Karlsruher Institut für Technologie

Algorithmen für Routenplanung

4. Termin, Sommersemester 2011

Reinhard Bauer | 2. Mai 2011

Institut für Theoretische Informatik · Algorithmik I · Prof. Dr. Dorothea Wagner

Ideensammlung

Wie Suche zielgerichtet machen?

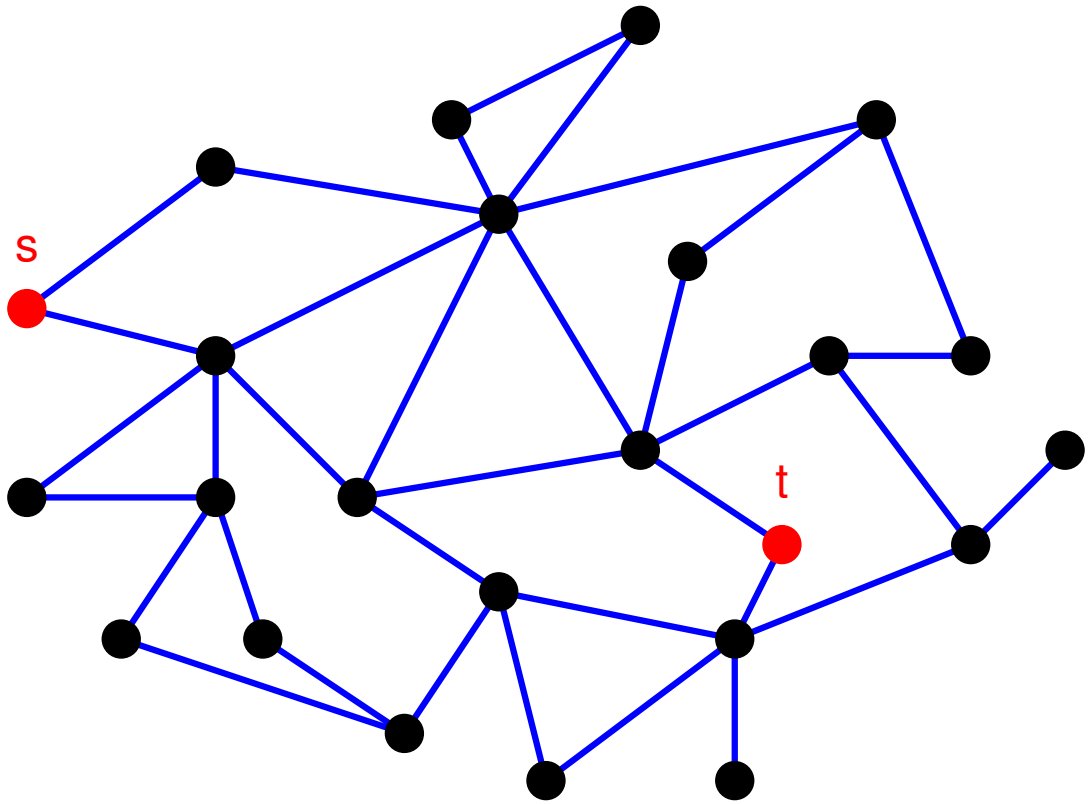
- Nichtbeachten von Kanten oder Knoten die in die “falsche” Richtung führen
- Reihenfolge in der Knoten besucht werden ändern

Jetzt: ersteres

Geometrische Container

Beobachtung:

- Subwege von kürzesten Wegen sind auch kürzeste Wege
- nicht jede Kante ist wichtig für ein bestimmtes Ziel



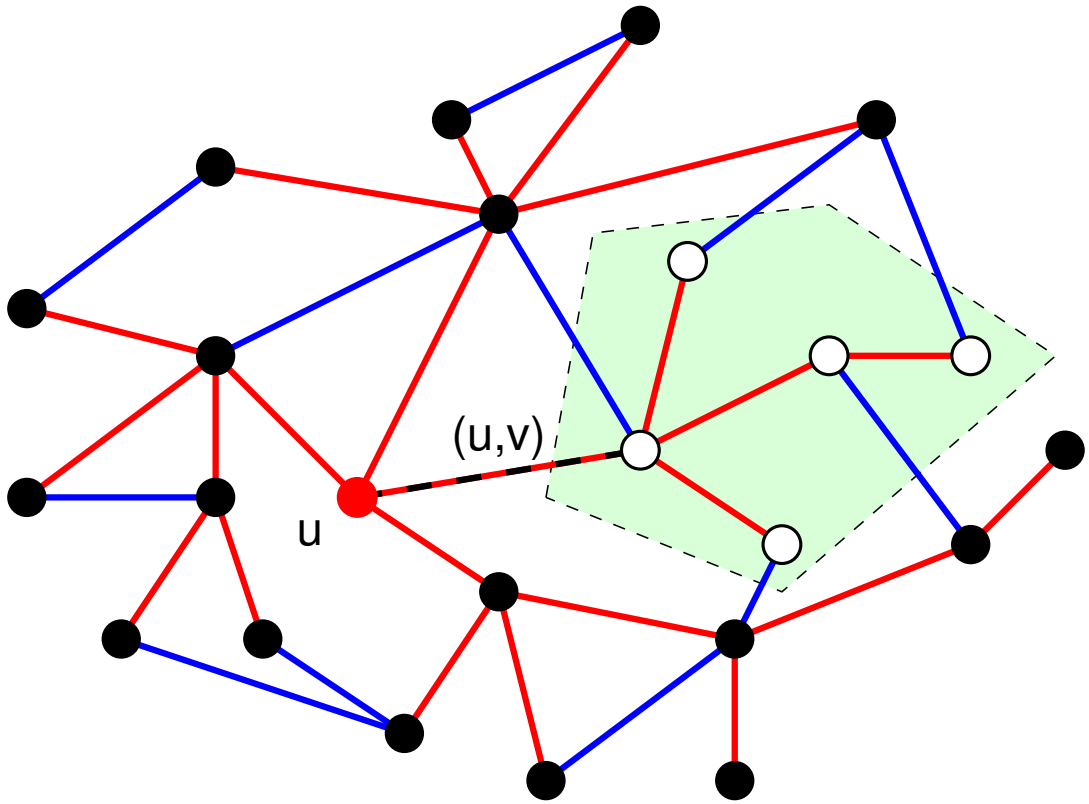
Idee:

- Speichere geometrisches Objekt für jede Kante, das alle Knoten des Unterbaums beinhaltet
- Relaxiere während Anfrage nur Kanten, für die Ziel *t* im Objekt ist

Geometrische Container

Beobachtung:

- Subwege von kürzesten Wegen sind auch kürzeste Wege
- nicht jede Kante ist wichtig für ein bestimmtes Ziel



Idee:

- Speichere geometrisches Objekt für jede Kante, das alle Knoten des Unterbaums beinhaltet
- Relaxiere während Anfrage nur Kanten, für die Ziel *t* im Objekt ist

