Arc Flags

Ferienakademie im Sarntal — Course 2
Distance Problems: Theory and Practice

Tobias Walter

October 27, 2010

Friedrich-Alexander-Universität Erlangen-Nürnberg
Technische Universität München
Universität Stuttgart

Outline

Outline

Motivation

- Often the shortest path problem has to be solved repeatedly for the same graph
- Decrease size of the search space by using additional information
- Many arcs aren't used for shortest paths of a certain length
- Prune arcs which aren't neccesary

Notation

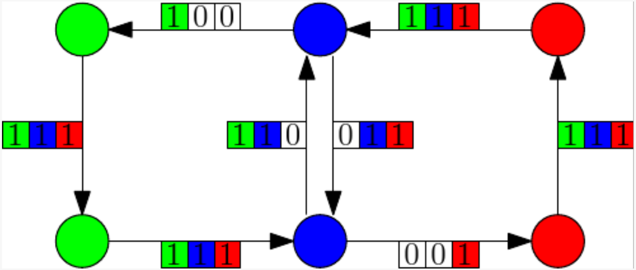
- Directed simple graph $G = (V, A, l)$, V finite set of nodes, $A \subseteq V \times V$ arcs, $l : A \to \mathbb{R}$ the arc lengths
- $n = |V|, m = |A|$
- Reversed graph $G_{rev} = (V, A_{rev}, l_{rev})$ with $A_{rev} = \{(v, u) \mid (u, v) \in A\}$ and $l_{rev}(v, u) = l(u, v)$
- G is called sparse, if $m \in \mathcal{O}(n)$

Basic Idea

- Divide the graph (V, A) into p regions $r : V \to \{1, \dots, p\}$
- For every arc a assign a flag vector $f_a : \{1, \dots, p\} \to \{\mathrm{false}, \mathrm{true}\}$
- $f_a(r_v)$ is true iff a is used on a shortest path to the region of $v \in V$ (This implies that arcs inside a region mark this region)
- Only consider arcs with $f_a(r_v) = \mathrm{true}$ in Dijkstra's algorithm for finding a shortest path to v

Basic Idea

- Divide the graph (V, A) into p regions $r : V \to \{1, \ldots, p\}$
- For every arc a assign a flag vector $f_a : \{1, \ldots, p\} \to \{\mathrm{false}, \mathrm{true}\}$
- $f_a(r_v)$ is true iff a is used on a shortest path to the region of $v \in V$ (This implies that arcs inside a region mark this region)
- Only consider arcs with $f_a(r_v) = \mathrm{true}$ in Dijkstra's algorithm for finding a shortest path to v



Modified Dijkstra still correct

Lemma

Dijkstra with arc flags finds a shortest path from s to t, $s, t \in V$ if one exists

Modified Dijkstra still correct

Lemma

Dijkstra with arc flags finds a shortest path from s to t, $s, t \in V$ if one exists

Proof.

- Consider a shortest path $s = v_0, \ldots, v_n = t$.
  - Case 1: (v_i, v_{i+1}) is inside the same region
  - Case 2: (v_i, v_{i+1}) is crossing between two regions

  In both cases $f_{(v_i, v_{i+1})}(r_{v_{i+1}})$ is true by definition.
- If a path is found, it is still a shortest path since the order of the processed arcs remains unchanged

□

Outline

All-Pairs Shortest Path

- For every $a = (h, t) \in A$ calculate the shortest path trees
- For every $v \in V$ check if $|d_h(v) - d_t(v)|$ equals the length of a
- If so set $f_a(r_v)$ to true
- Complexity of $\mathcal{O}(2m(m + n\log(n)))$
  Takes weeks for 1M nodes and 2.5M arcs

Boundary Arcs

Definition

$(u, v) \in A$ is a boundary arc if $r_u \neq r_v$. v is then called a boundary node.

Lemma

If the flag vectors f_a are computed with the set of shortest paths to boundary nodes only, then Dijkstra's algortihm with arc flags is still correct

Proof.

