CORPORATE TECHNOLOGY

An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background

Ulrich Lauther

Siemens AG, München
ulrich.lauther@siemens.com

Overview

- Application Scenarios and Motivation
- Demonstration
- Basic Idea - Definition of Edge Flags
- How to Use Edge Flags
- How to Calculate Edge Flags
- How to Define Regions
- Results
- Conclusions

CT SE 6

Software & Engineering
Discrete Optimization

Motivation

Application Scenarios:

- Autonomous Car Navigation Systems:
  need cheapest hardware
  need short response time for one path calculation
- centralized traffic guidance system in client/server architecture:
  powerful hardware
  need high throughput of the server responding to many requests

Resources:

- virtual memory / RAM
- online CPU-time
- often ignored: preprocessing time

Complexity of Shortest Path Calculations:

For a graph $G = (V, E)$ with n nodes and m edges, we have

- **Worst case complexity** of Dijkstra's algorithm: between $O(n^2)$ and $O(m)$, depending on assumptions on density and edge length distibution.
- **Expected runtime** for shortest path calculations on a roadmap: $O(d^2 \log d)$ where d is the number of edges in the shortest path.

Demonstration

Common Remedy: Heuristics

- Layer concept
- Modified A^* algorithm

Drawback:

- no guarantee for good solution

Much less used and investigated: preprocessing

- trading preprocessing time for runtime in application

Preprocessing - Basic Idea

Road map is partitioned into r regions:

- regions are connected and constitute a parcelization, e.g., rectangular parcelization
- each node belongs to exactly one region

Two edge flags are calculated per edge and region:

for each edge $e = (v, w)$ and each region i we calculate $v\_flag_{e,i}$ and $w\_flag_{e,i}$, s.t.

$v\_flag_{e,i} = 1$ iff $\exists$ shortest path into region i from node v over edge e

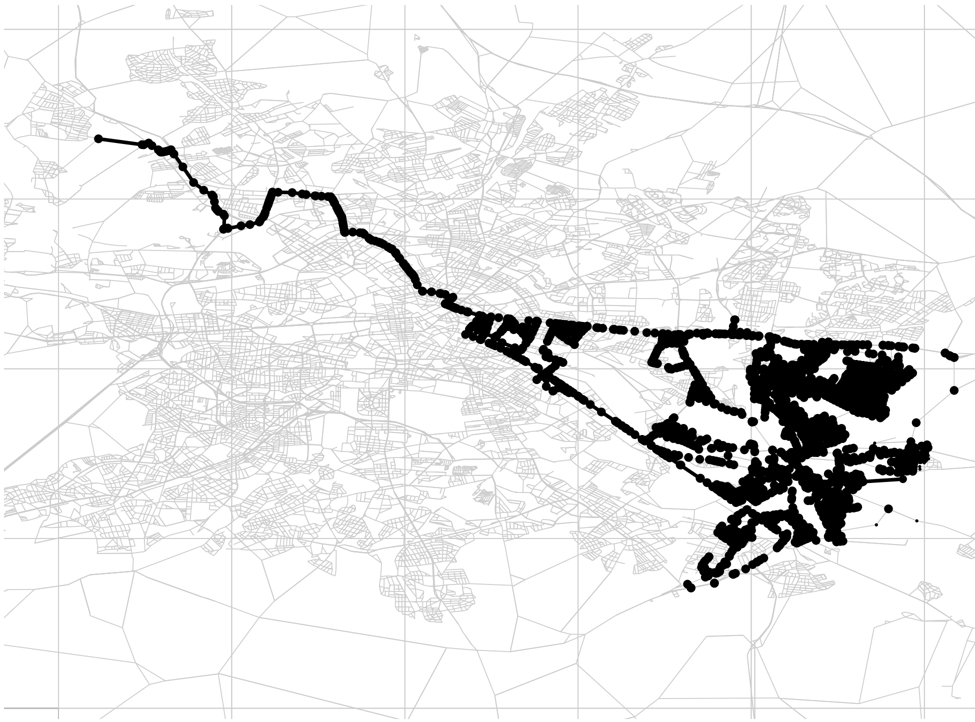
$w\_flag_{e,i} = 1$ iff $\exists$ shortest path into region i from node w over edge e

Memory: 2*number_of_edges*number_of_regions bits

How to Use Edge Flags:

```
int dijkstra(Graph& g, Node* source, Node* target) {

  int target_region = target->region();

  Node* v;
  forall_nodes(v,g) v->dist = ∞; // initialize all nodes

  priority_queue q; // initialize priority queue
  source->dist = 0;
  q.insert(source);

  while ((v = q.del_min())) { // while q not empty
    if (v == target) return v->dist;
    Node* w;
    Edge* e;
    forall_adj_edges(w,v,e,g) { // scan node v
      if (! flagged(v,e,target_region)) continue;
      if (w->dist > v->dist + e->length) {
        w->dist = v->dist + e->length;
        if (q.member(w)) q.update(w);
        else             q.insert(w);
      }
    }
  }
  return ∞;

} // dijkstra
```

The Problem of Cones



- Problem: near the target region many edges will be flagged
- Solution: bidirectional path calculation
- edge flags needed for "wrong" direction!
- Memory: 4*number_of_edges*number_of_regions bits (for bidirectional edges)