Geometrische Container

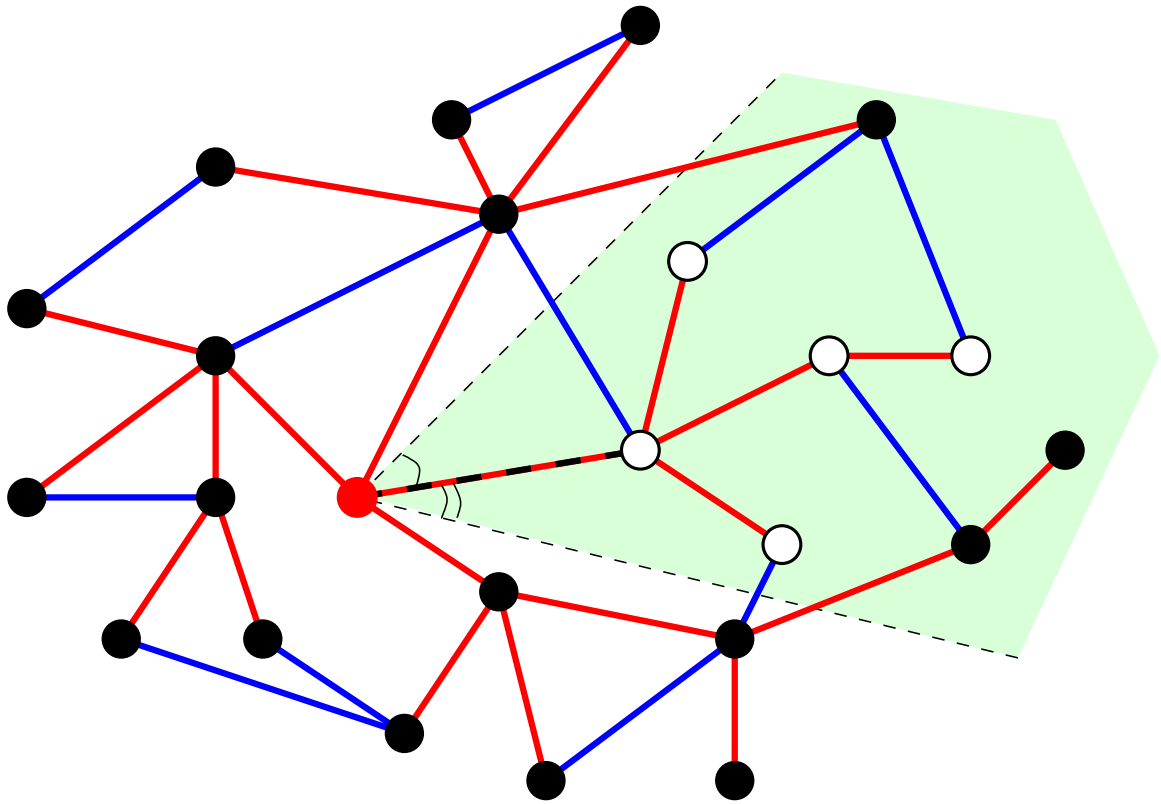
DIJKSTRA($G = (V, E, \text{len})$, s, Container $C()$)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      forall the edges e = (u, v) ∈ E do
6          if t ∉ C(e) then continue
7          if d[u] + len(e) < d[v] then
8              d[v] ← d[u] + len(e)
9              if v ∈ Q then Q.decreaseKey(v, d[v])
10             else Q.insert(v, d[v])
```

Formen

Viele Formen möglich:

- Winkel
- Winkel + Distanz
- Umgebenes Rechteck
- Konvexe Hülle



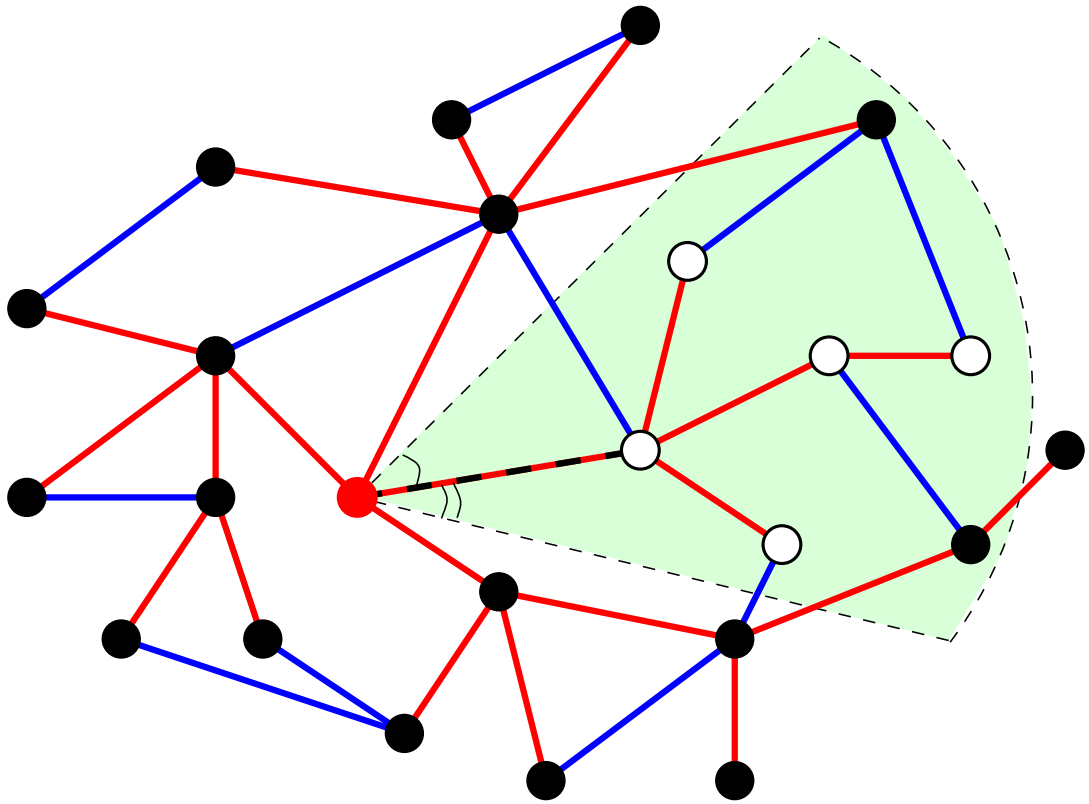
Trade-Off:

- Speicherplatz pro Kante
- Overhead zur Bestimmung ob t im Container liegt
- Umgebenes Rechteck scheint am besten zu sein

Formen

Viele Formen möglich:

- Winkel
- Winkel + Distanz
- Umgebenes Rechteck
- Konvexe Hülle



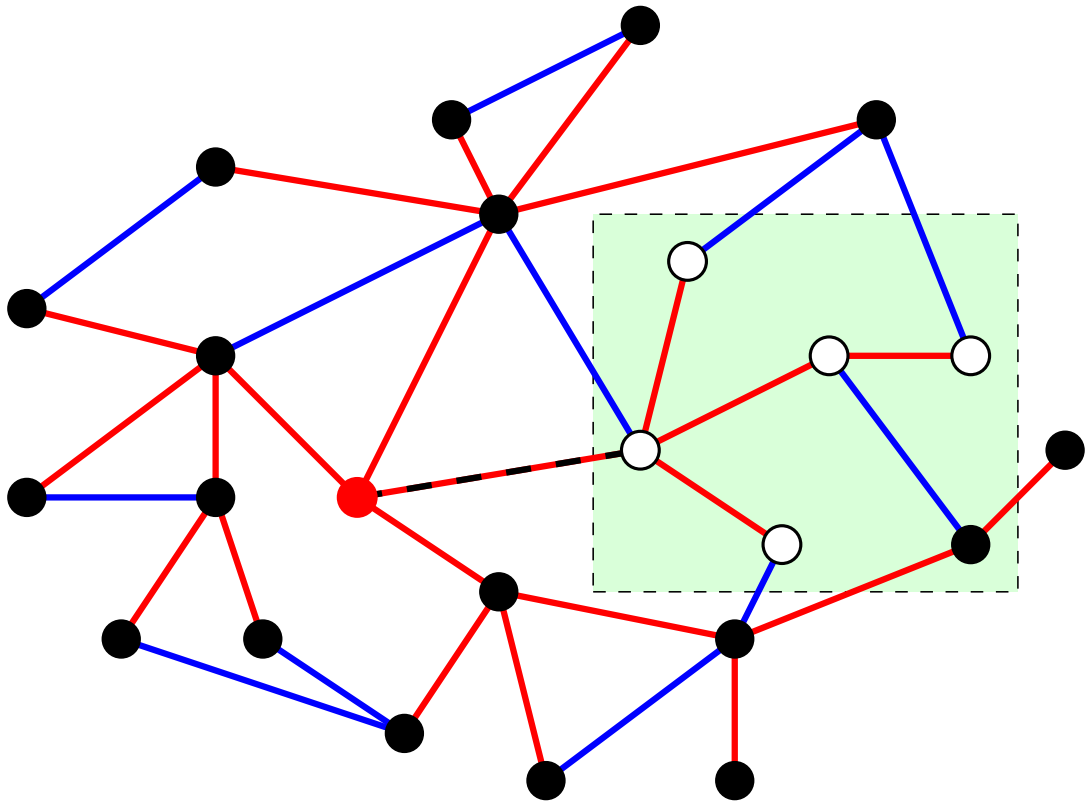
Trade-Off:

- Speicherplatz pro Kante
- Overhead zur Bestimmung ob t im Container liegt
- Umgebenes Rechteck scheint am besten zu sein

Formen

Viele Formen möglich:

- Winkel
- Winkel + Distanz
- Umgebenes Rechteck
- Konvexe Hülle



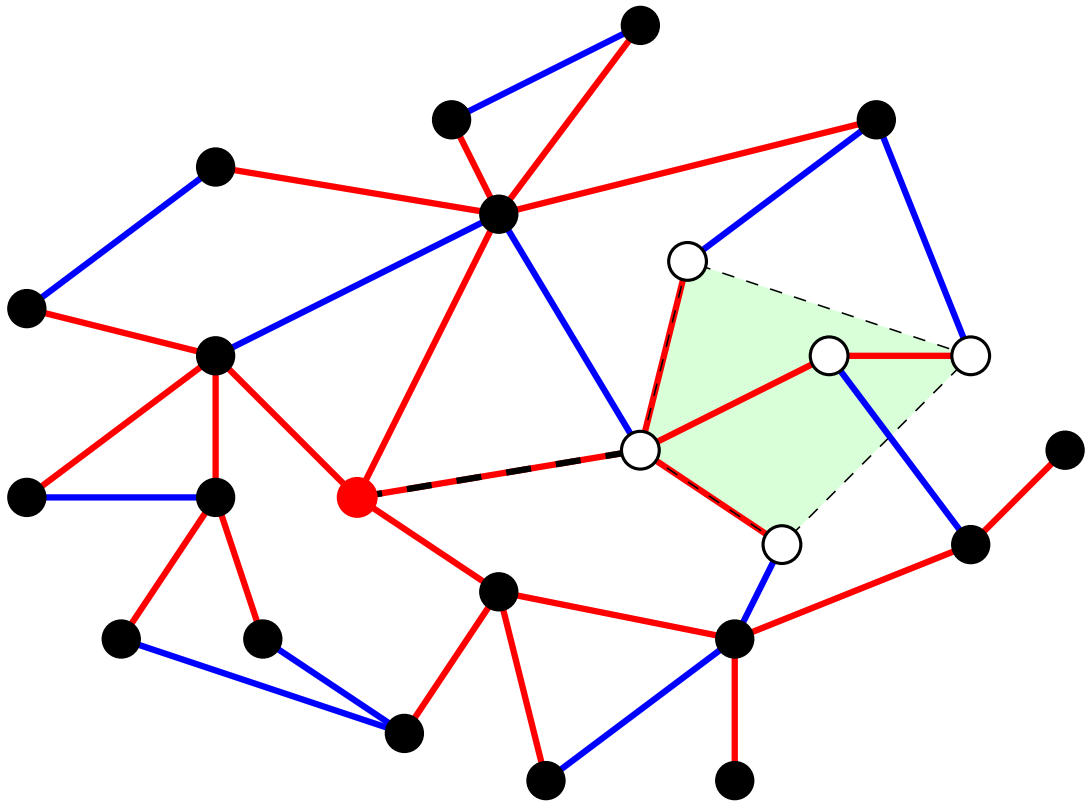
Trade-Off:

- Speicherplatz pro Kante
- Overhead zur Bestimmung ob t im Container liegt
- Umgebenes Rechteck scheint am besten zu sein

Formen

Viele Formen möglich:

- Winkel
- Winkel + Distanz
- Umgebenes Rechteck
- Konvexe Hülle



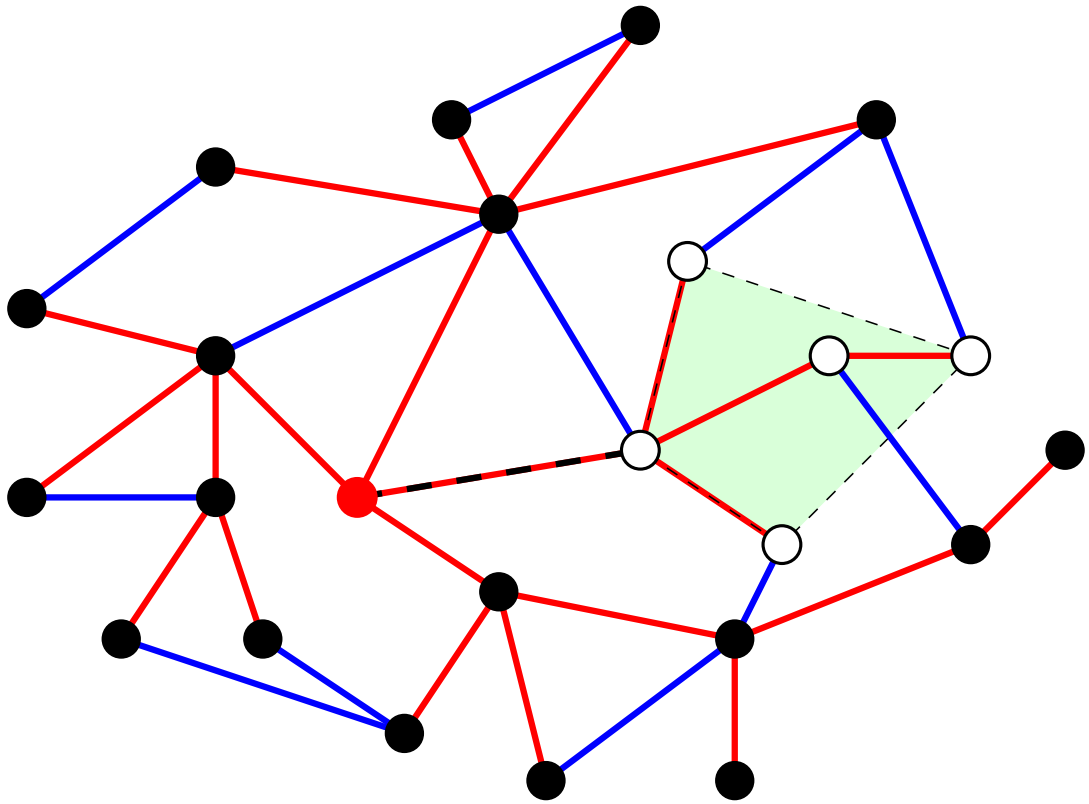
Trade-Off:

- Speicherplatz pro Kante
- Overhead zur Bestimmung ob t im Container liegt
- Umgebenes Rechteck scheint am besten zu sein

Formen

Viele Formen möglich:

- Winkel
- Winkel + Distanz
- Umgebenes Rechteck
- Konvexe Hülle



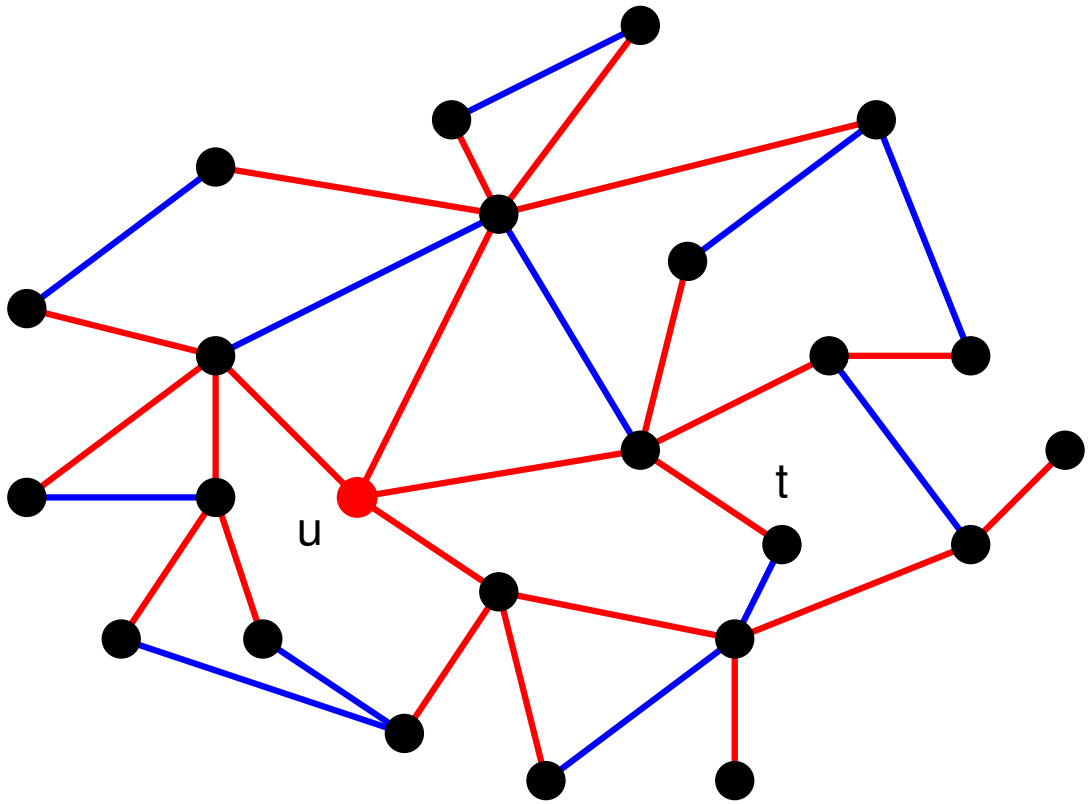
Trade-Off:

- Speicherplatz pro Kante
- Overhead zur Bestimmung ob t im Container liegt
- Umgebenes Rechteck scheint am besten zu sein

Vorberechnung

Vorgehen:

- für jeden Knoten einen kürzeste Wege Baum berechnen
- dann für jede Kante den minimalen Container berechnen
- speicher Container an der Kante



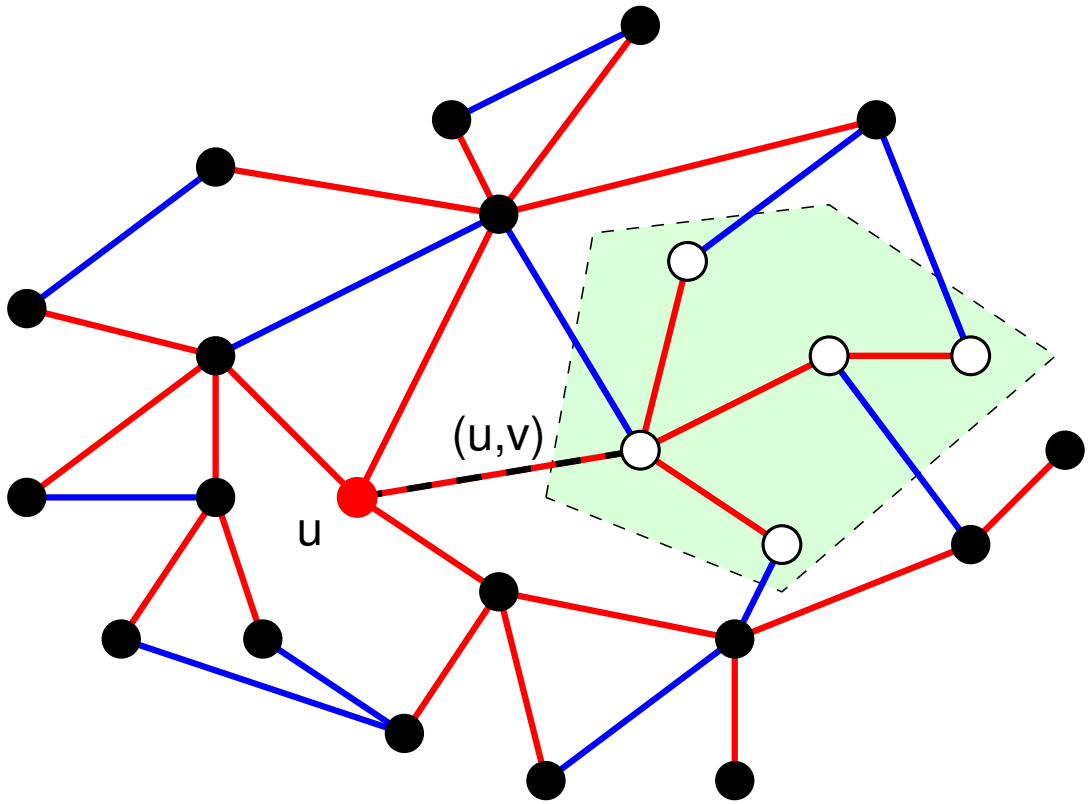
Zeit- und Platz-Bedarf:

- m Container, Größe abhängig von Komplexität des Containers
- n Dijkstras + m Berechnungen der Container

Vorberechnung

Vorgehen:

- für jeden Knoten einen kürzeste Wege Baum berechnen
- dann für jede Kante den minimalen Container berechnen
- speicher Container an der Kante



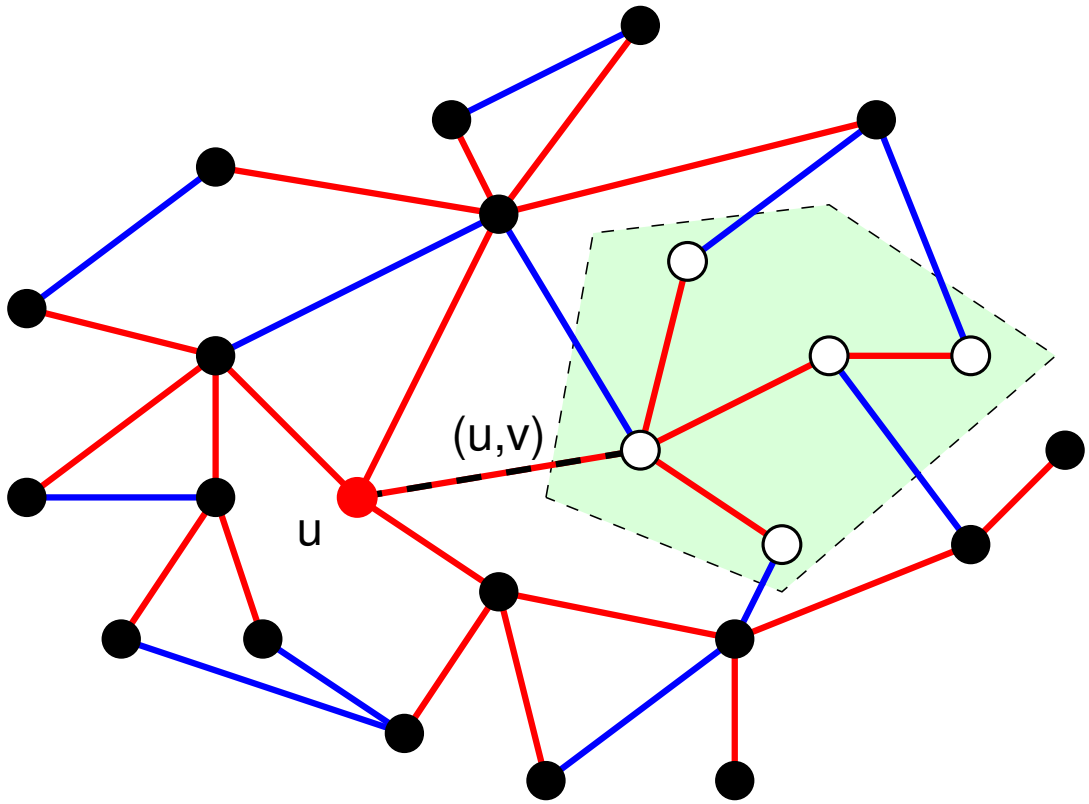
Zeit- und Platz-Bedarf:

- m Container, Größe abhängig von Komplexität des Containers
- n Dijkstras + m Berechnungen der Container

Vorberechnung

Vorgehen:

- für jeden Knoten einen kürzeste Wege Baum berechnen
- dann für jede Kante den minimalen Container berechnen
- speicher Container an der Kante



Zeit- und Platz-Bedarf:

- *m* Container, Größe abhängig von Komplexität des Containers
- *n* Dijkstras + *m* Berechnungen der Container

Diskussion

Vorteile:

- einfacher Anfrage-Algorithmus
- Beschleunigung um bis zu Faktor 40
- Vorberechnung einfach (basiert v.a. auf Dijkstra's Algorithmus)

Nachteile:

- Hohe Laufzeit der Vorberechnung
- daher nicht berechenbar auf sehr großen Netzen
  (ungefähr 500 Jahre für das Straßennetzwerk von Europa)
- Vorberechnungsplatz: m Container (Box: 2 Punkte)
- Einbettung der Graphen nötig
- Container können sehr ungenau sein

Diskussion

Vorteile:

- einfacher Anfrage-Algorithmus
- Beschleunigung um bis zu Faktor 40
- Vorberechnung einfach (basiert v.a. auf Dijkstra's Algorithmus)

Nachteile:

- Hohe Laufzeit der Vorberechnung
- daher nicht berechenbar auf sehr großen Netzen (ungefähr 500 Jahre für das Straßennetzwerk von Europa)
- Vorberechnungsplatz: m Container (Box: 2 Punkte)
- Einbettung der Graphen nötig
- Container können sehr ungenau sein

Diskussion

Vorteile:

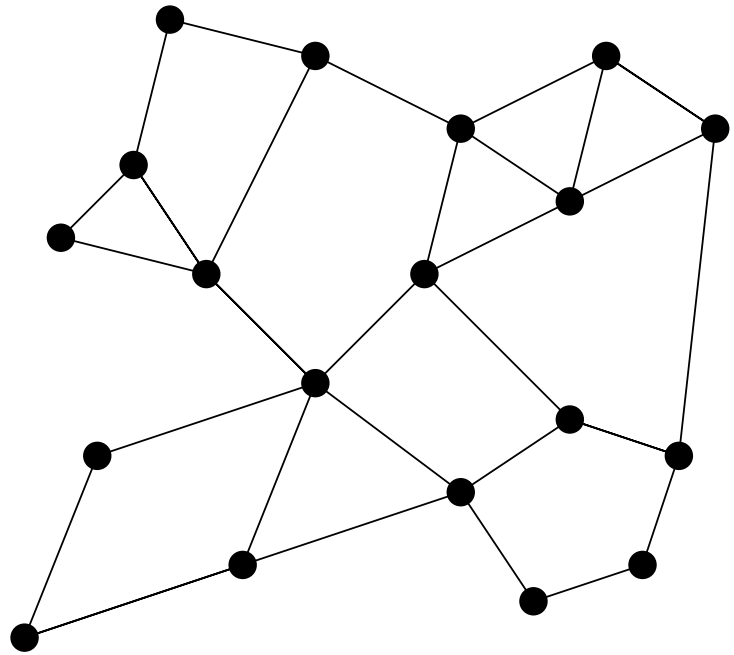
- einfacher Anfrage-Algorithmus
- Beschleunigung um bis zu Faktor 40
- Vorberechnung einfach (basiert v.a. auf Dijkstra's Algorithmus)

Nachteile:

- Hohe Laufzeit der Vorberechnung
- daher nicht berechenbar auf sehr großen Netzen
  (ungefähr 500 Jahre für das Straßennetzwerk von Europa)
- Vorberechnungsplatz: m Container (Box: 2 Punkte)
- Einbettung der Graphen nötig
- Container können sehr ungenau sein

Technik: Arc-Flags [Lauther 04]

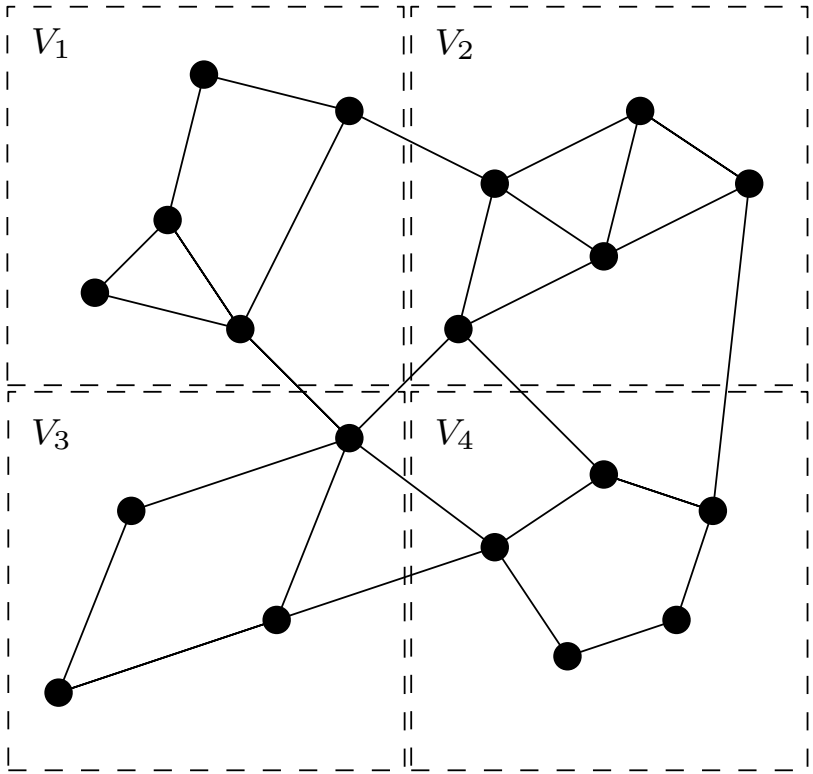
Offline-Phase:



Technik: Arc-Flags [Lauther 04]

Offline-Phase:

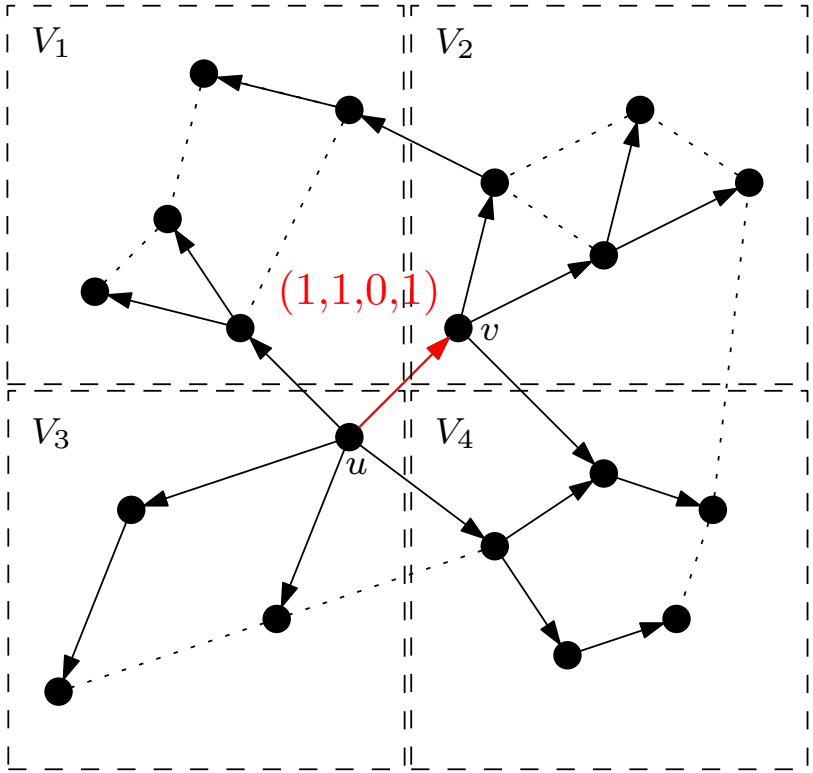
- Partitioniere den Graphen



Technik: Arc-Flags [Lauther 04]

Offline-Phase:

- Partitioniere den Graphen
- Für jede Kante (u, v) und jede Zelle i:
  Führt ein kürzestester Wegen von (u, v) nach i?



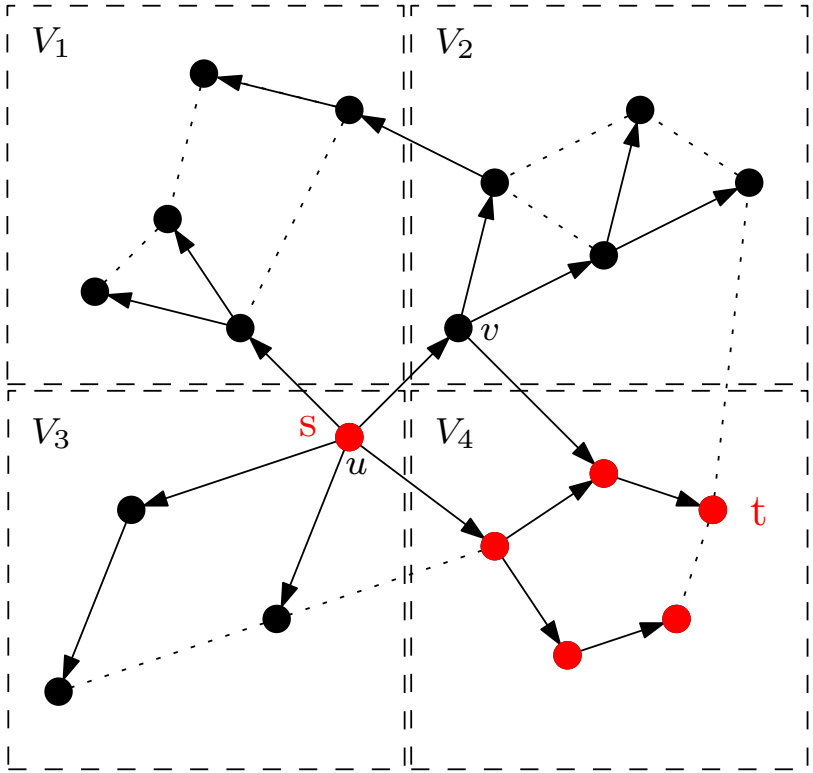
Technik: Arc-Flags [Lauther 04]

Offline-Phase:

- Partitioniere den Graphen
- Für jede Kante (u, v) und jede Zelle i:
  Führt ein kürzestester Wegen von (u, v) nach i?

Online-Phase:

- Wie Dijkstra's Algorithmus
- Nur Kanten ins Zielgebiet



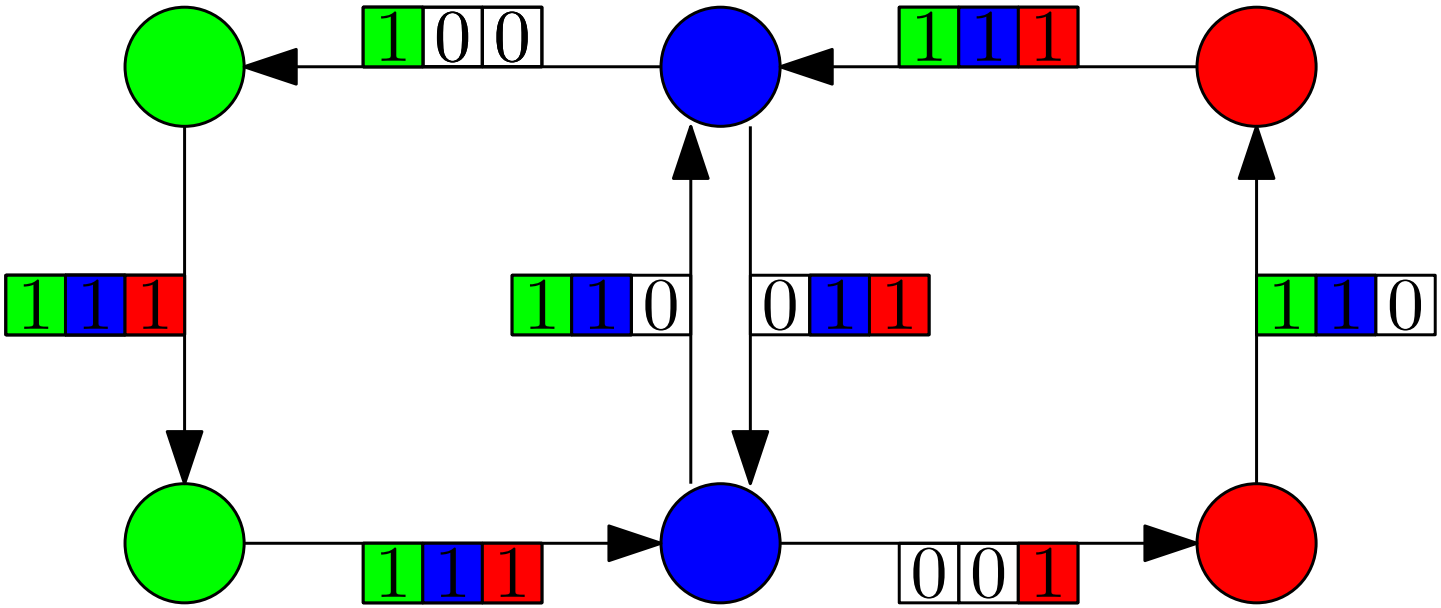
Anfragen

ARC-FLAG-DIJKSTRA($G = (V, E)$, s, t, Arc-Flags $AF_\cdot(\cdot)$)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      forall the edges e = (u, v) ∈ E do
6          if d[u] + len(e) < d[v] and AF_T(e) = true then
7              d[v] ← d[u] + len(e)
8              if v ∈ Q then Q.decreaseKey(v, d[v])
9              else Q.insert(v, d[v])
```