Every shortest path from s to t, where $r_s \neq r_t$ has to enter the region r_t with some boundary arc (u, v). The subpath from s to v is also a shortest path. Hence the flag vectors are still the same. □

Boundary Arcs (cont'd)

- For a region $r \in R$ and a boundary node b of r we calculate $T_b = \{a \in A \mid f_a(r) = \mathrm{true}, a$ is on a shortest path via b to any node in $r\}$
- The corresponding arcs in G_{rev} to T_b form a shortest path tree
- Calculate for every bounding node b a shortest path tree in G_{rev}
- $\mathcal{O}(k(m + n \log n))$ if there are k different boundary nodes
- Running time corresponds with the choice of the partition

Centralized Shortest Path Search

- Instead of starting from just one bounding node, start from all bounding nodes $B = \{b_1, \ldots, b_j\}$ of a region
- Assign to each vertex v a label $L_v : B \to \mathbb{R}^+$
- $L_v(b_i)$ is the length of the currently shortest path to v from b_i in G_{rev}
- Use a heap to store those nodes that wait to propagate labels
- Key $k(v)$ used for sorting the heap
- How to choose an initialization of labels and keys?

Centralized Shortest Path Search

How to initialize labels?

- Set unknown values to infinity

Centralized Shortest Path Search

How to initialize labels?

- Set unknown values to infinity
- Limited initialization
  - Limit Dijkstra search to the current region
  - Gives upper bound of all boundary nodes if the region is connected

Centralized Shortest Path Search

How to initialize labels?

- Set unknown values to infinity
- Limited initialization
  - Limit Dijkstra search to the current region
  - Gives upper bound of all boundary nodes if the region is connected
- Aborted initialization
  - Dijkstra search aborted when all boundary nodes have been scanned
  - Correct labels between the boundary nodes

Centralized Shortest Path Search

How to choose a good key?

- Minimum tentative key
  - Let K be the set of change values
  - Set $k(v) := \min(K \cup k(v))$ if v is already in the heap, else set $k(v) := \min(K)$
  - Good theoretical upper bound: Each node at most $|B|$ times in heap

Centralized Shortest Path Search

How to choose a good key?

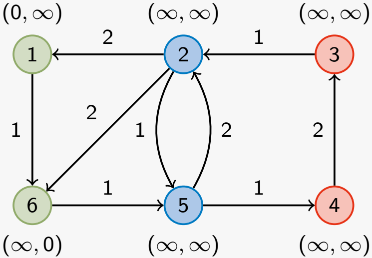
- Minimum tentative key
  - Let K be the set of change values
  - Set $k(v) := \min(K \cup k(v))$ if v is already in the heap, else set $k(v) := \min(K)$
  - Good theoretical upper bound: Each node at most $|B|$ times in heap
- Minimum total key
  - Set $k(v)$ as the minimum of all label values
  - Behaves like $|B|$ parallel Dijkstra calls
  - Worse theoretical upper bound, but good at experiments

Centralized Shortest Path Search

How to choose a good key?

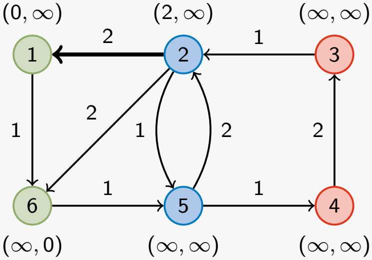
- Minimum tentative key
  - Let K be the set of change values
  - Set $k(v) := \min(K \cup k(v))$ if v is already in the heap, else set $k(v) := \min(K)$
  - Good theoretical upper bound: Each node at most $|B|$ times in heap
- Minimum total key
  - Set $k(v)$ as the minimum of all label values
  - Behaves like $|B|$ parallel Dijkstra calls
  - Worse theoretical upper bound, but good at experiments
- Domination value
  - Store domination value: number of values which have been improved
  - Order first by domination value, then by minimum total key

