graph, query search space, query time etc. Although an 'optimal' node ordering seems to be a hard problem, simple and easy-to-compute local heuristics appear to work quite well on real road networks.

Next we present different terms, that can be used to compute priority of the node. Note that the linear coefficients are important tuning parameter that could be used to tweak the algorithm for specific problems.

- **Edge difference**. The Edge difference of a node `v` is defined as the number of shortcuts needed when `v` is contracted minus the number of edges incident to `v`. Contracting the nodes with least edge difference will make the resulting graph as small as possible.

However, the edge difference only is not sufficient to obtain good Contraction Hierarchies. Experiments show that contracting nodes everywhere in a graph in an uniform way might be not a bad idea. The next two terms represent the possible measurements of 'uniformity' of nodes contraction.

- **Deleted Neighbors**. This is simply the number of already contracted neighbors including those reached via shortcuts.
- **Voronoi Regions**. The square root of the Voronoi Region size is used as a term for our priority function.

  The Voronoi Region of a node `v` is defined as a set of nodes that are closer to `v` then to any other node of the graph:

  $$R(v) := \{u | d(v,u) < d(w,u)\ \forall w \in E\}$$

  Since the Voronoi Region of a contracted node will be 'eaten up' by the Voronoi Regions of its neighbors, the nodes with many nodes being contracted in their neighborhood will have big Voronoi Regions. Thus contracting nodes with small value of this term will provide uniformity of contraction.

To improve preprocessing time, the next term can be used in the priority function:

- **Cost of contraction**: the cost of making a decision, if and how many shortcuts are needed when contracting specific node `v`.

We also can try to estimate, how contracting a node will affect the query times:

- **Cost of query**. The estimate `Q(v)` is to be the upper bound of the number of hops on a path $\langle s, .., v \rangle$. Initially Q(v)=0 for all nodes. When node `v` is contracted, for each neighbor `u` we set:

$$\text{Q(u)}:=\max\{\text{Q(u)},\text{Q(v)}+1\ \}$$

It is clear, that contraction of a node may affect the priorities of other nodes. So additional techniques are needed to deal with this problem.

- **Lazy update:** before contracting node `v` its priority function is evaluated again. If new priority of `v` is greater then priority of the second smallest element of the queue, we reinsert `v`. This actions are repeated until consistent minimum is found.
- We recompute the priorities of the neighbors of contracted node.
- It might be a good idea to recompute all priorities from time to time.

3 Contraction

In this section we take the process of node contraction under the scope.

Suppose we have an overlay graph G=(V',E') given and the next node to contract is `v`. While contracting this node we have to add some shortcuts to replace unique shortest paths going through `v`. It means, that for each `u` with $(u, v) \in E'$ and `w` with $(v, w) \in E'$ we have to look for shortest distances in our overlay graph ignoring `v`. If the distance we found d(u,v) > c(u,v)+c(v,w) then shortcut is needed and we add a new edge between `u` and `w` with weight c(u,v)+c(v,w).

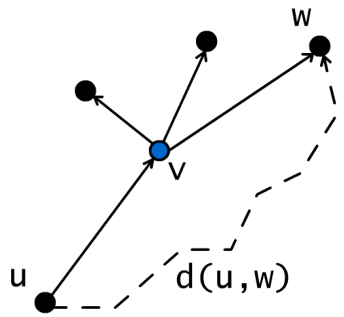

Figure 1: Contraction of a node.

Since the computation of exact value of the distance d(u,w) can be quite expensive, we perform the shortest distance search (Dijkstra) with limited number of hops. In this case the maximum number of hops is another tuning parameter: if we choose small hop limit, we will have fast preprocessing but possibly slow query processing. On the other hand, the large hop limit gives us smaller query time at a price of slower node contraction.

4 Queries

For query processing we split the contraction hierarchy CH=(V,E) (original nodes, original edges + shortcuts) into:

- **upward graph** $G_\uparrow := (V, E_\uparrow)$ with edges going to nodes with higher priority:

$$E_\uparrow := \{(u, v) \in E : u < v\}$$

- **downward graph** $G_\downarrow := (V, E_\downarrow)$ with edges going from nodes with higher priority:

$$E_\downarrow := \{(u, v) \in E : u > v\}$$

Then the modified bidirectional version of Dijkstra is executed to compute a shortest distance between start node `s` and target node `t`.