Flaggenberechnung: Erster Versuch

- für jeden Knoten $t \in V$
  - konstruiere Kürzeste-Wege Baum T auf Rückwärtsgraph mit Wurzel v
  - für jede Kante (u, v) in T
    - Setze flag von Region von t und Kante (u, v) auf true



Flaggenberechnung: Erster Versuch

Problem: Vorberechnung wieder zu teuer!

Flaggenberechnung: Randknoten

Randknoten

Ein Randknoten ist ein Knoten v, so dass es eine Kante (u, v) gibt mit u in anderer Zelle als v.

Beobachtung: Man muss eine Zelle durch einen Randknoten „betreten“

- Für jede Kante (u, v) setze das Flag für die Zelle von v auf true
- Berechne Kürzeste-Wege-Bäume nur für Randknoten

Übersicht Vorberechungsarten

In aufsteigender Güte:

- Kürzeste-Wege-Baum von jedem Knoten
- Kürzeste-Wege-Baum von jedem Randknoten
- Sogennante „Zentralisierte Bäume“ (nicht Stoff der Vorlesung)
- Schnelles APSP (all-pairs shortest paths, kommt noch)

Partitionierung

Anforderungen:

- balanciert

mögliche Partitionen:

- Gitter
- Quad-Baum
  - iterativ in 4 Zellen unterteilen
- kd-Baum
  - Verallgemeinerung von Quad-Baum

weitere Anforderungen?

Partitionierung

Anforderungen:

- balanciert

mögliche Partitionen:

- Gitter
- Quad-Baum
  - iterativ in 4 Zellen unterteilen
- kd-Baum
  - Verallgemeinerung von Quad-Baum

weitere Anforderungen?

Partitionierung

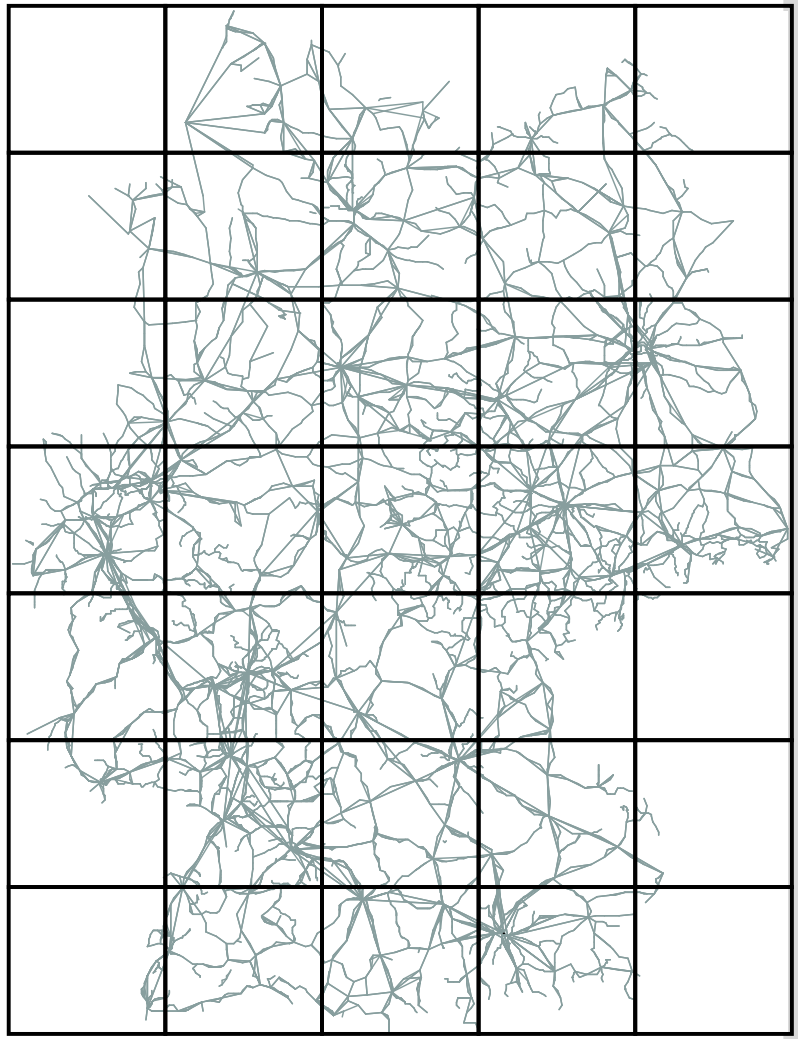
Anforderungen:

- balanciert

mögliche Partitionen:

- Gitter
- Quad-Baum
  - iterativ in 4 Zellen unterteilen
- kd-Baum
  - Verallgemeinerung von Quad-Baum

weitere Anforderungen?



Partitionierung

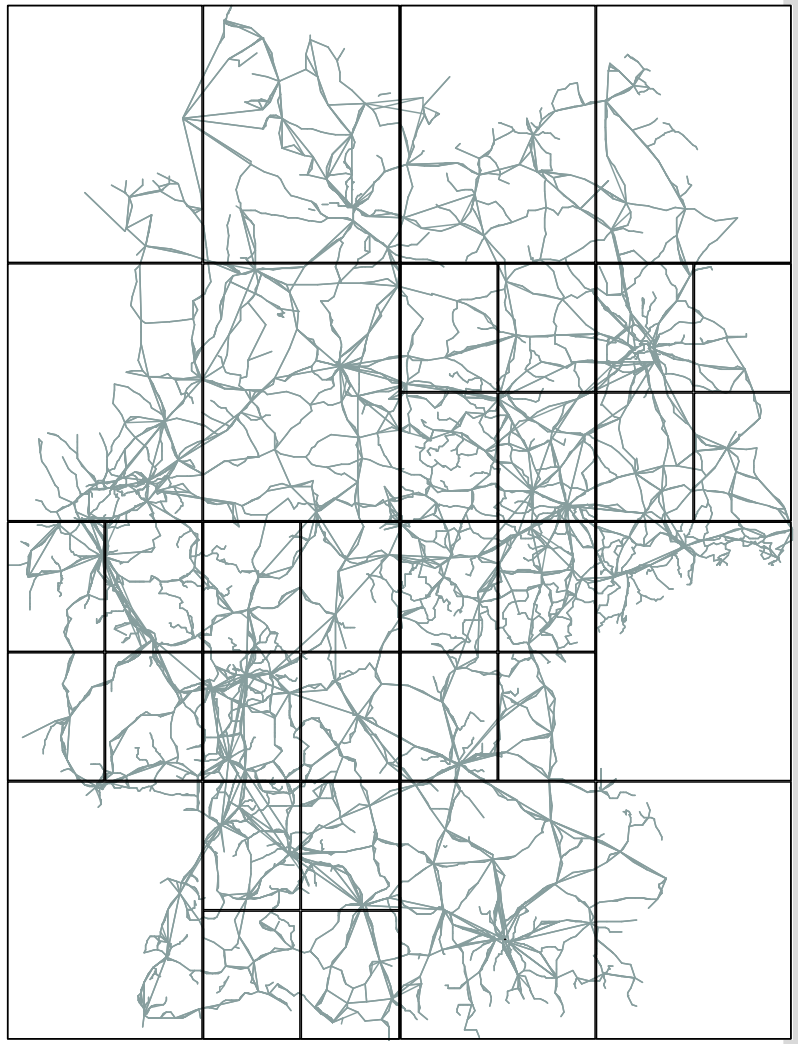
Anforderungen:

- balanciert

mögliche Partitionen:

- Gitter
- Quad-Baum
  - iterativ in 4 Zellen unterteilen
- kd-Baum
  - Verallgemeinerung von Quad-Baum

weitere Anforderungen?



Partitionierung

Anforderungen:

- balanciert

mögliche Partitionen:

- Gitter
- Quad-Baum
  - iterativ in 4 Zellen unterteilen
- kd-Baum
  - Verallgemeinerung von Quad-Baum

weitere Anforderungen?

Partitionierung

Anforderungen:

- balanciert

mögliche Partitionen:

- Gitter
- Quad-Baum
  - iterativ in 4 Zellen unterteilen
- kd-Baum
  - Verallgemeinerung von Quad-Baum

weitere Anforderungen?

Partitionierung

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Black-Box Partitionierer:

- benutzen keine Einbettung
- oft: teilen rekursiv Graphen in k Teile mit kleinem Schnitt
- bringen gute Lösungen für Arc-Flags

Partitionierung

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Black-Box Partitionierer:

- benutzen keine Einbettung
- oft: teilen rekursiv Graphen in k Teile mit kleinem Schnitt
- bringen gute Lösungen für Arc-Flags

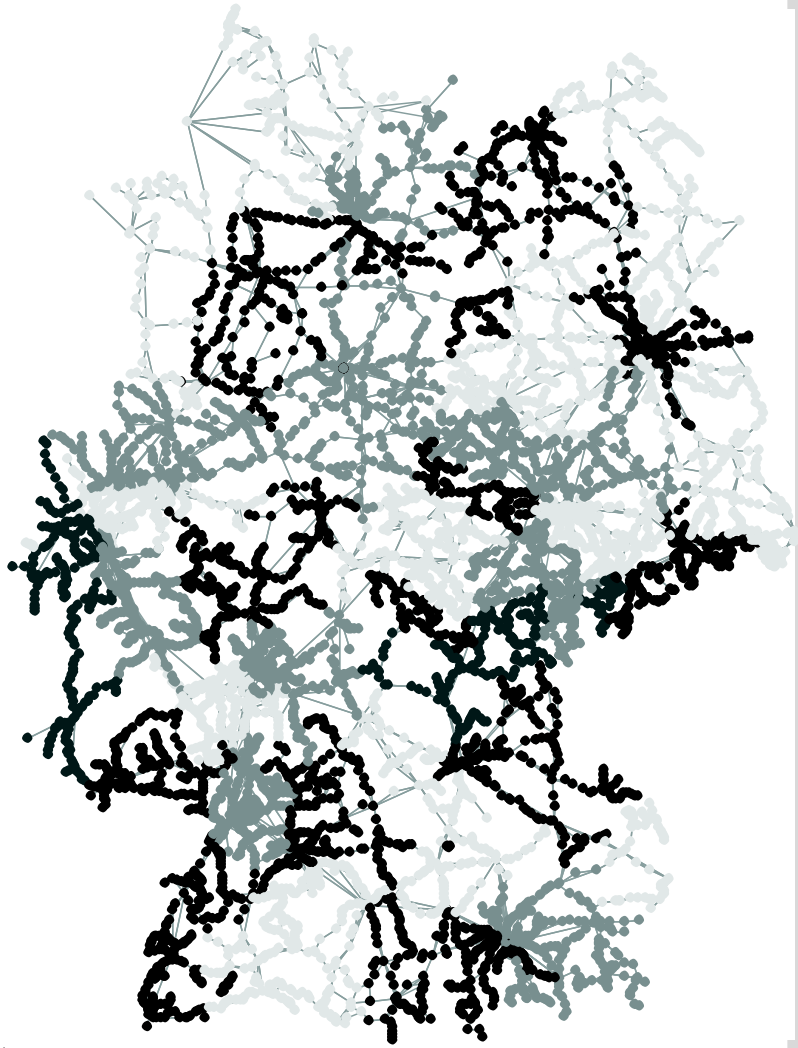
Partitionierung

Anforderungen:

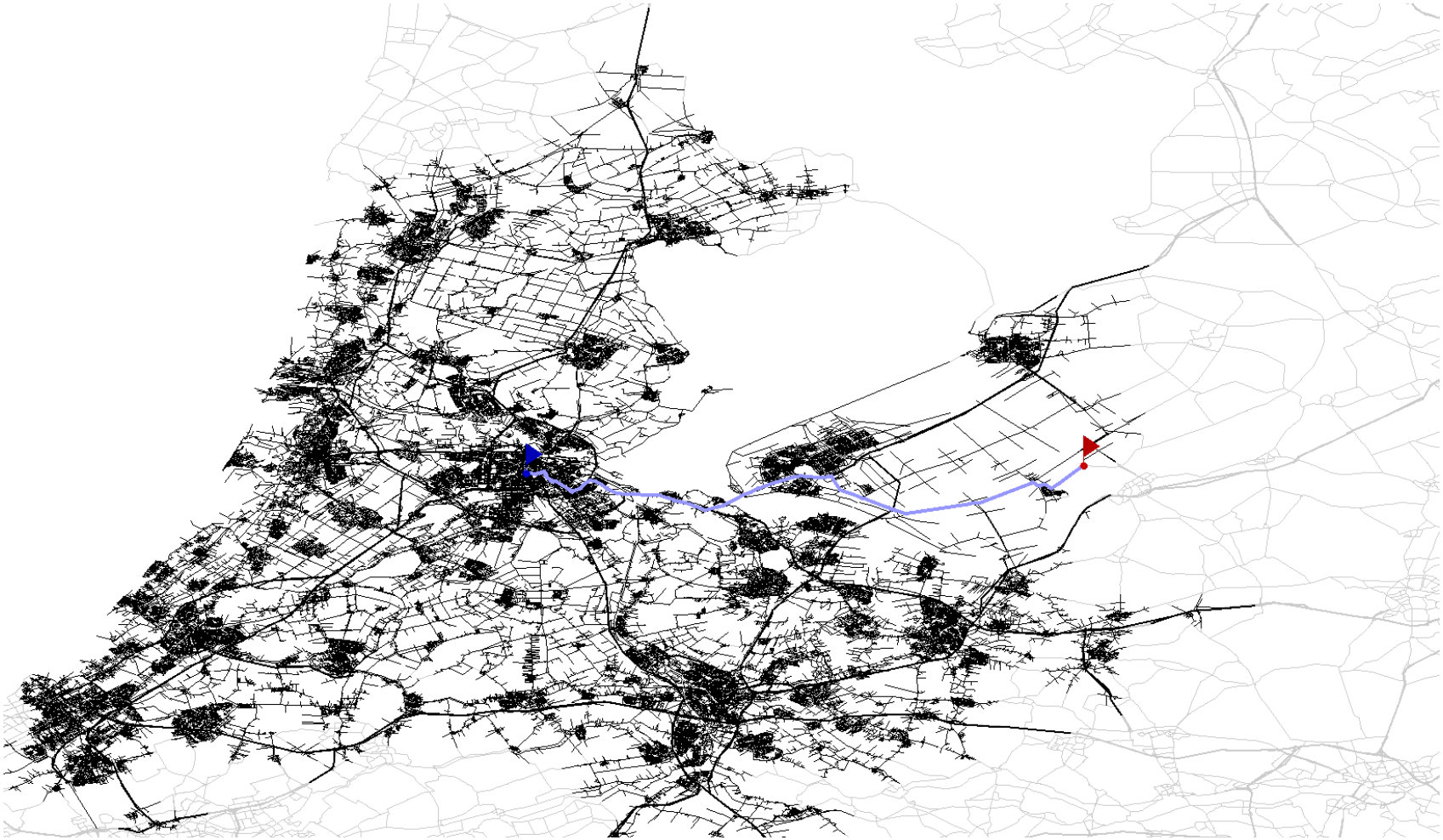
- ausbalanciert
- wenige Randknoten
- zusammenhängend

Black-Box Partitionierer:

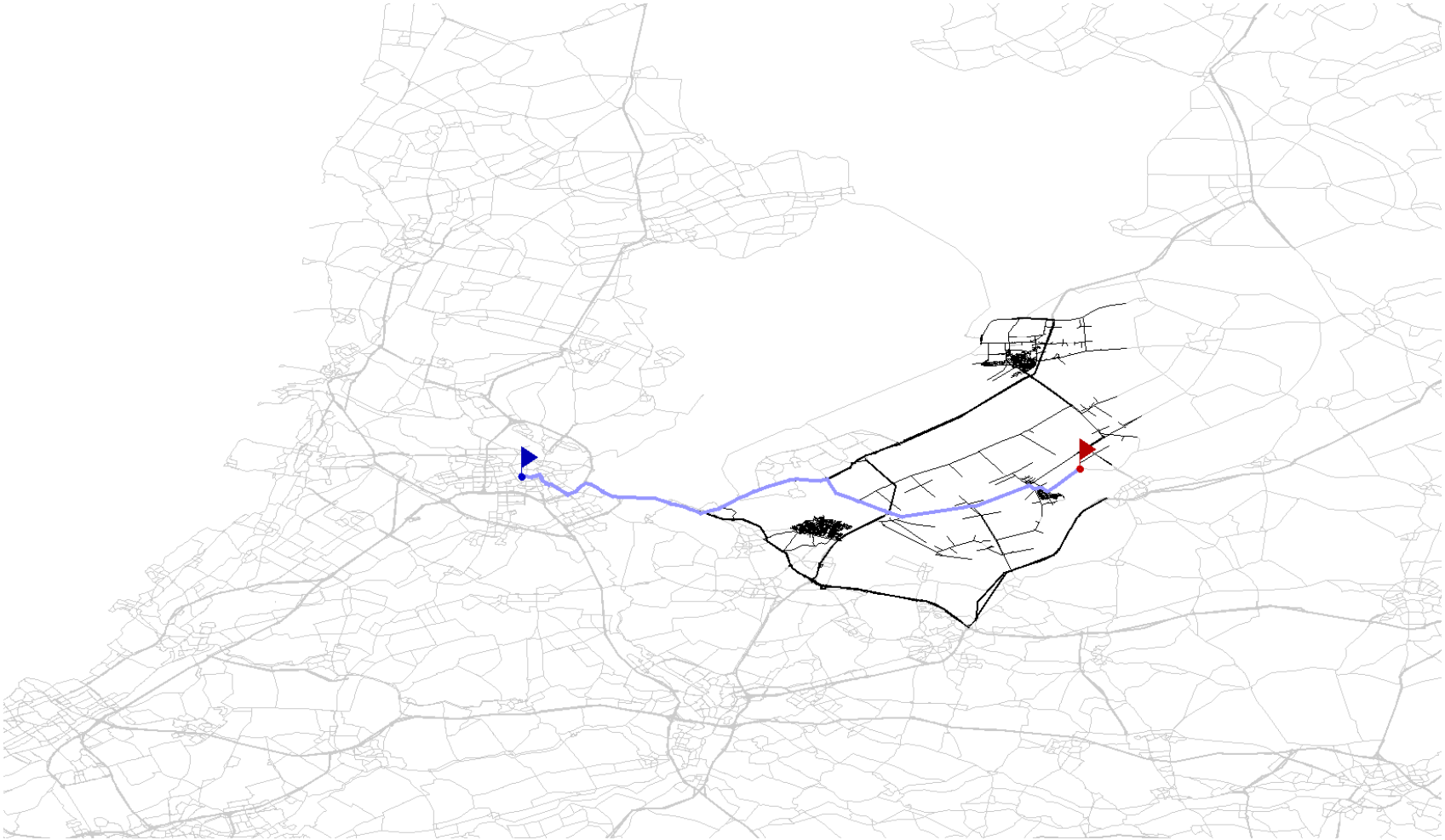
- benutzen keine Einbettung
- oft: teilen rekursiv Graphen in k Teile mit kleinem Schnitt
- bringen gute Lösungen für Arc-Flags



Suchraum



Suchraum



Diskussion

Vorteile:

- einfacher Anfrage Algorithmus
- leicht on-top auf bestehende Systeme zu setzen
- ausreichende Beschleunigung

Nachteile:

- hohe Vorberechnungsdauer (wird durch schnelles APSP aufgehoben)
- Coning-Effekt:
  - mehr Flaggen gesetzt, wenn nah an der Zielzelle
  - alle Flaggen in Zielzelle gesetzt
  - Anfragen in einer Zelle ohne Beschleunigung

Automatisches Setzen von Flaggen

Beobachtung:

- Für manche Kanten kann man die Flaggen automatisch setzen

Angehangene Bäume:

- Kanten zur Wurzel hin haben alle Flaggen gesetzt
- Kanten von Wurzel weg haben nur eine Flagge gesetzt
- also können die Bäume vor der Vorberechnung vom Graphen entfernt werden
- Knotenzahl verringert sich um 1 Drittel

Automatisches Setzen von Flaggen

Beobachtung:

- Für manche Kanten kann man die Flaggen automatisch setzen

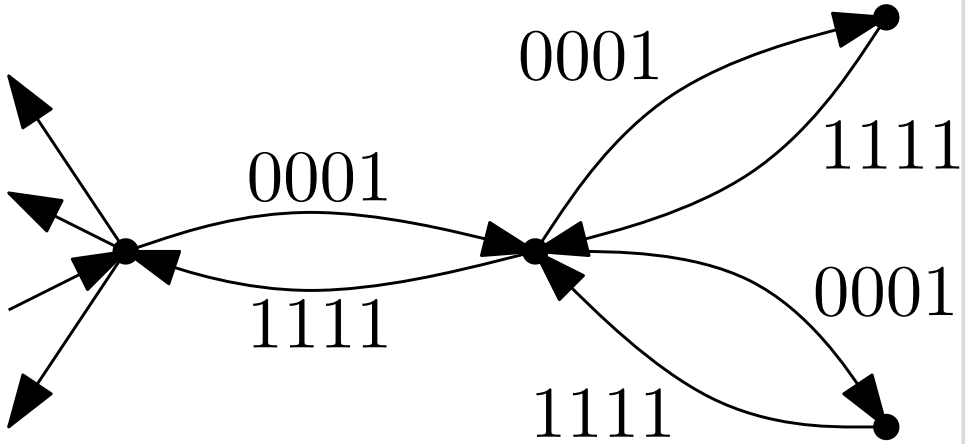
Angehangene Bäume:

- Kanten zur Wurzel hin haben alle Flaggen gesetzt
- Kanten von Wurzel weg haben nur eine Flagge gesetzt
- also können die Bäume vor der Vorberechnung vom Graphen entfernt werden
- Knotenzahl verringert sich um 1 Drittel

Automatisches Setzen von Flaggen

Beobachtung:

- Für manche Kanten kann man die Flaggen automatisch setzen



Angehangene Bäume:

- Kanten zur Wurzel hin haben alle Flaggen gesetzt
- Kanten von Wurzel weg haben nur eine Flagge gesetzt
- also können die Bäume vor der Vorberechnung vom Graphen entfernt werden
- Knotenzahl verringert sich um 1 Drittel

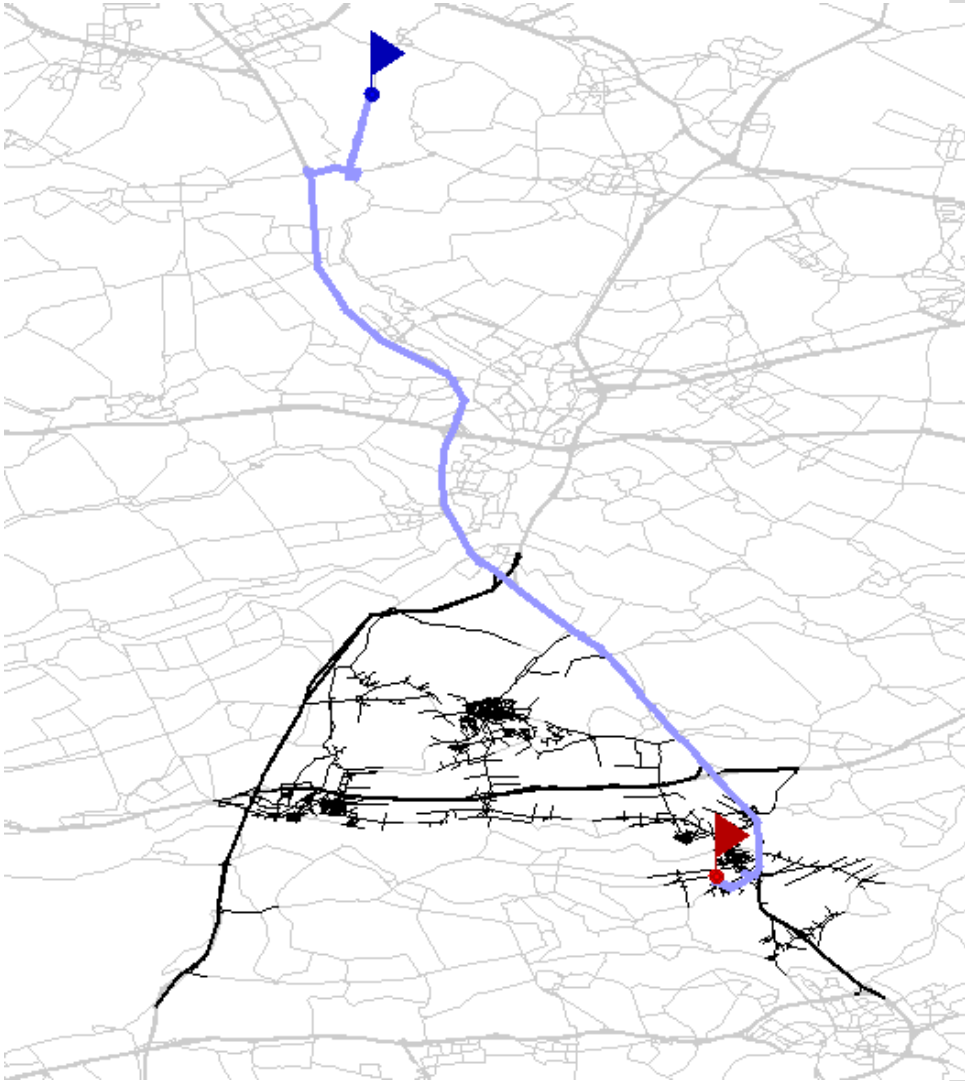
Coning-Effekt

Beobachtung:

- lange Zeit nur eine Kante wichtig
- daher immer nur ein Knoten in der Queue
- aber: je näher an der Zelle, desto mehr Kanten werden wichtig
- Suche fächert sich auf
- in Zelle werden dann alle Kanten relaxiert

Zwei Ansätze:

- bidirektionale Flags
- multi-level Flags



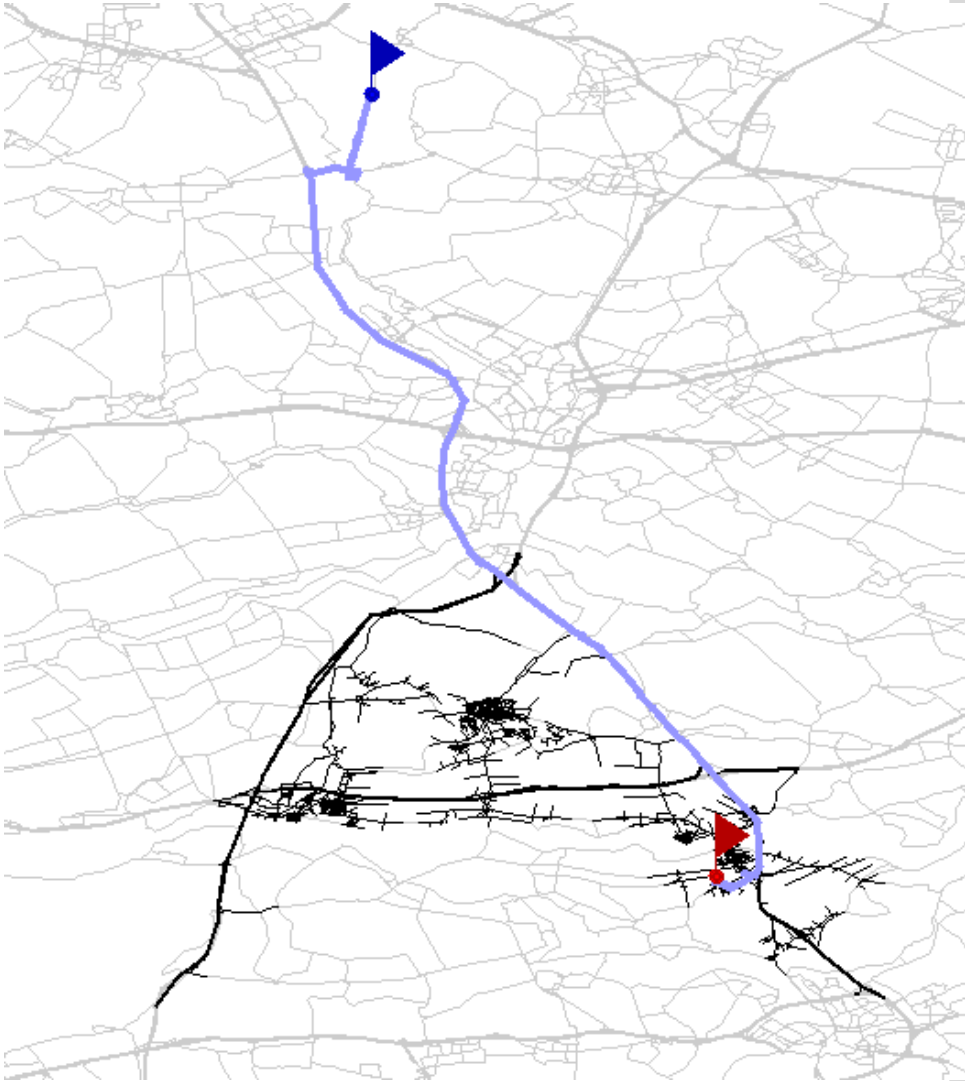
Coning-Effekt

Beobachtung:

- lange Zeit nur eine Kante wichtig
- daher immer nur ein Knoten in der Queue
- aber: je näher an der Zelle, desto mehr Kanten werden wichtig
- Suche fächert sich auf
- in Zelle werden dann alle Kanten relaxiert

Zwei Ansätze:

- bidirektionale Flags
- multi-level Flags



Bidirektionale Arc-Flags

Vorberechung:

- Vorwärts- und Rückwärtsflaggen
- Rückwärtsflaggen können analog für eingehende Kanten berechnet werden
- Vorberechungszeit in gerichteten Graphen erhöht sich um Faktor 2

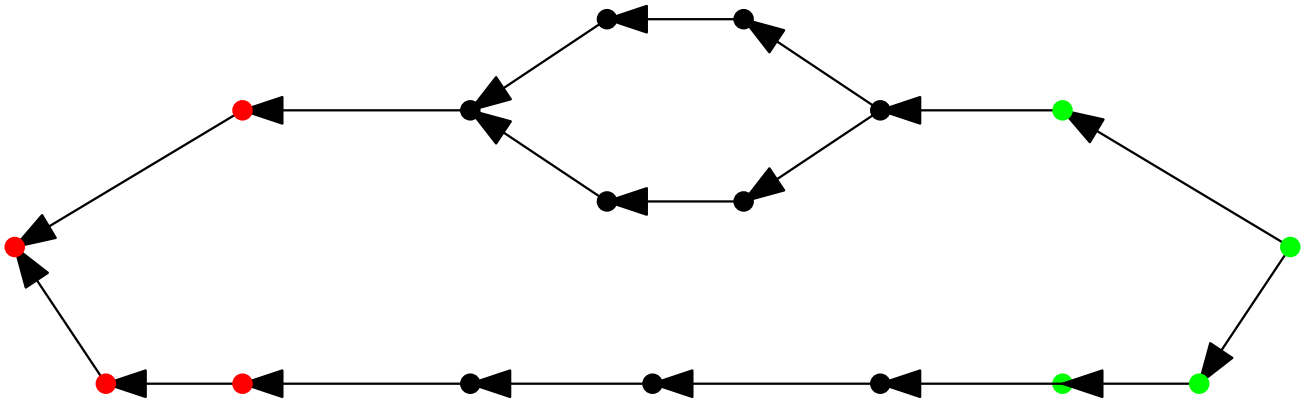
Anfrage:

- bidirektional:
  - Vorwärtssuche relaxiert nur Kanten mit Flagge für T
  - Rückwärtssuche nur Kanten mit Flaggen für S
- normales Stopp-Kriterium von bidirektionalem Dijkstra

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



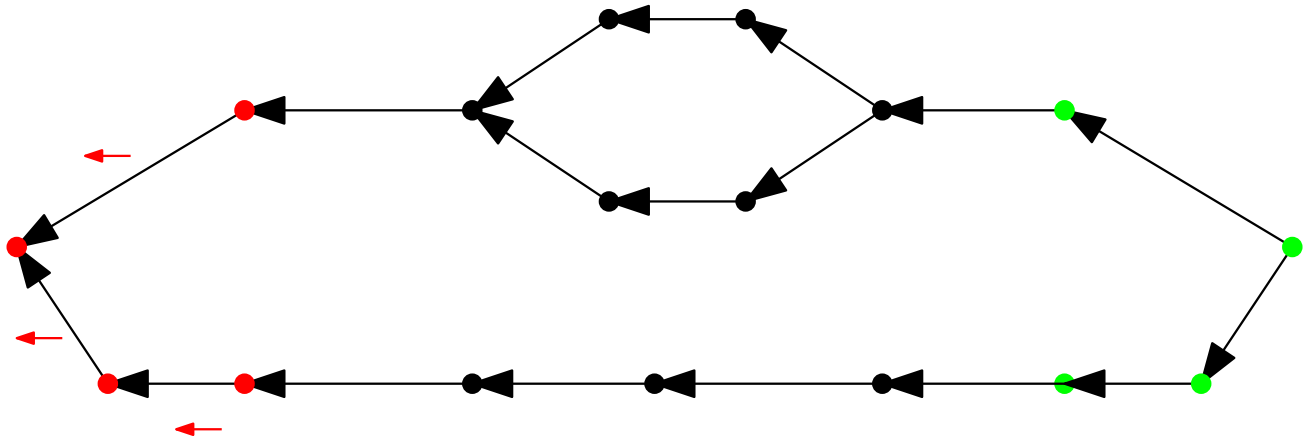
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



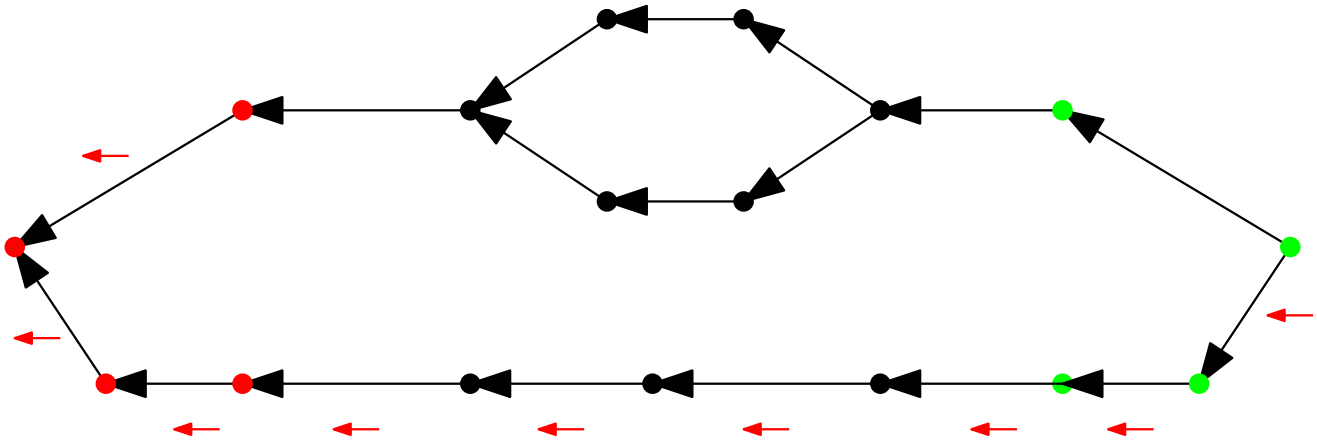
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