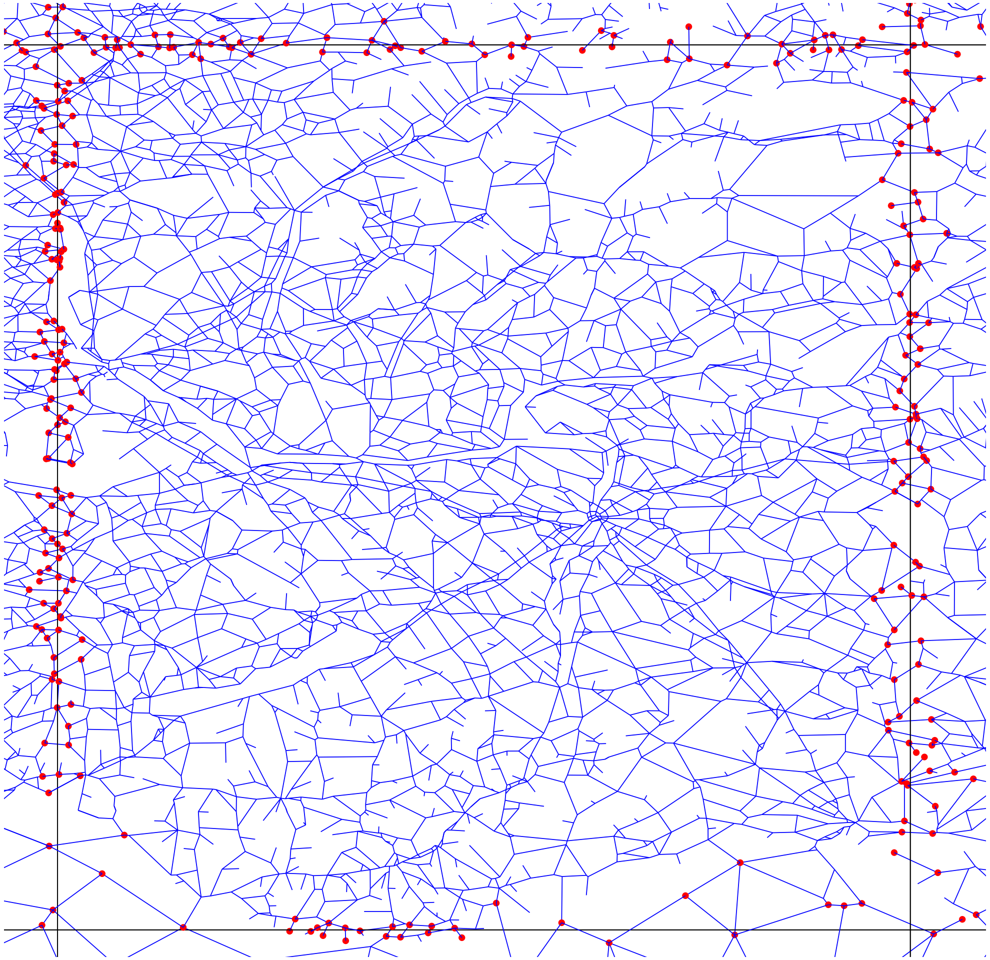
How to Calculate Edge Flags

Simple But Dead Slow Method:

```
for all regions i {

  for all nodes u of region i {

    calculate shortest path tree rooted at u;

    for all edges e = (v,w) {
      if (w->dist == v->dist + e->length) e->set_flag(w,i);
      if (v->dist == w->dist + e->length) e->set_flag(v,i);
    }
  }

}
```

Above code needs to be run twice: for forward and backward traversal of edges.

A Faster Method: expand from exported nodes only



If an edge crosses a region boundary, the two incident nodes are *exported* nodes.

A shortest path from outside the region into the region must cross an exported node.

A Fast but Wrong Method: Expand from all exported nodes of a region simultaneously

- Implementation: put all exported nodes with distance 0 into priority queue, then run standard Dijkstra
- Resulting edge flags describe shortest path to nearest exported node.
- This is not necessarily a shortest path to a target node inside the region.

A Fast Method: use similarity of shortest path trees rooted at neighboring nodes

Runtimes for Various Methods:

Mode	Runtime	Comment
Expand from all nodes	9 weeks	extrapolated
Expand from exported nodes	4.2 hours	extrapolated
The fast method	12 minutes	measured

Memory Needs:

- In principle: $4 * r * m$ bits
- Using compaction: 6.9 bytes/edge with 139 regions

How to Define Regions

Possible Methods:

- Use parcelization of the application
- Grid based
- Square Cover Algorithm
- Other cluster methods

Results

Speed-up in Target Application measured using 300 random source/target pairs

Nodes	326159
Edges	513567
Regions	139
Runtime without preprocessing [sec/path]	0.302
Runtime with preprocessing [sec/path]	0.0047
Speed-up	64.3

Average speed-up: 64

Long paths: Runtime drops from $O(d^2 \log d)$ to $O(d)$

Availability

- The concept of edge flags and associated algorithms are protected by a patent.
- Source code licenses are available.

Conclusions

- A new preprocessing technique for static graphs with coordinates in nodes
- Resulting shortest path calculations are exact and extremely fast

Thanks for listening!

Still questions?