Example: Centralized Shortest Path with Minimum Total Key



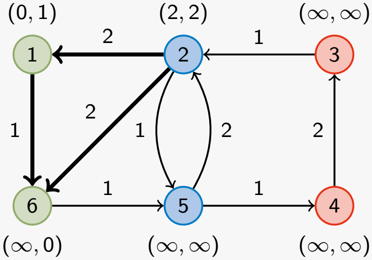
Nodes in the heap: $\{1, 6\}$

Example: Centralized Shortest Path with Minimum Total Key



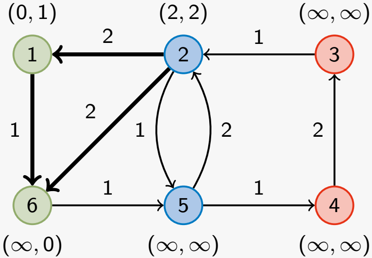
Nodes in the heap: $\{6, 2\}$

Example: Centralized Shortest Path with Minimum Total Key



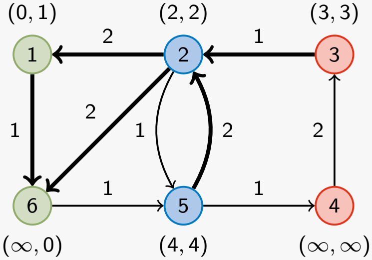
Nodes in the heap: $\{1, 2\}$

Example: Centralized Shortest Path with Minimum Total Key



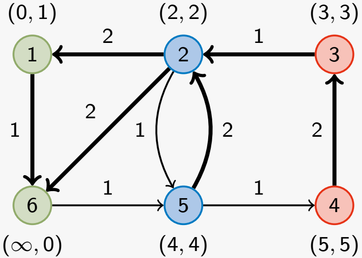
Nodes in the heap: $\{2\}$

Example: Centralized Shortest Path with Minimum Total Key



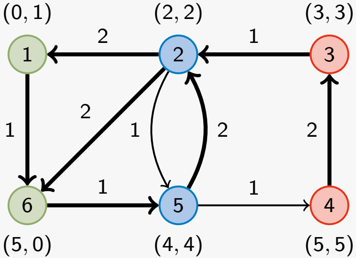
Nodes in the heap: $\{3, 5\}$

Example: Centralized Shortest Path with Minimum Total Key



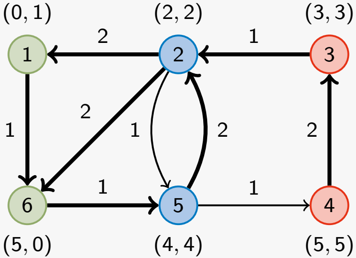
Nodes in the heap: $\{5, 4\}$

Example: Centralized Shortest Path with Minimum Total Key



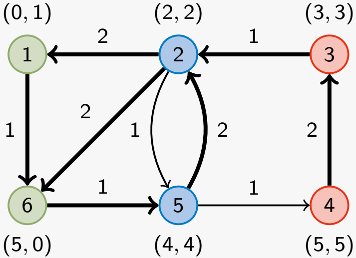
Nodes in the heap: $\{6, 4\}$

Example: Centralized Shortest Path with Minimum Total Key



Nodes in the heap: {4}

Example: Centralized Shortest Path with Minimum Total Key



Nodes in the heap: {}

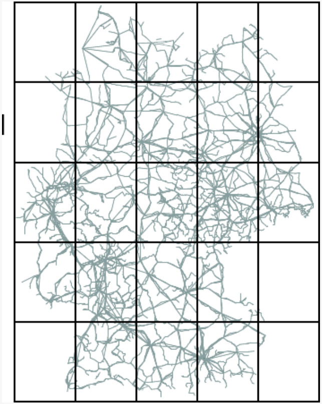
Outline

Criteria for choosing a partition

- Number of separator arcs
- Balanced size of partitions
- Number of almost-full flag vectors