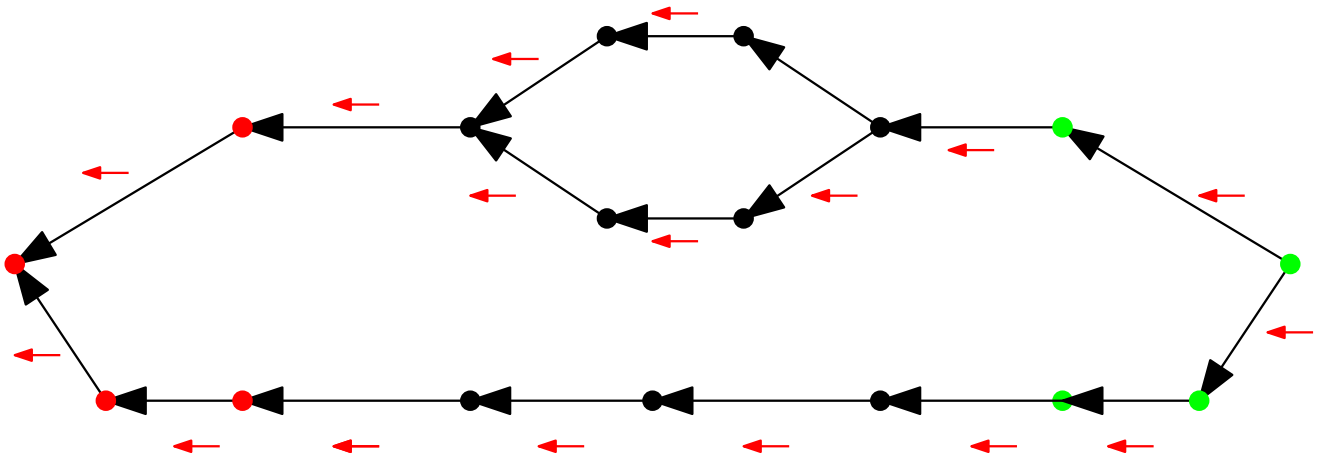
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



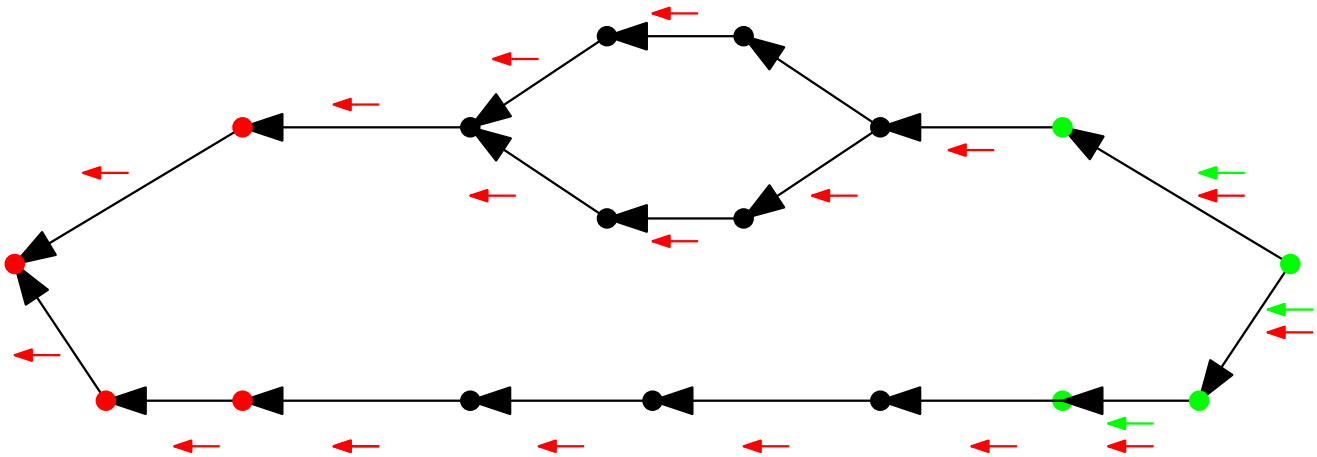
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



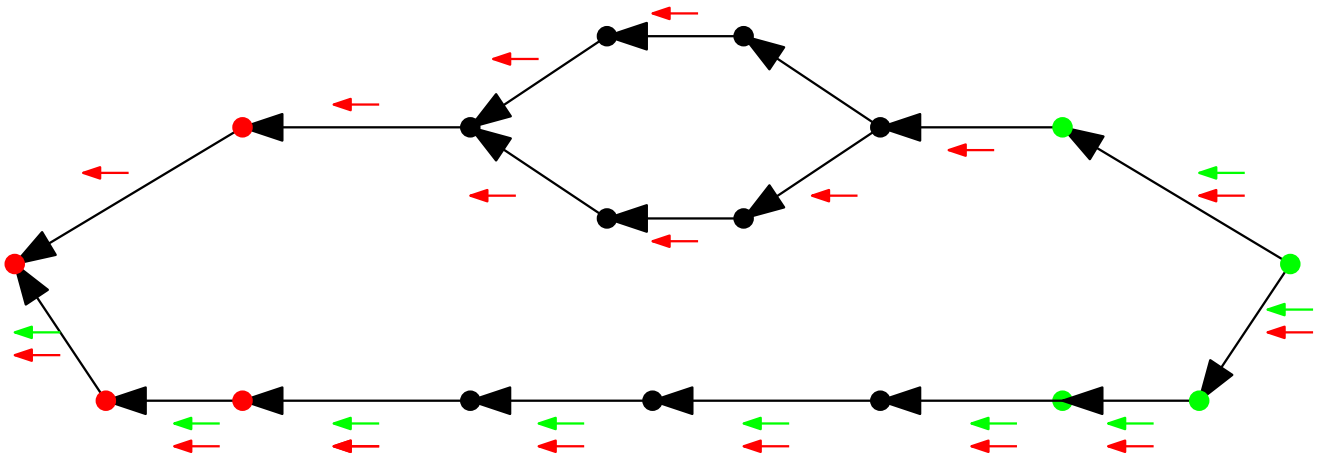
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



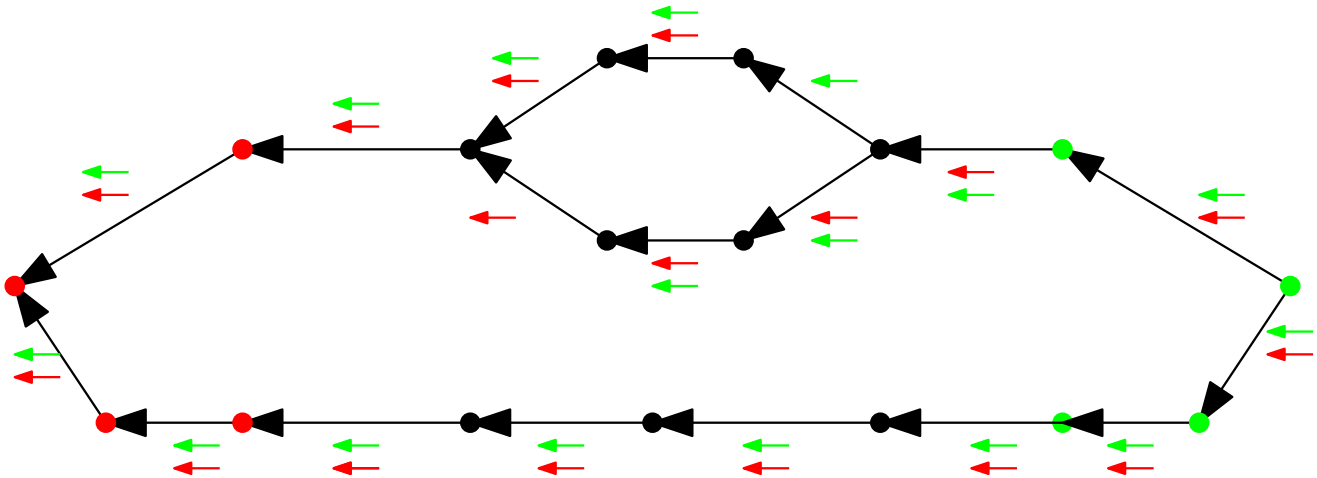
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



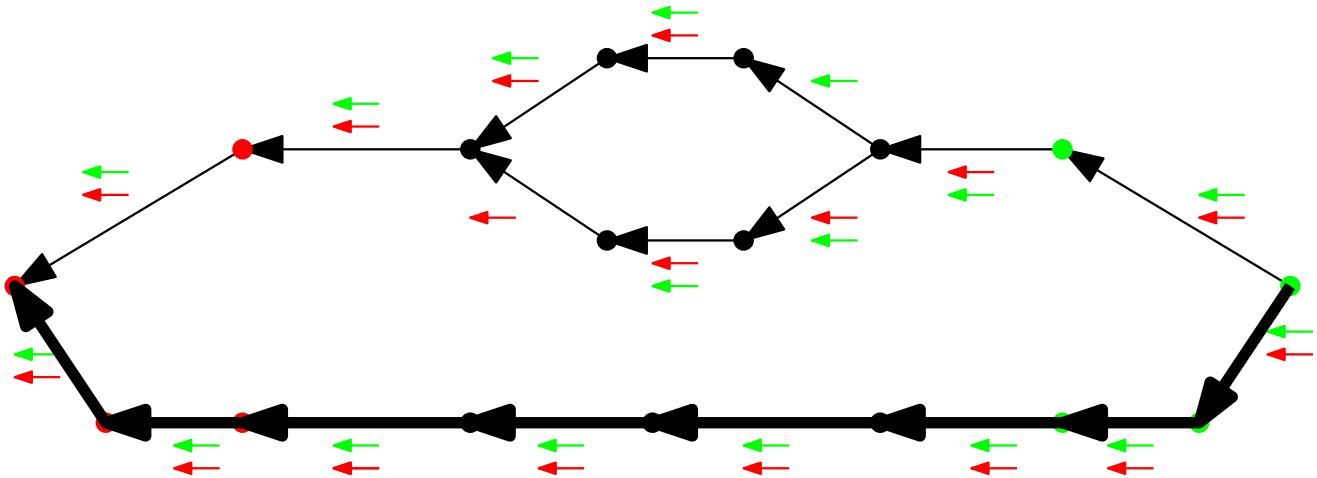
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



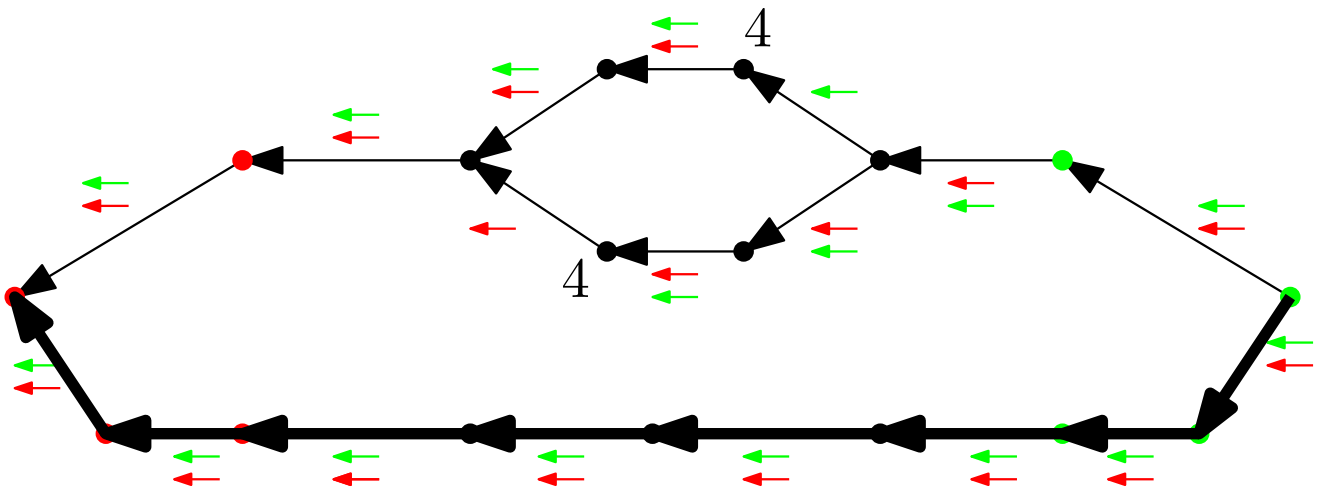
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



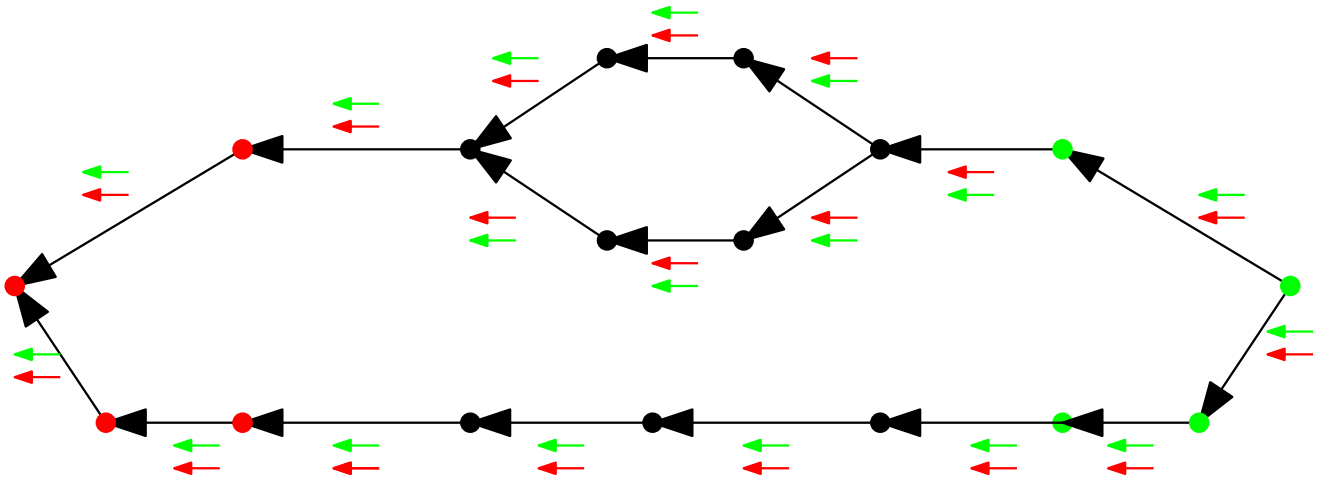
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



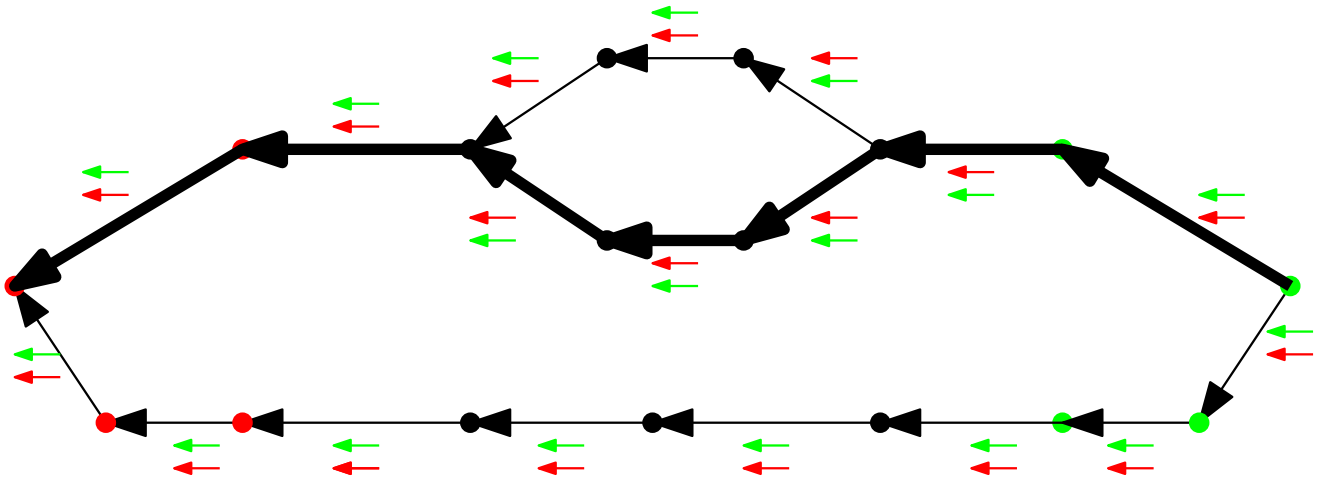
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Bidirektionale Arc-Flags

Problem:

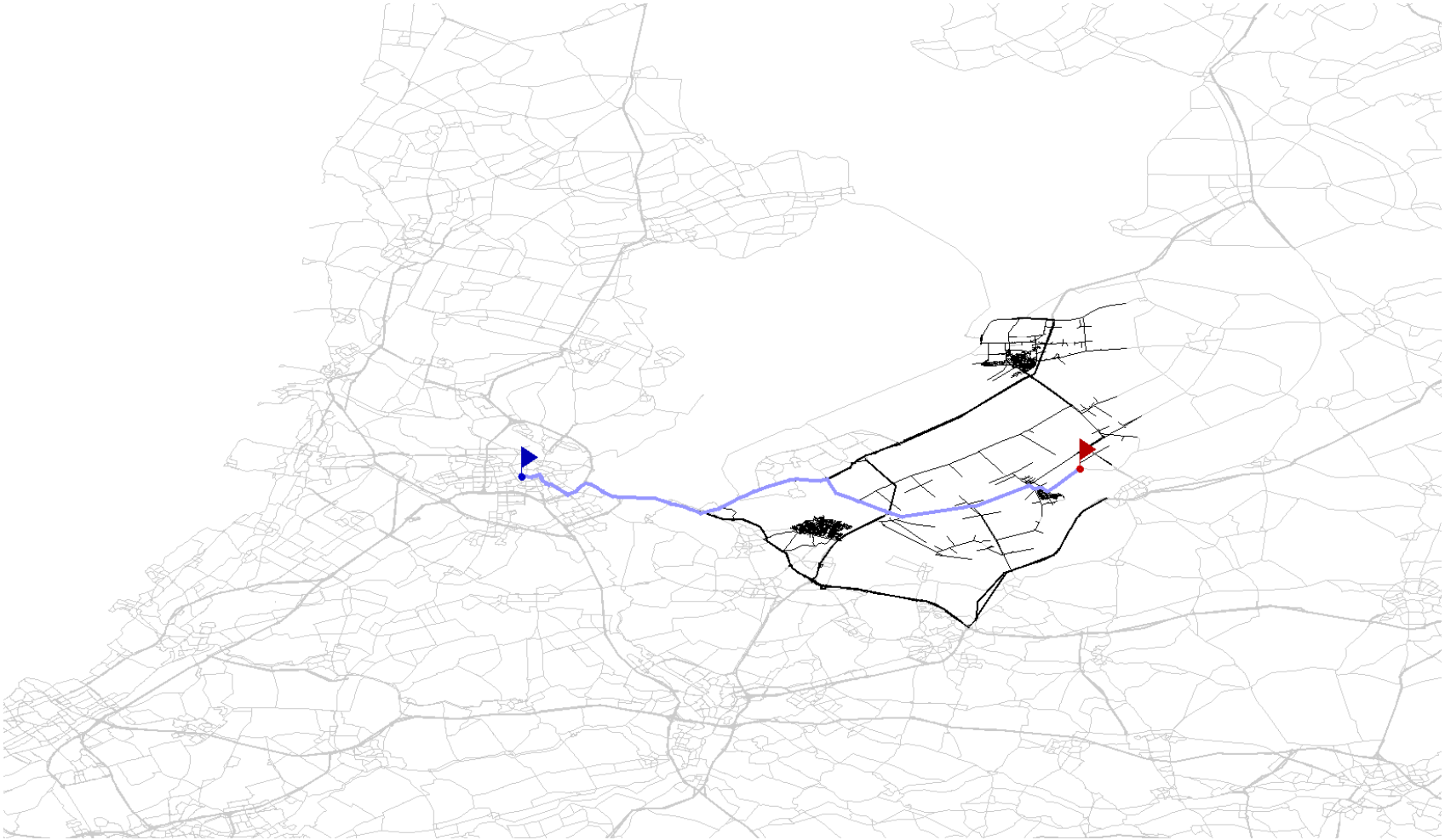
- Eindeutigkeit der Wege
- eventuell nicht korrekt