The forward search is performed in the upward graph and the backward search in the downward graph. Note that unlike the "classical" bidirectional Dijkstra, this search can not be stopped when there is one node settled in both search scopes. So shortest distance is computed as:

$$\min\{\text{d(s,v)+d(v,t):v is settled in both searches}\}$$

The next Lemma proves correctness of this approach.

Lemma 4.1. *$d(s,t) = min\{d(s,v) + d(v,t) : v$ is settled in both searches$\}$*

Proof. In other words, we have to show the existence of a shortest path $P = \langle s, ..v, ..t\rangle$ with following properties:

- There is some node `v` - the node with highest priority in `P`.
- The first part of `P`, $\langle s, ..v\rangle$ goes in ascending priority.
- The second part of `P`, $\langle v, ..t\rangle$ - in descending priority.

Obviously, if a shortest path with this properties exists, it will be found by our modified bidirectional Dijkstra query algorithm.

To prove the existence of such a path, we first assume the contrary: all existing shortest paths between `s` and `t` have some node `v` in it, with both predecessor `w` and successor `u` in `P` having priorities higher, then `v`, which violates the last two properties.

In this case, at a moment of contracting node `v`, both `u` and `w` would still be present in current overlay graph, so we would either add a shortcut edge between those or there were a path shorter or of equal length using only nodes with priorities higher then `v`.

This is a contradiction to our previous assumption about shortest paths. □

Since a shortest path found by our algorithm may contain also shortcut edges, we need some special technique to "extract" a shortest path that uses only edges from input graph. The main observation is that each shortcut edge bypasses exactly one node, so if we store this node together with the shortcut we will be able to unpack paths recursively.

Note that if the extraction of shortest paths for some reason is not required, we can reduce the space consumption of our algorithm by storing each edge (u,v) only in the node with minimum priority min{u,v}.

5 Experiments

The Contraction Hierarchies algorithm was tested on a road network of Western Europe with over 18 M nodes and over 42 M directed edges. The table below shows the performance of various variants of Contraction Hierarchies (especially node ordering) executed on the test network. The last line gives corresponding data for Highway Node Routing.

Node ordering heuristics:

- E = edge difference
- D = deleted neighbors
- S = search space size
- V = $\sqrt{\text{Voronoi region size}}$
- Q = upper bound on edges in search paths
- L = limit search space on weight calculation
- W = relative betweenness
- digits: hop limit

method	node ordering [s]	hierarchy construction [s]	query [μs]
E	13010	1739	670
ED	7746	1062	183
ES	5355	123	245
ED5	634	98	224
EDS5	652	99	213
EDS1235	**545**	**57**	223
EDSQ1235	591	64	211
EDSQL	1648	199	173
EVSQL	1627	170	159
EDSQWL	1629	199	163
EVSQWL	1734	180	**154**
HNR	594	203	802

As we can see, even using only simple heuristics (for instance, edge difference and deleted neighbors) we get much better query times then Highway Node Routing.

6 Conclusion

Contraction Hierarchies are simple and efficient approach to a shortest path/shortest distance problem.

It can be very useful, if some/all edge weights may change from time to time in our network. The reason for this is, that the preprocessing consists of two main steps: node ordering and node contraction. The algorithm is correct for all node orderings. So if the weights have being changed, only contraction step needs to be performed again. In many application scenarios the important nodes remain important even after the change of weights (Example: switching from driving time to driving distance), so even without reordering the nodes we will get fast query processing.

Contraction Hierarchies can also be used as a part of some more sophisticated routing algorithm, for instance as preprocessing routine in Transit-Node Routing.

References

[Del] Daniel Delling. Algorithmen fuer Routenplanung - Vorlesung 5.

[RGD08] Dominik Schultes Robert Geisberger, Peter Sanders and Daniel Delling. Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks. 2008.