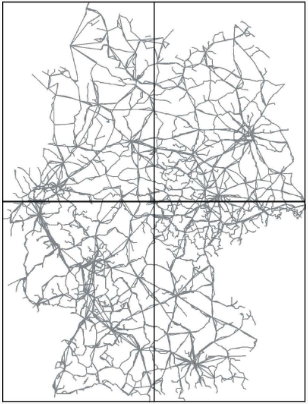
Rectangle

- Divide a bounding-box in a $x \times y$-grid
- Easy method
- Ignores structure of the graph
- Layout necessary



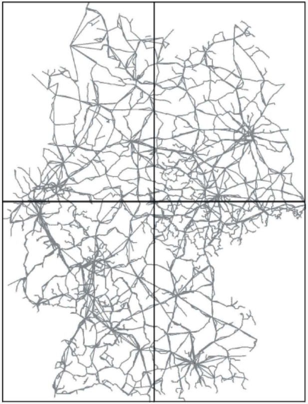
Quad-Trees

- Root node corresponds to the bounding-box
- Recursively divide each region in four quadrants
- Each quadrant is a child of the region
- Stop if there are less points in a region compared to a given upper bound



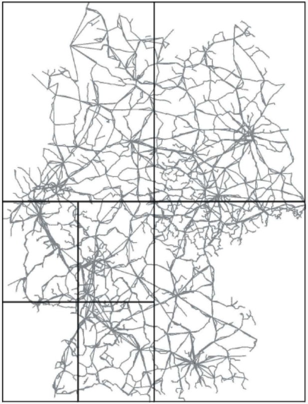
Quad-Trees

- Simple partition
- Almost balanced regionsize
- Layout necessary
- Separator set can be large



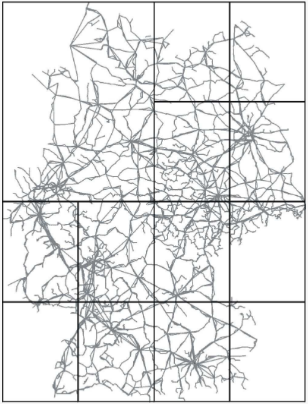
Quad-Trees

- Simple partition
- Almost balanced regionsize
- Layout necessary
- Separator set can be large



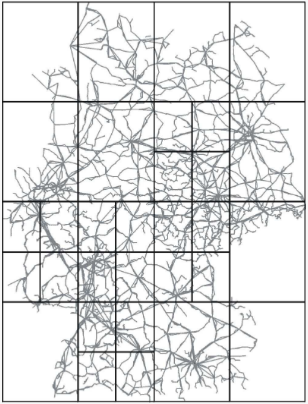
Quad-Trees

- Simple partition
- Almost balanced regionsize
- Layout necessary
- Separator set can be large



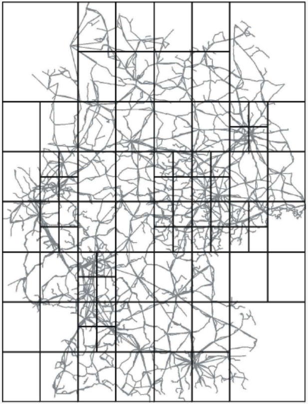
Quad-Trees

- Simple partition
- Almost balanced regionsize
- Layout necessary
- Separator set can be large



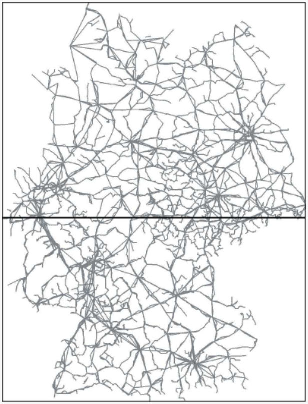
Quad-Trees

- Simple partition
- Almost balanced regionsize
- Layout necessary
- Separator set can be large



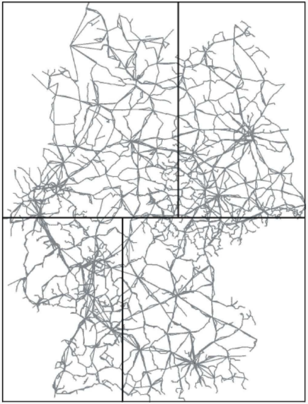
kd-Trees

- Recursively separate in two halves
- Alternate separating parallel to *x*- or *y*-axis
- Median for the separating line
- Layout necessary
- Balanced regionsize
- Separator set can be large