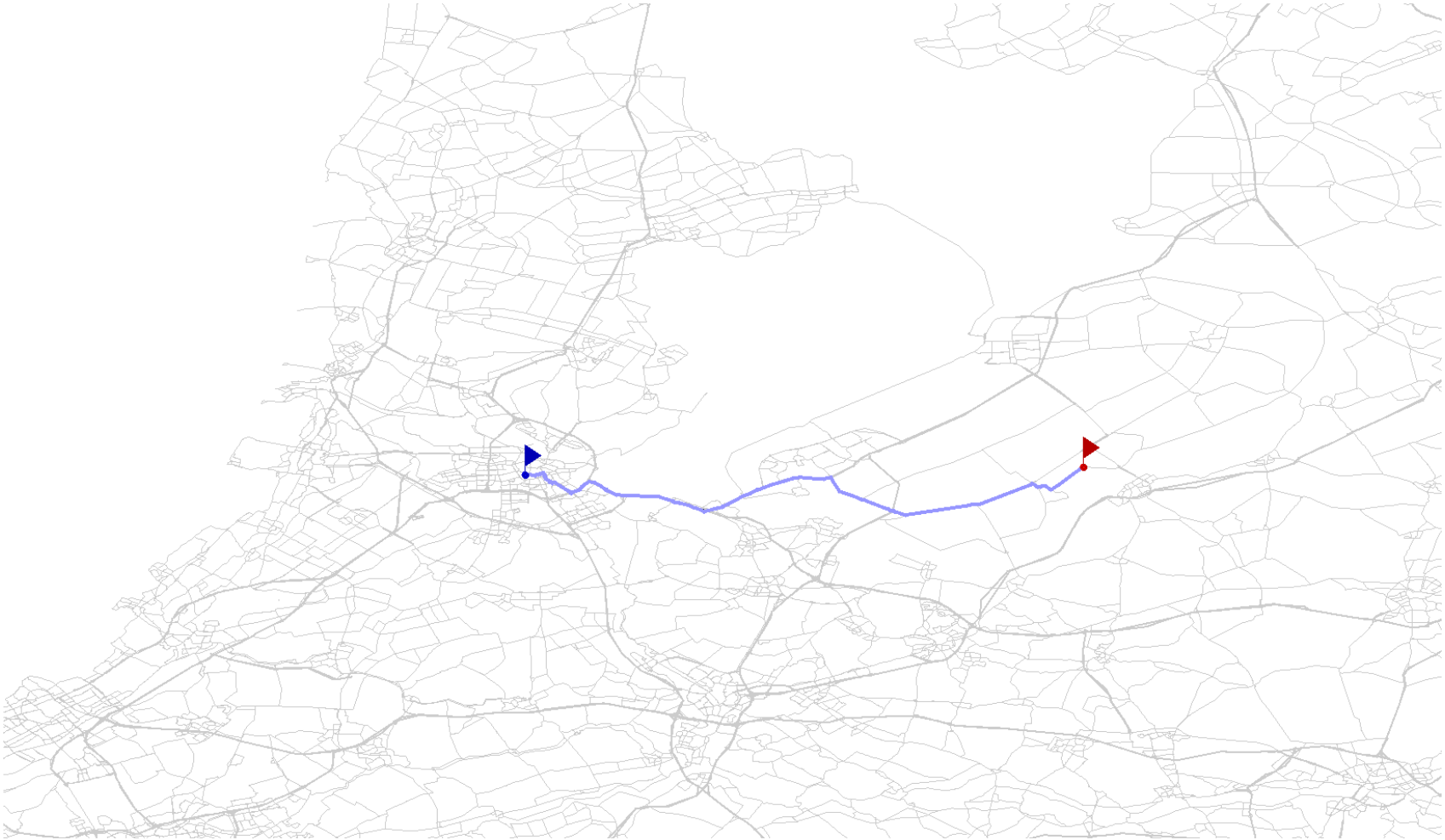
Lösung:

- kommt in Straßengraphen kaum vor
- daher öffne Flaggen für alle möglichen Wege

Suchraum



Suchraum



Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen

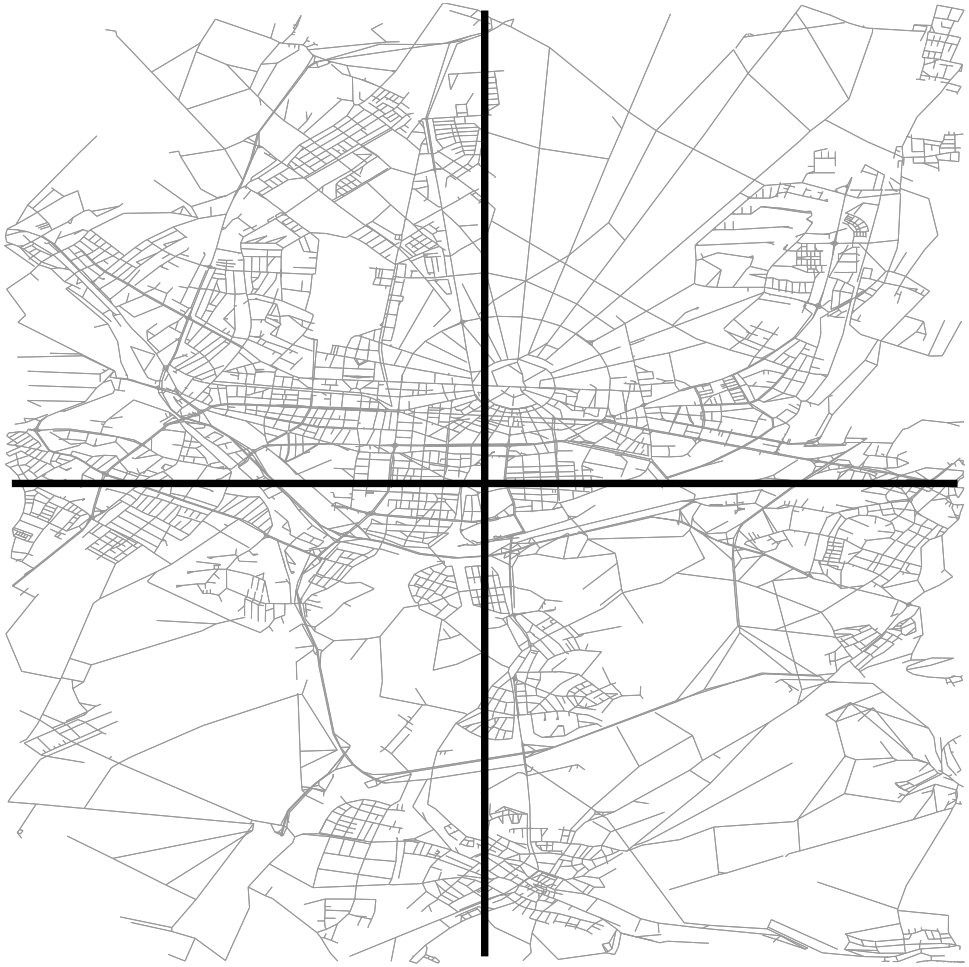
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



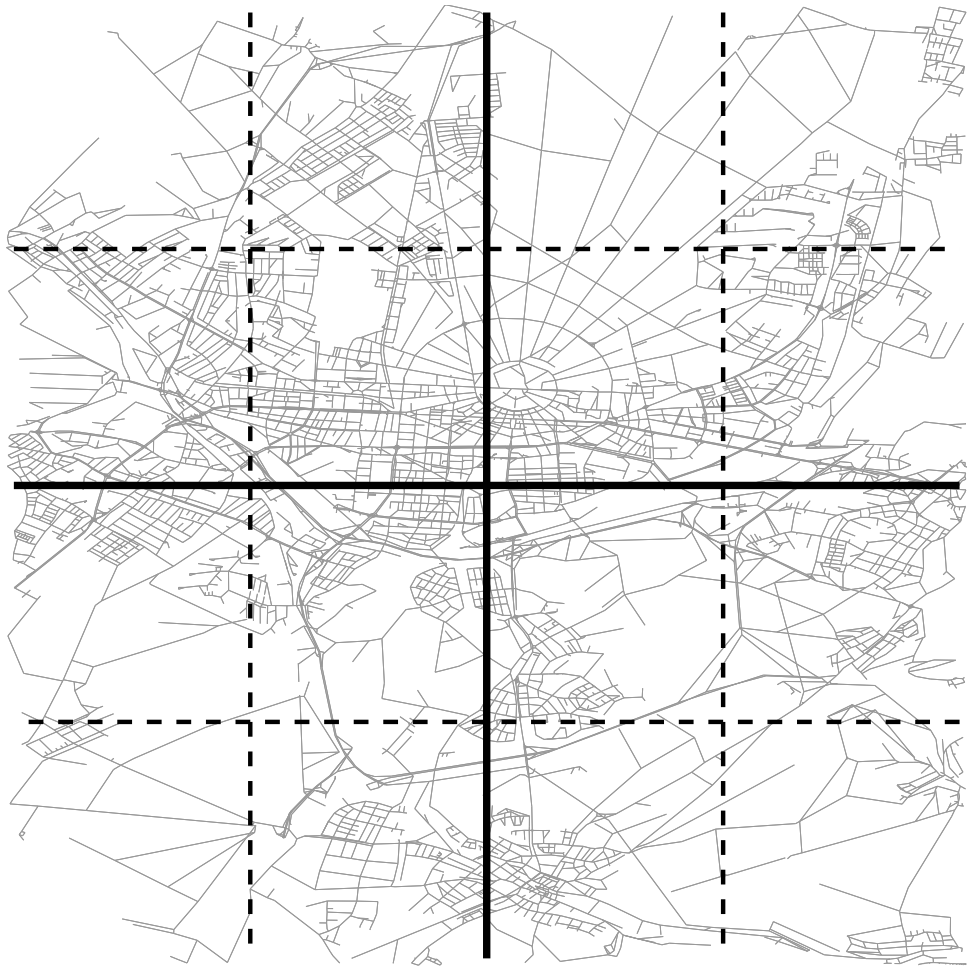
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



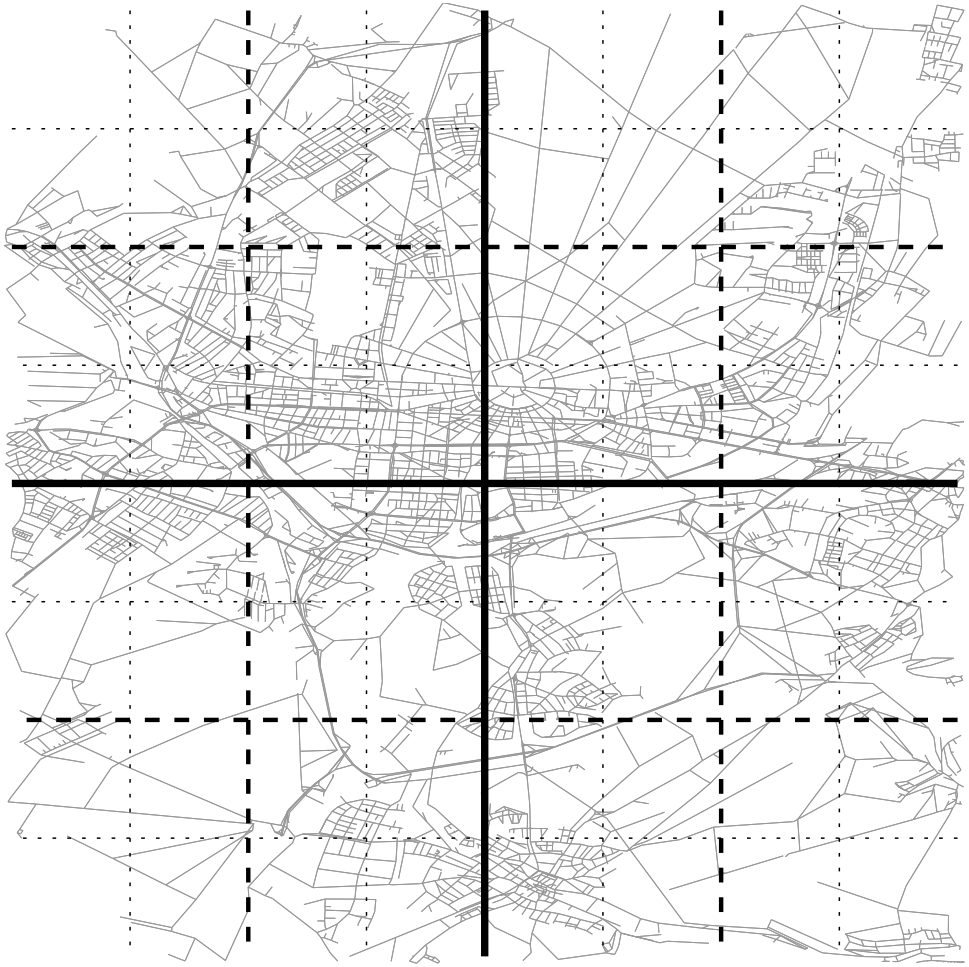
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



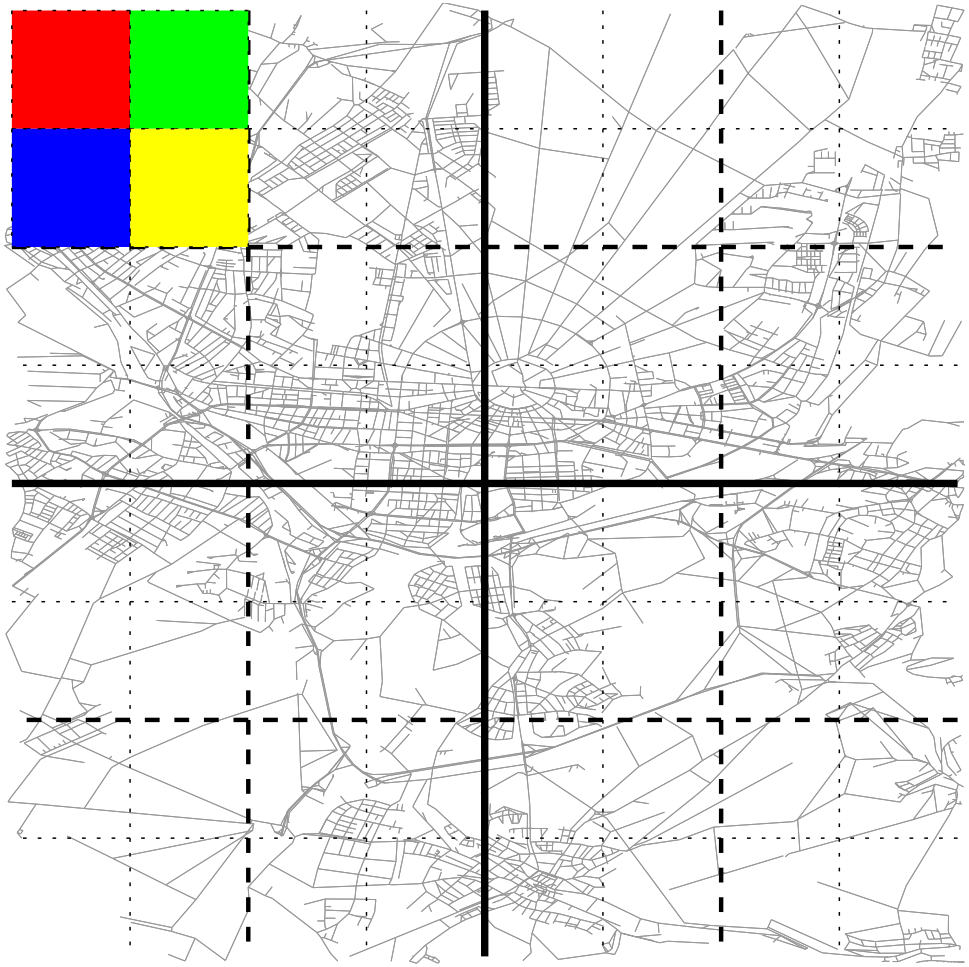
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



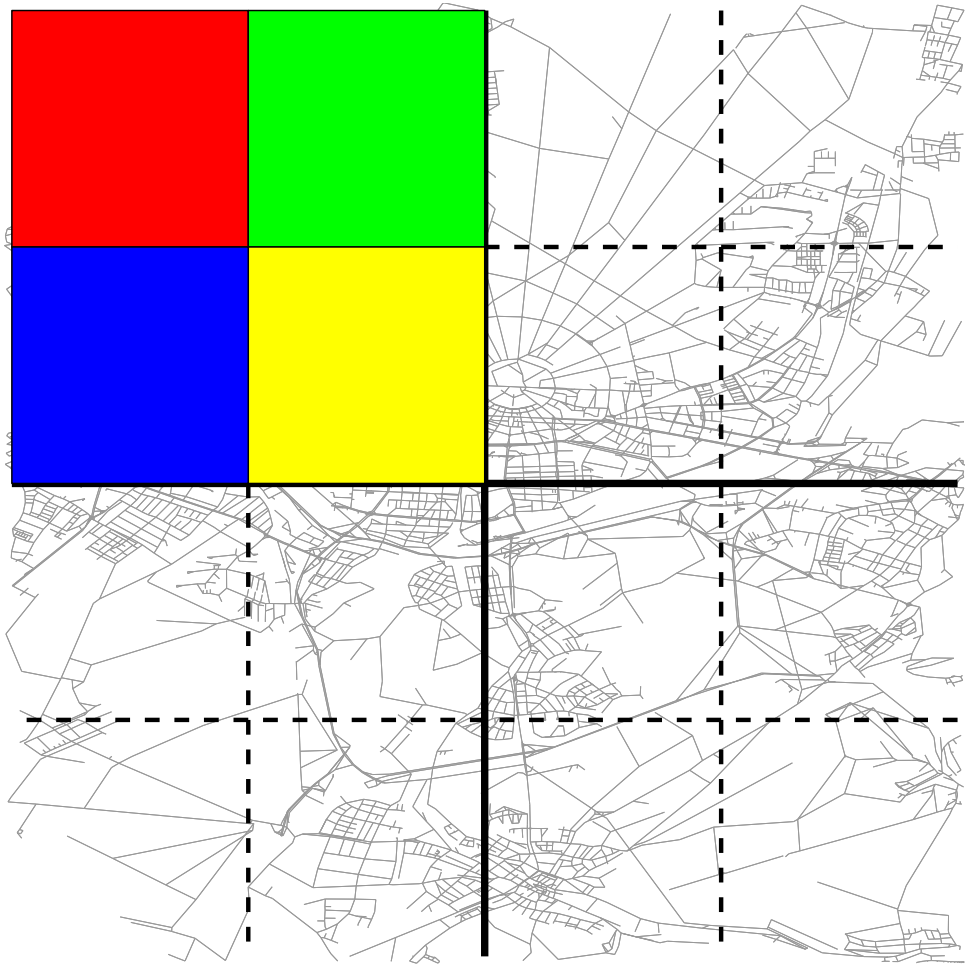
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



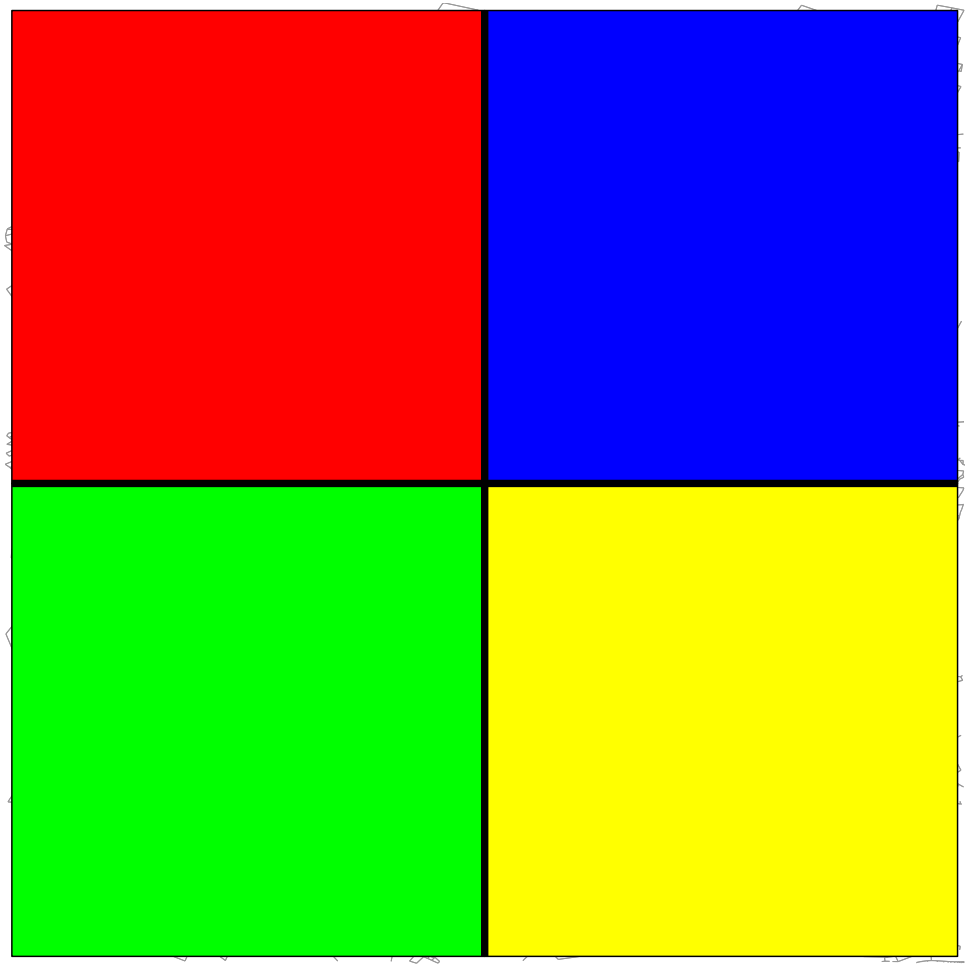
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