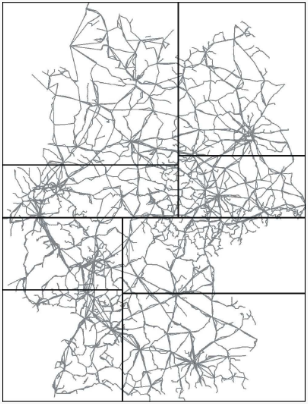
kd-Trees

- Recursively separate in two halves
- Alternate separating parallel to x- or y-axis
- Median for the separating line
- Layout necessary
- Balanced regionsize
- Separator set can be large



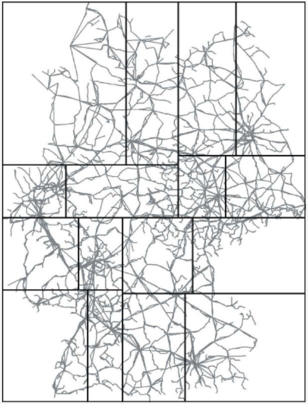
kd-Trees

- Recursively separate in two halves
- Alternate separating parallel to x- or y-axis
- Median for the separating line
- Layout necessary
- Balanced regionsize
- Separator set can be large



kd-Trees

- Recursively separate in two halves
- Alternate separating parallel to x- or y-axis
- Median for the separating line
- Layout necessary
- Balanced regionsize
- Separator set can be large

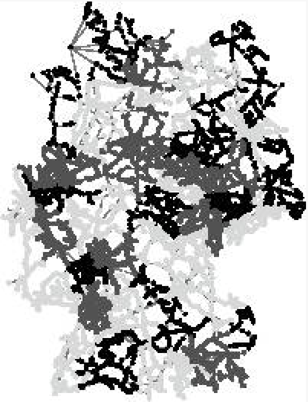


kd-Trees

- Recursively separate in two halves
- Alternate separating parallel to x- or y-axis
- Median for the separating line
- Layout necessary
- Balanced regionsize
- Separator set can be large

Multi-way arc separator

- Doesn't use 2d-layout
- Divides graph recursively in parts with minimal cut
- Balanced regionsize
- Almost minimal separator set

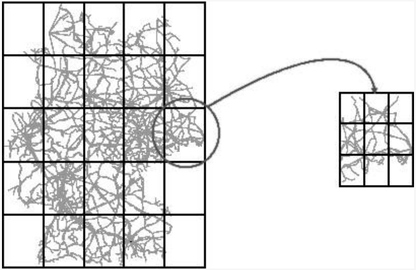


Space Consumption

- Trade-Off between one node per region and one region with all nodes
- Arc-Flags are an interpolation
- Finer partion increases speedup, but also space consumption and preprocessing time
- 225 regions proved to be sufficient for the german network

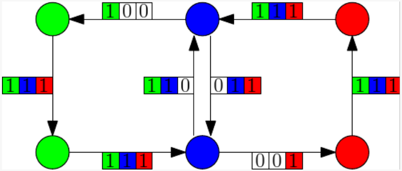
Multilevel partition

- Use a coarse partition
- Every region in the coarse partition is divided in smaller regions
- Flag vector has a local meaning
- Can be seen as a lossy compression of the flag vector



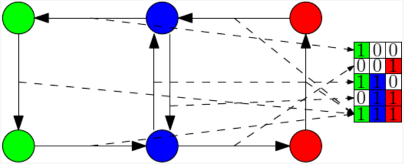
Data structure

- One flag per arc
- Number of combinations is bounded by $2^{|R|}$



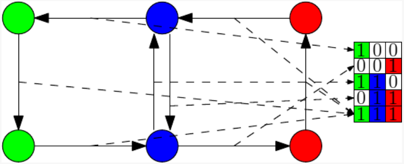
Data structure

- One flag per arc
- Number of combinations is bounded by $2^{|R|}$
- Idea: Store flags in array and use pointers



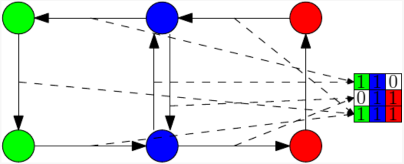
Compression

- We can flip bits from 0 to 1, but not from 1 to 0



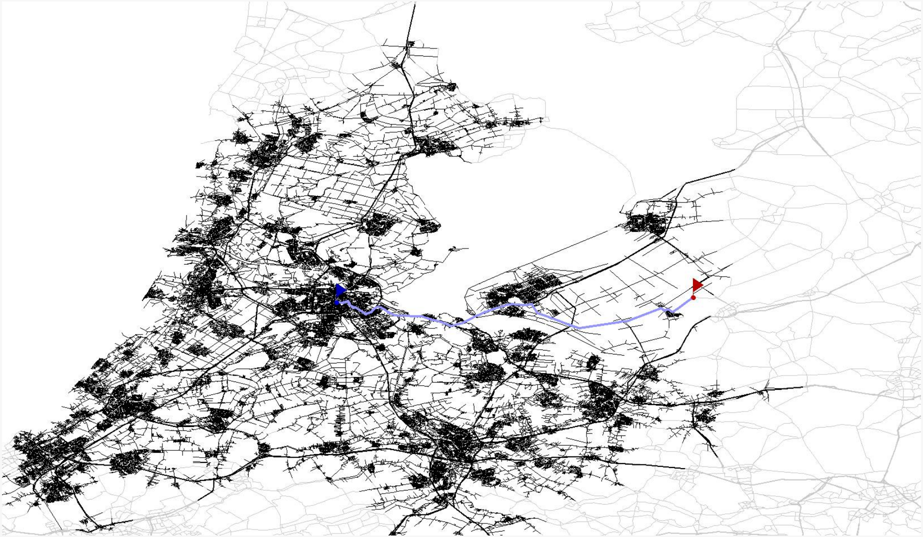
Compression

- We can flip bits from 0 to 1, but not from 1 to 0
- Idea: reduce number of arc flags by flipping the right bits
- Therefore we have a compression of the arc flags

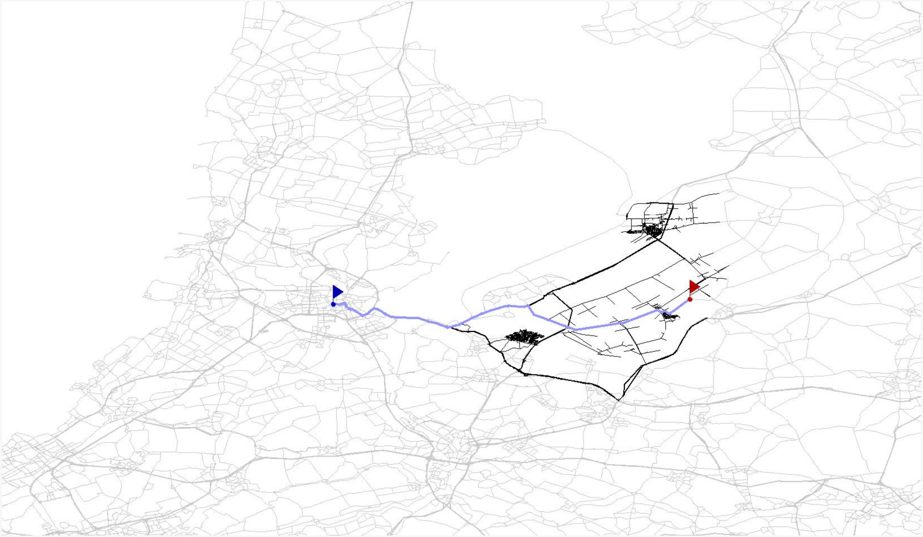


Outline

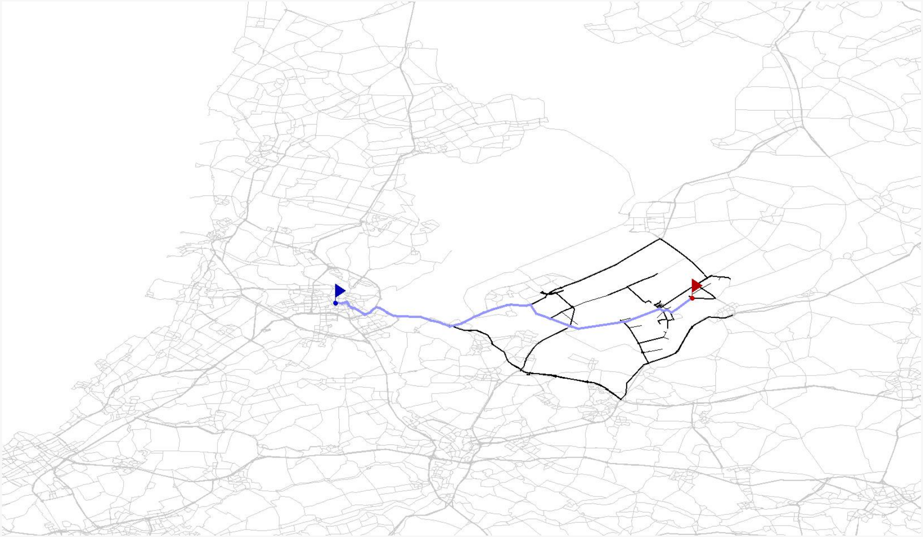
Search space



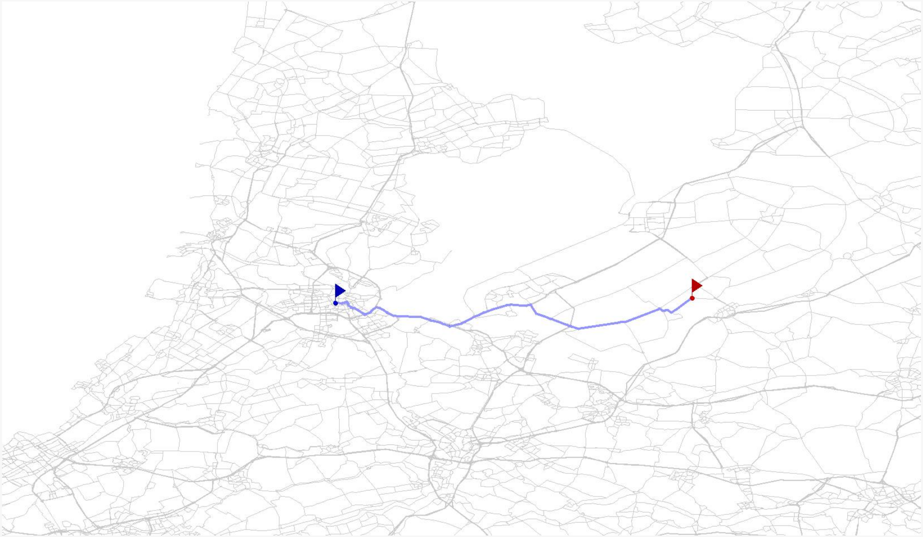
Coning-Effect



Search space using multilevel partition

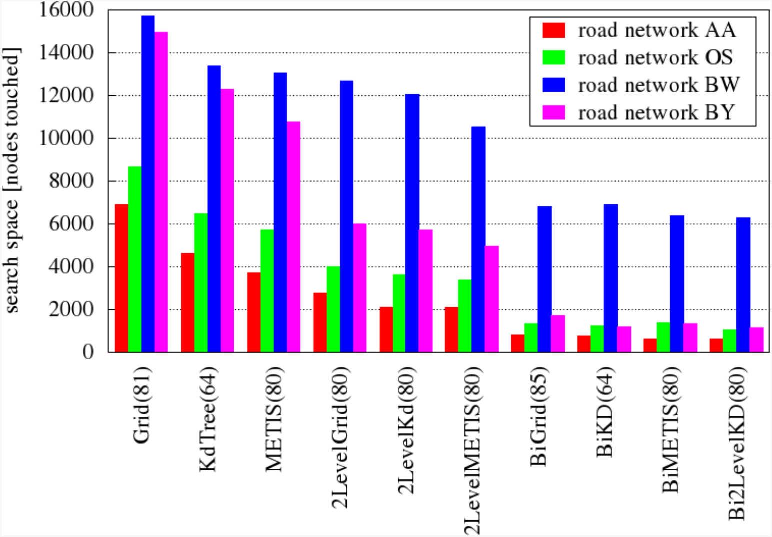


Search space using bidirectional arc flags

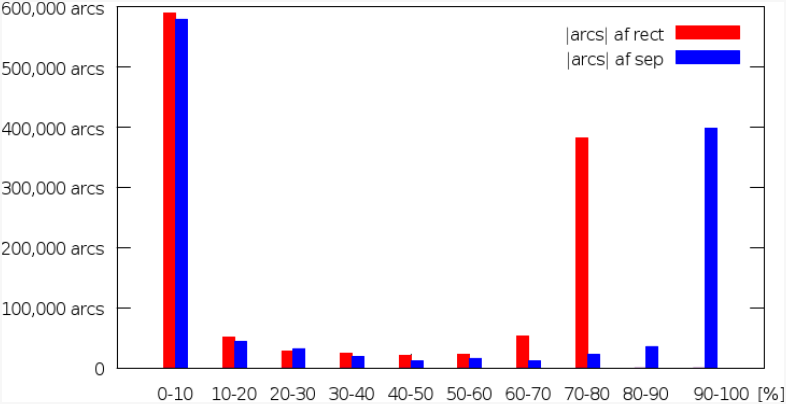


Outline

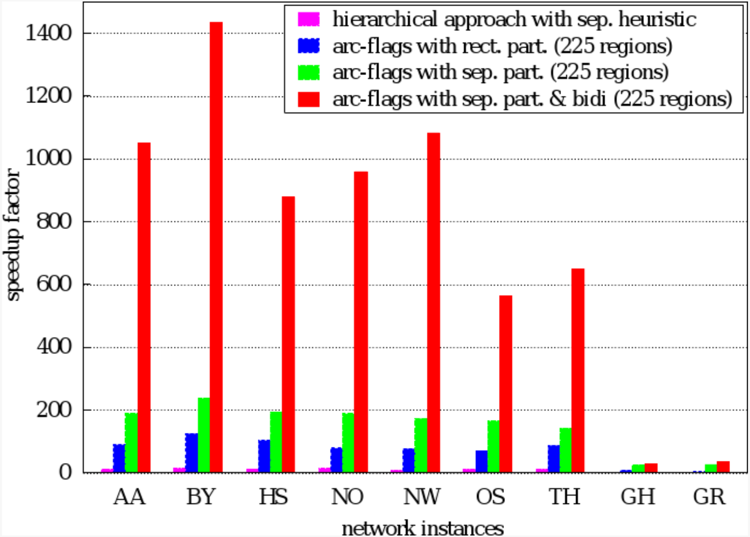
Search space using different partitions



Fill rate of flag vectors(474k nodes, 1.17m arcs)



Speed-Up compared to Dijkstra



Thank you!

Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling.
Fast Point-to-Point Shortest Path Computations with Arc-Flags.
In *Shortest Paths: Ninth DIMACS Implementation Challenge*, 2009.

Rolf H. Möhring, Heiko Schilling, Birk Schütz, Dorothea Wagner, Thomas Willhalm
Partitioning Graphs to Speed Up Dijkstra's Algorithm
Online Resource: http://www.math.tu-berlin.de/coga/people/schillin/pub/wea2005.2.pdf

Daniel Delling
Lecture: Algorithmen für Routenplanung
Online Resource: http://i11www.iti.uni-karlsruhe.de/_media/teaching/sommer2009/routenplanung/rp_vorlesung4