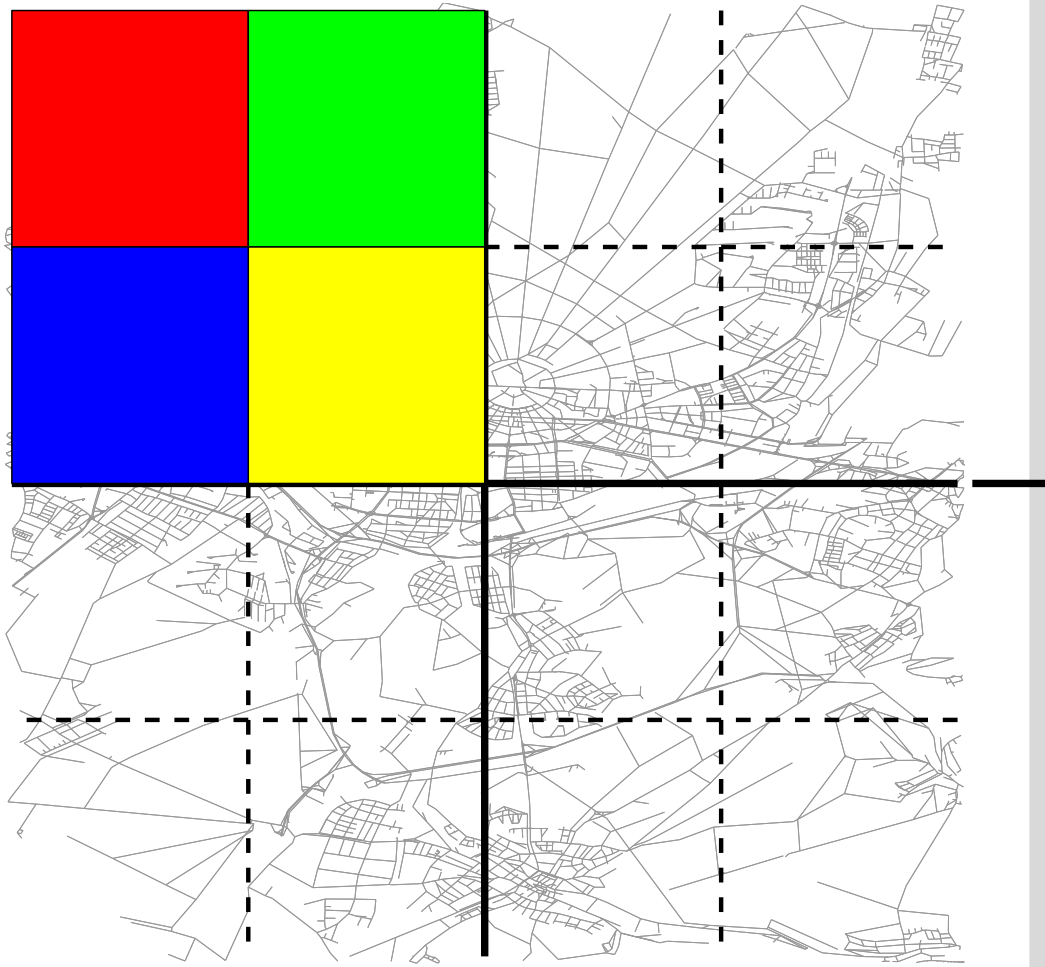
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



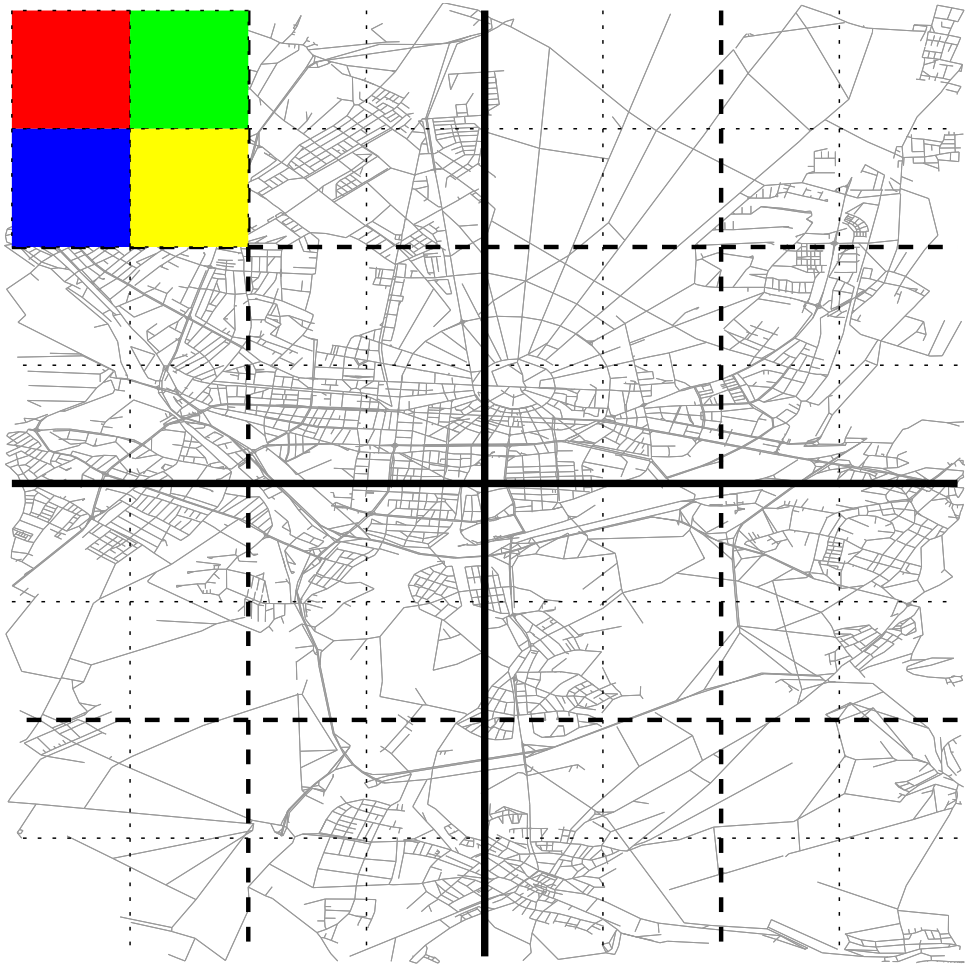
Multi-Level Arc-Flags

Problem:

- Anfragen in einer Zelle ohne Beschleunigung
- viele Real-Welt Anfragen sind lokal

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



Multi-Level Arc-Flags

Vorberechnung:

- oberster Level wie gehabt
- auf unteren Leveln:
  - von jedem Randknoten führe Dijkstra aus, bis alle Knoten der Superzelle abgearbeitet worden sind
  - setze Flaggen für alle Kanten (u, v) für die u in Superzelle
  - Hinweis: es reicht nicht, nur den Subgraphen der Superzelle zu betrachten (Übungsaufgabe)

Anfragen:

- bestimme gemeinsamen Level l von u und t
- werte Flaggen auf dem Level l aus

Korrektheit: siehe Übung

Multi-Level Arc-Flags

Vorberechnung:

- oberster Level wie gehabt
- auf unteren Leveln:
  - von jedem Randknoten führe Dijkstra aus, bis alle Knoten der Superzelle abgearbeitet worden sind
  - setze Flaggen für alle Kanten (u, v) für die u in Superzelle
  - Hinweis: es reicht nicht, nur den Subgraphen der Superzelle zu betrachten (Übungsaufgabe)

Anfragen:

- bestimme gemeinsamen Level l von u und t
- werte Flaggen auf dem Level l aus

Korrektheit: siehe Übung

Multi-Level Arc-Flags

Vorberechnung:

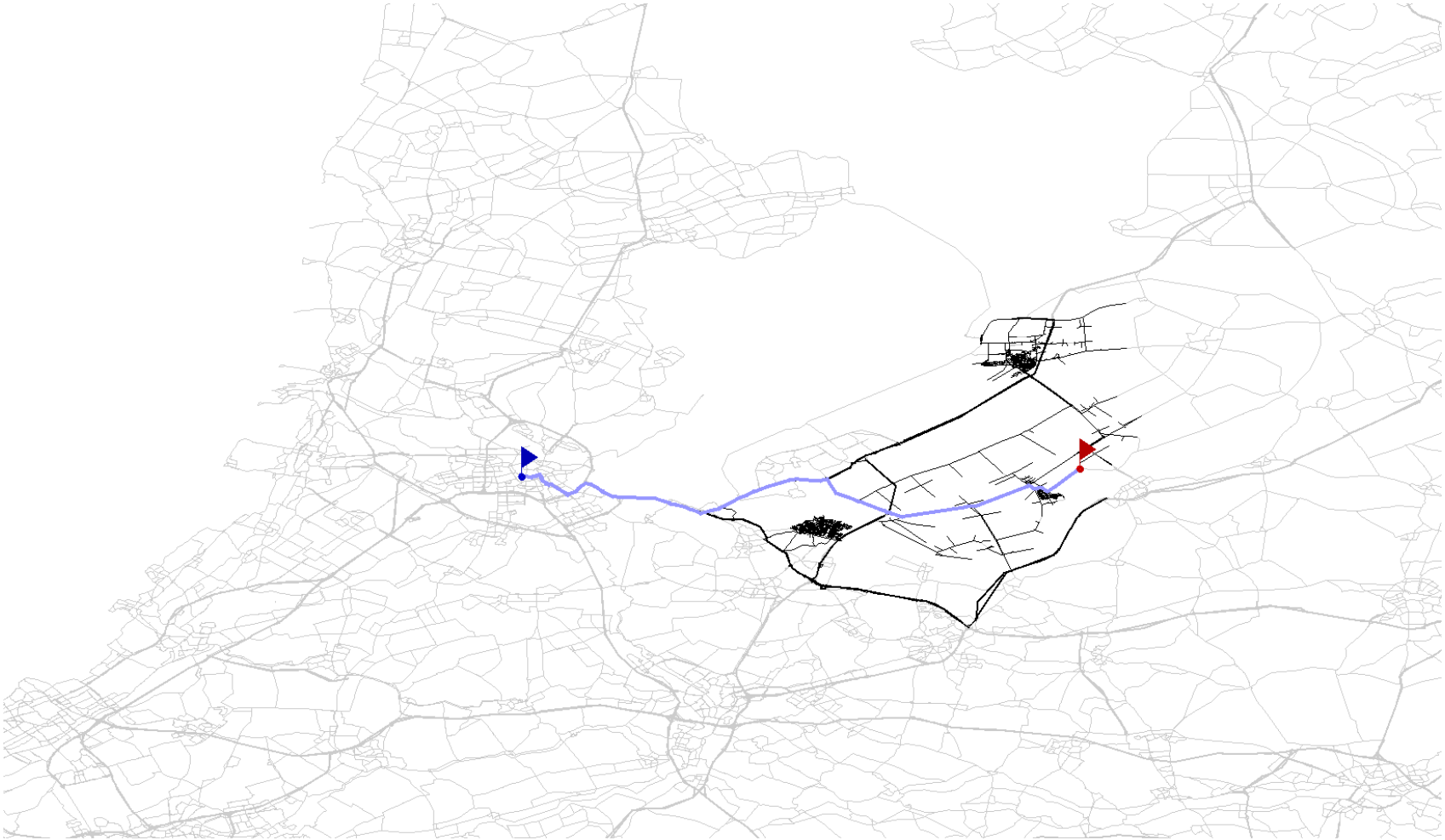
- oberster Level wie gehabt
- auf unteren Leveln:
  - von jedem Randknoten führe Dijkstra aus, bis alle Knoten der Superzelle abgearbeitet worden sind
  - setze Flaggen für alle Kanten (u, v) für die u in Superzelle
  - Hinweis: es reicht nicht, nur den Subgraphen der Superzelle zu betrachten (Übungsaufgabe)

Anfragen:

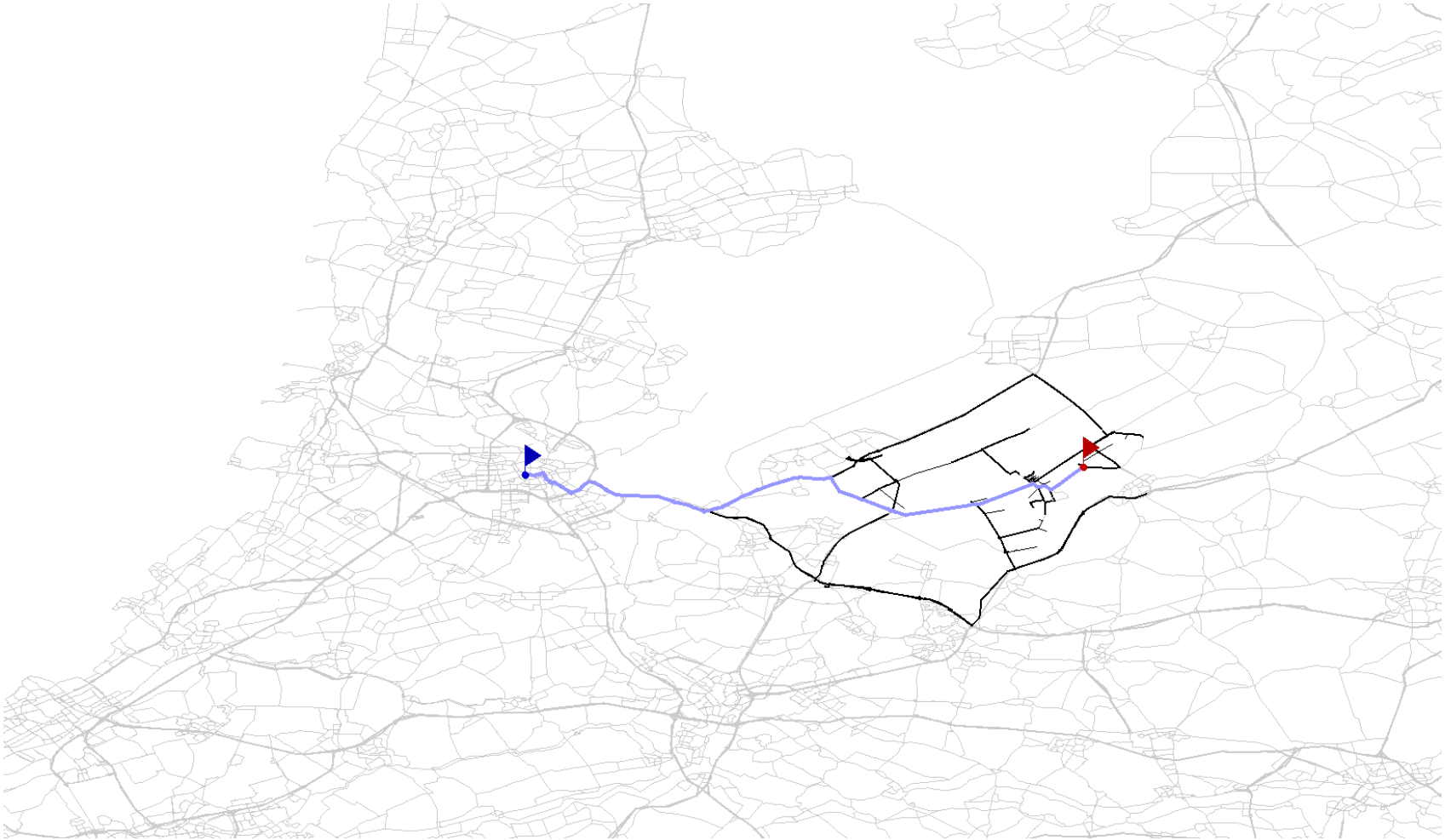
- bestimme gemeinsamen Level l von u und t
- werte Flaggen auf dem Level l aus

Korrektheit: siehe Übung

Suchraum



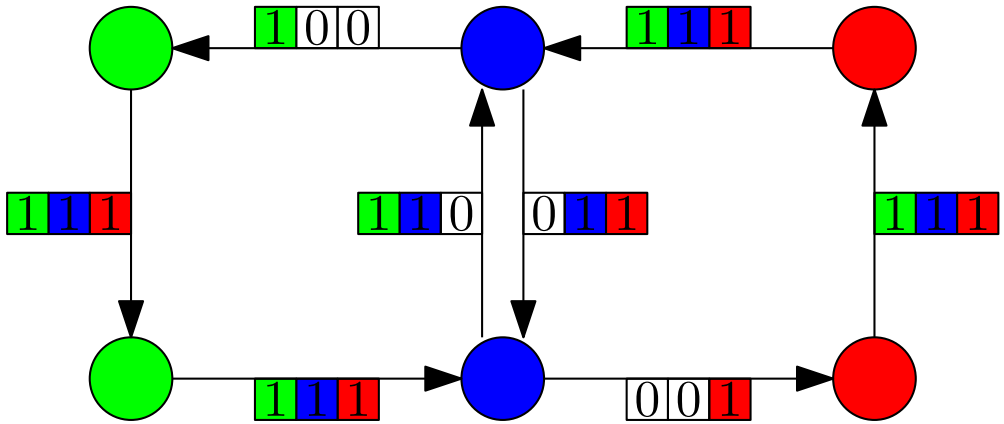
Suchraum



Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



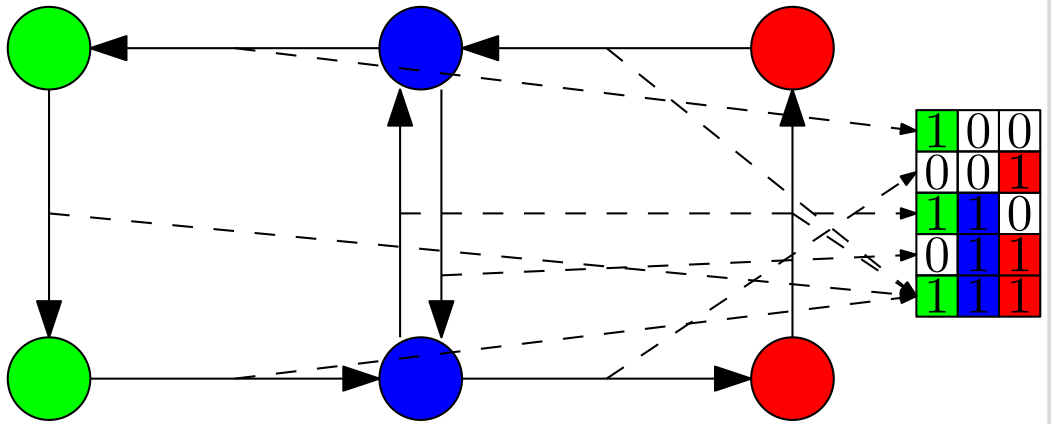
Idee:

- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



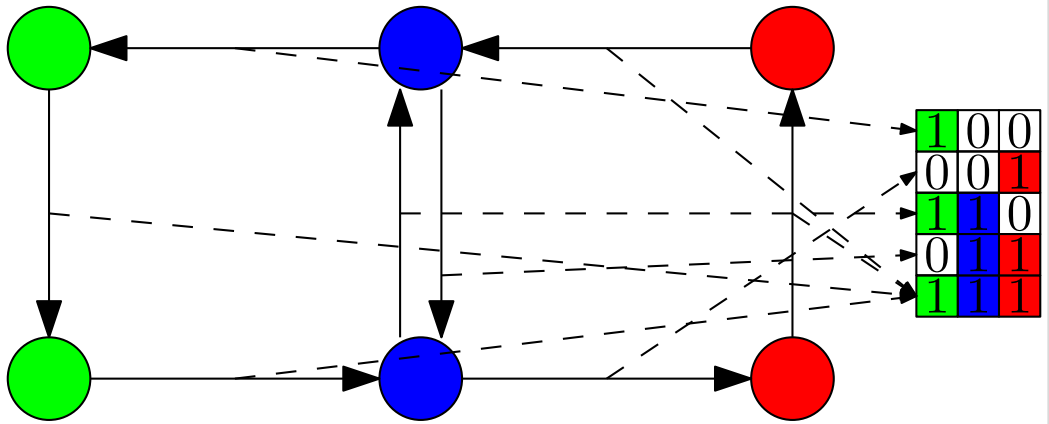
Idee:

- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Flaggenkompression

Beobachtung:

- kippen eines Bits von 1 auf 0 verboten
- kippen eines Bits von 0 auf 1 erlaubt (weiterhin korrekt, eventuell langsamer)



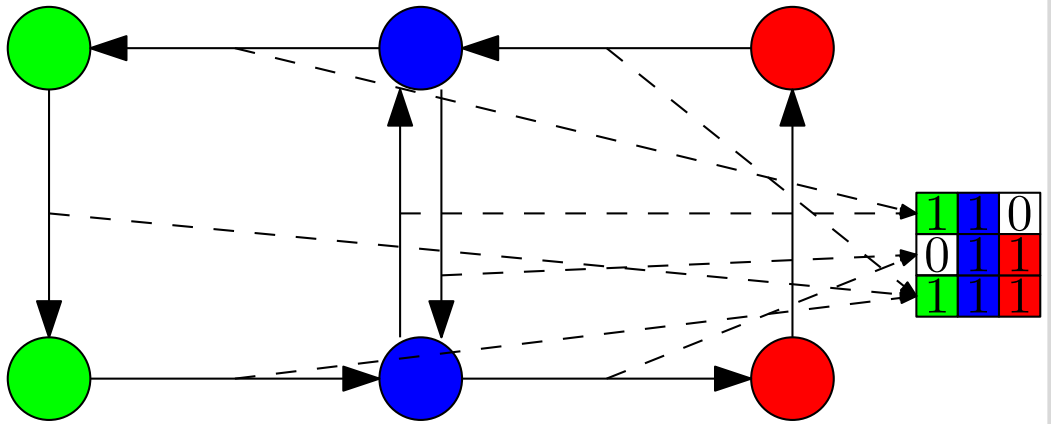
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping (Studienarbeit WS 08/09)

Flaggenkompression

Beobachtung:

- kippen eines Bits von 1 auf 0 verboten
- kippen eines Bits von 0 auf 1 erlaubt (weiterhin korrekt, eventuell langsamer)



Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping (Studienarbeit WS 08/09)

Experimente

Eingaben:

- Straßennetzwerke
  - Europa: 18 Mio. Knoten, 42 Mio. Kanten
  - USA: 22 Mio. Knoten, 56 Mio. Kanten

Evaluation:

- Vorberechnung in Minuten und zusätzliche Bytes pro Knoten
- durchschnittlicher Suchraum (#abgearbeitete Knoten) und Suchzeiten (in *ms*) von 10 000 Zufallsanfragen

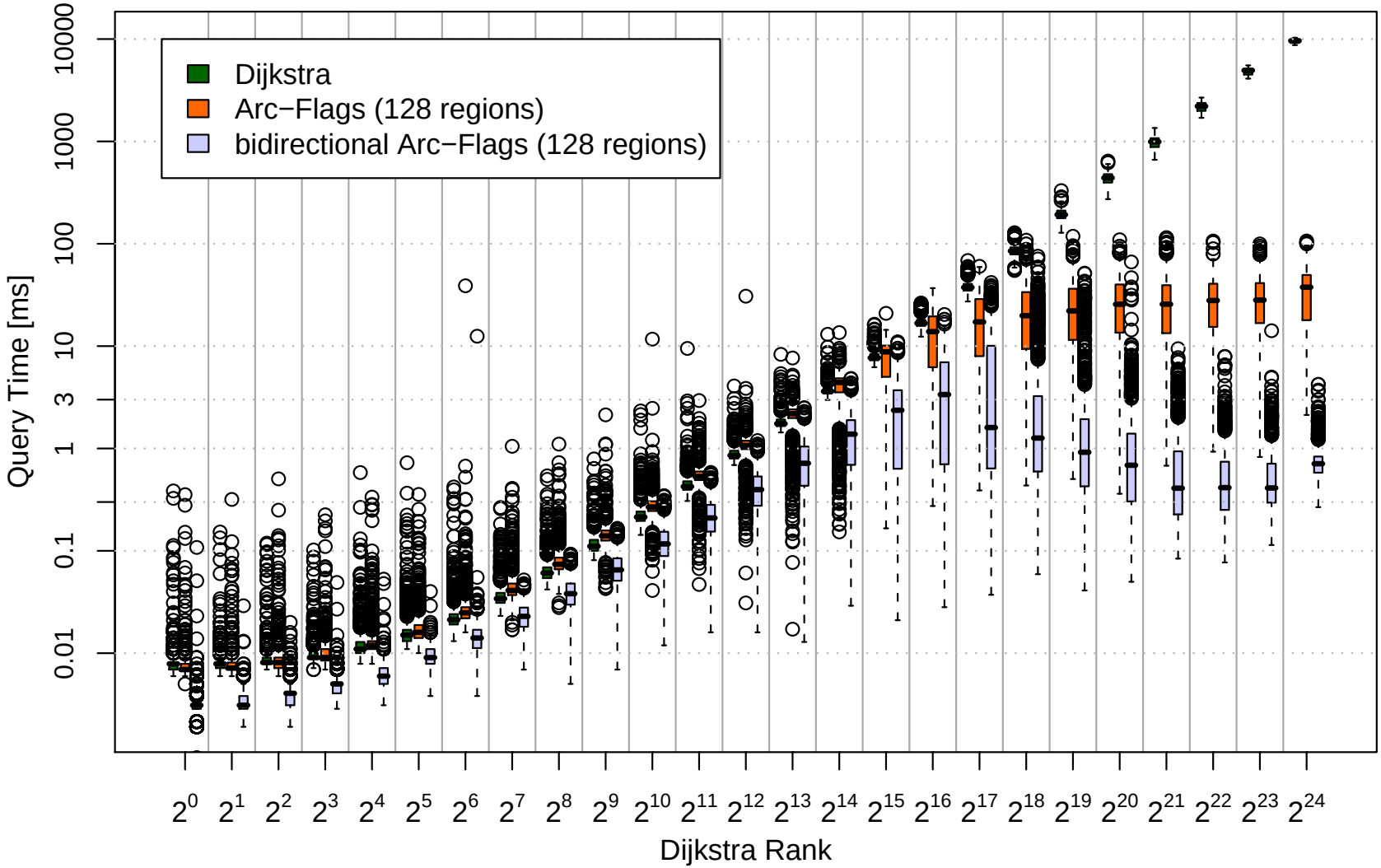
Zufallsanfragen: Anzahl Regionen

	Prepro		Query		
regions	time [min]	space [B/n]	# settled nodes	time [ms]	spd up
0	0	0	9 114 385	5 591.6	1.0
200	1 028	19	2 369	1.6	3 494.8
400	1 366	20	1 868	1.2	4 659.7
600	1 723	21	1 700	1.1	5 083.3
800	1 892	23	1 642	1.4	3 994.0
1000	2 156	25	1 593	1.1	5 083.3

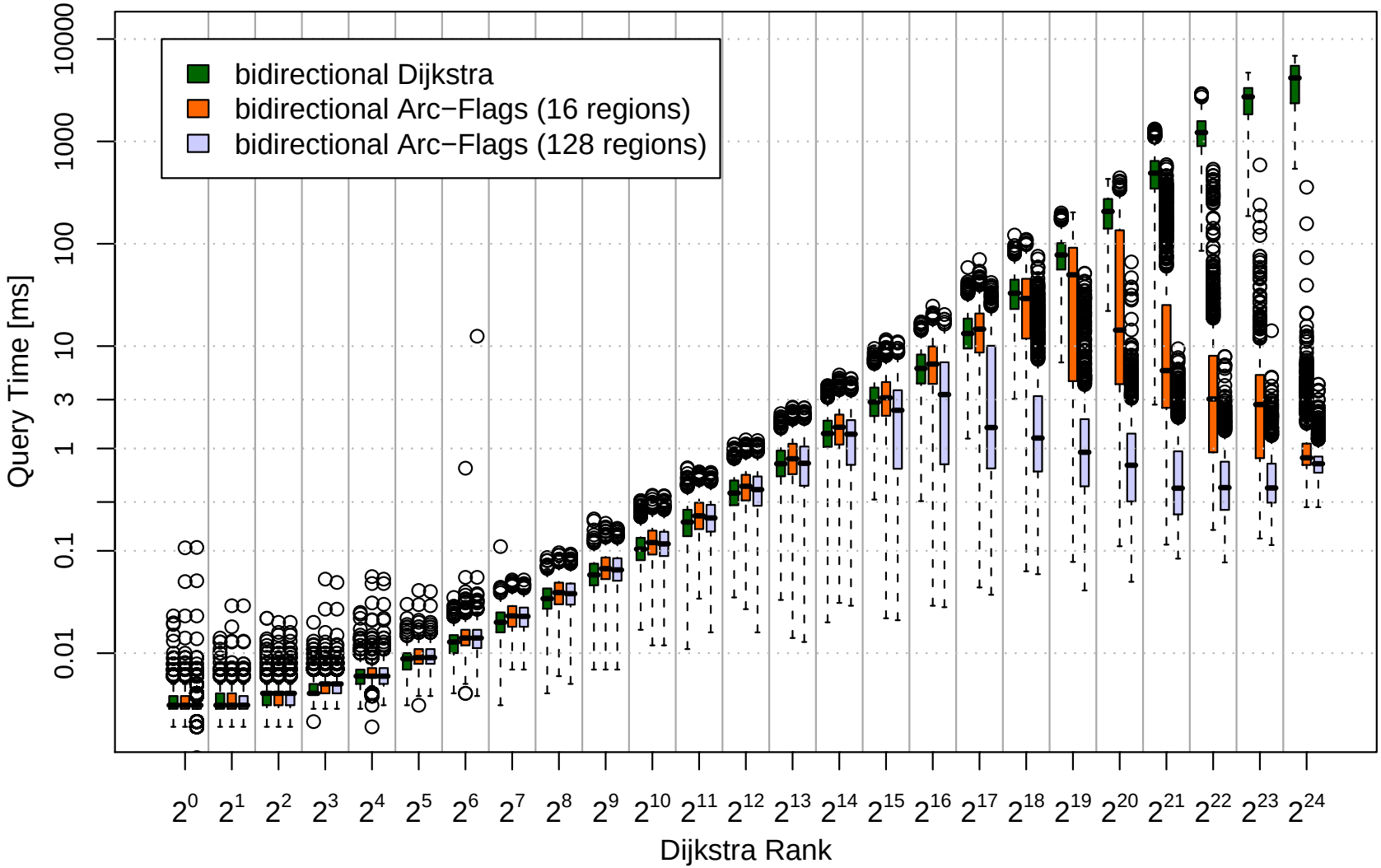
Beobachtungen:

- lange Vorberechnung
- hohe Beschleunigung
- geringer Speicherverbrauch
- mehr als 200 Regionen lohnt sich nicht

Lokale Anfragen Arc-Flags I



Lokale Anfragen Arc-Flags I

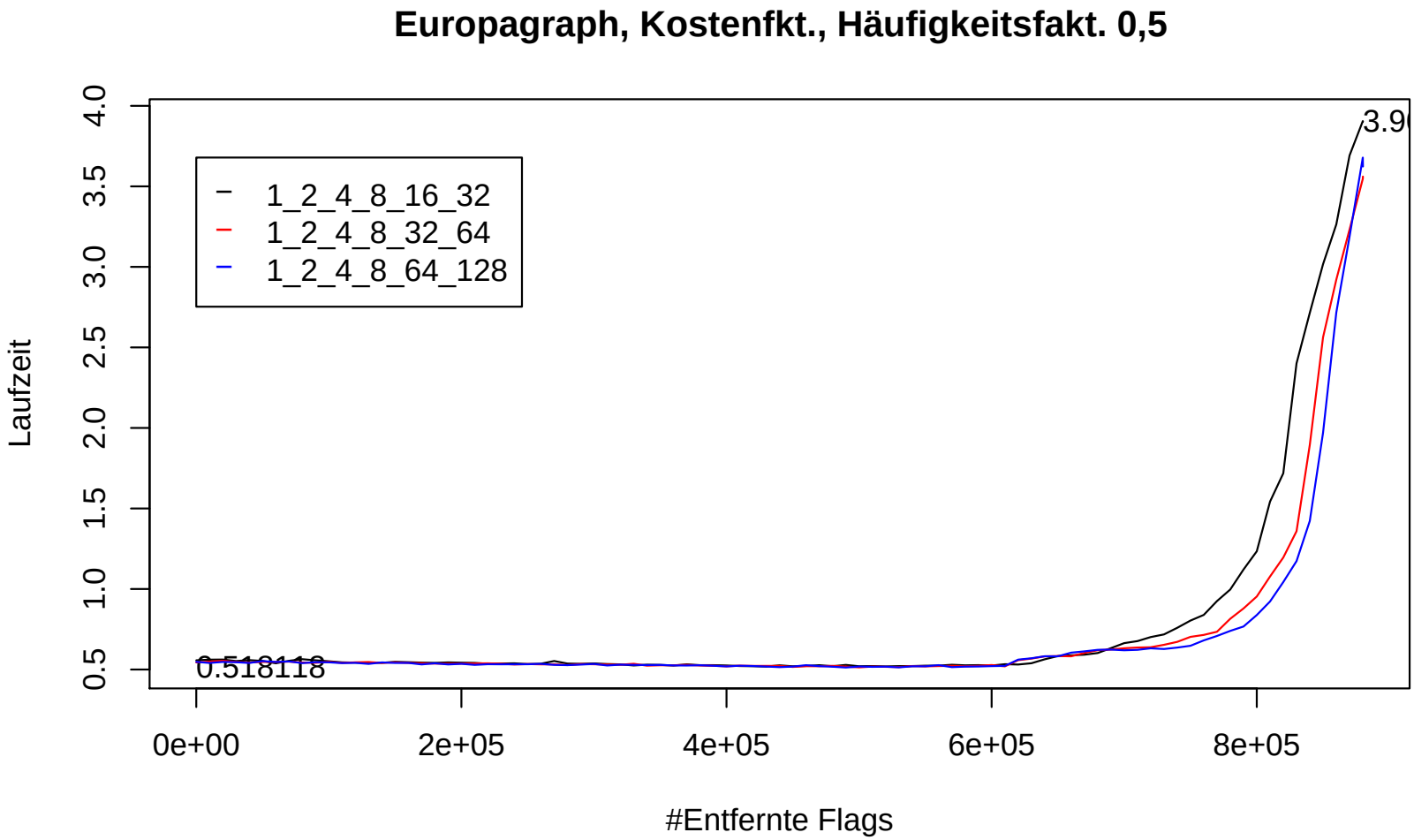


Beobachtungen

- birektionale Arc-Flags deutlich schneller als unidirektionale
- gegenüber Dijkstra nur Beschleunigung für weite Anfragen
- 128 Regionen deutlich besser 16 (bei 2^{24} nahezu gleich auf)

Flaggenkompression

(Multi-Level, unidirektional)



Beobachtungen

- kaum Verlust bis zu 60% entfernte Flaggen
- geringer Verlust bis zu 80% entfernte Flaggen
- kippen von niedrig-leveligen Flaggen billiger

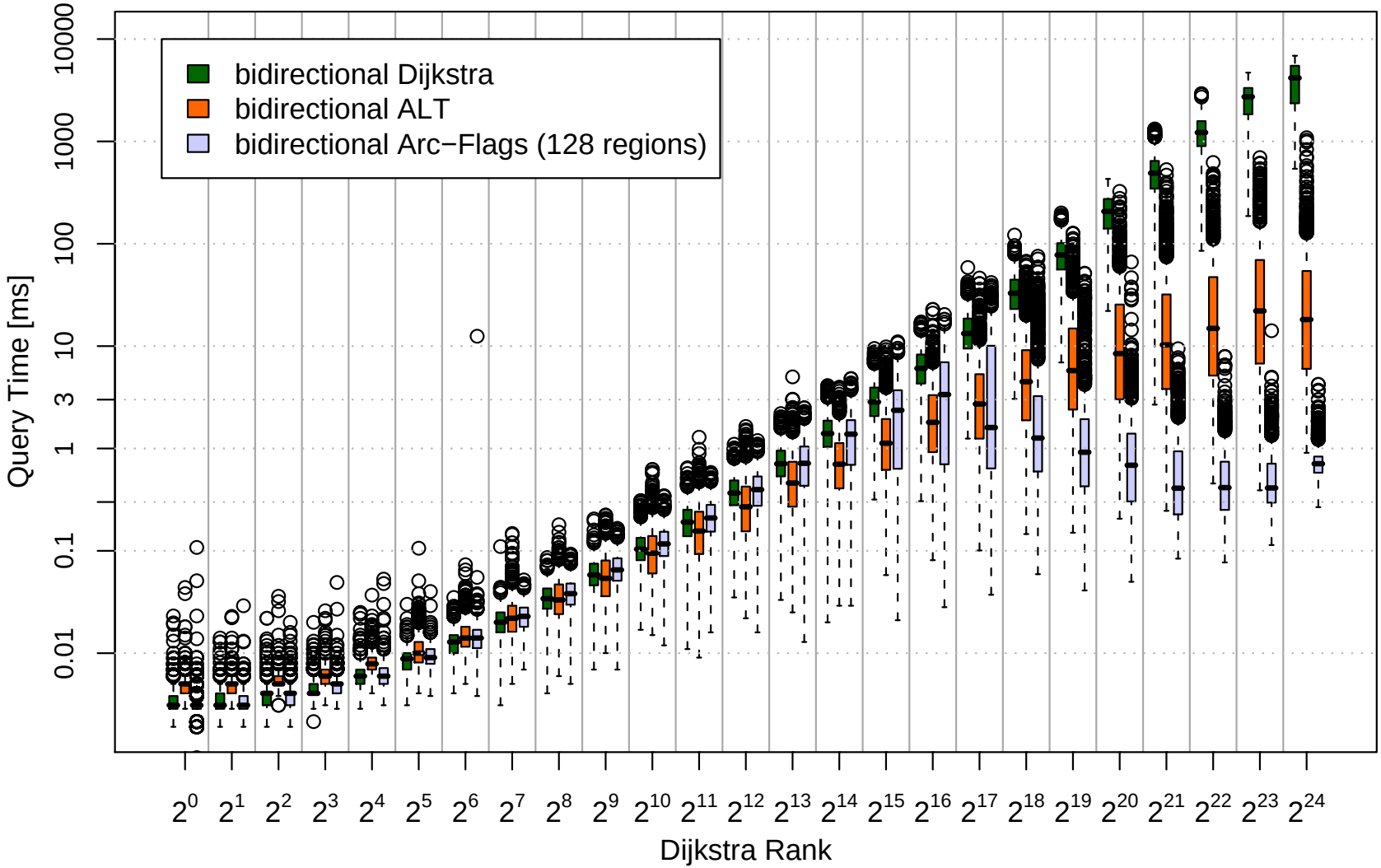
Übersicht: bisherige Techniken

	Vorberechnung		Anfrage		
	Zeit [h:m]	Platz [byte/n]	Such raum	Zeit [ms]	Beschl.
Dijkstra	0:00	0	9 114 385	5 591.6	1.0
Bi-Dijkstra	0:00	0	4 764 110	2 713.2	2.1
Uni-ALT-16	1:25	128	815 639	327.6	17.1
Uni-ALT-64	1:08	512	604 968	288.5	19.4
Bi-ALT-16	1:25	128	74 669	53.6	104.3
Bi-ALT-64	1:08	512	25 324	19.6	285.3
Uni Arc-Flags (128)	8:34	20	92 545	31.9	175.3
Bi Arc-Flags (128)	17:08	10	2 764	0.8	6 988.1

Beobachtung:

- ALT: deutlich unterlegen bei Anfragen und Platzverbrauch
- Arc-Flags: deutlich längere Vorberechnungszeiten

Lokale Anfragen Vergleich



Zusammenfassung

Geometrische Container

- speichere pro Kante einen Container ab
- beinhaltet alle Knoten, die über diese Kante erreicht werden können
- einfacher Anfrage-Algorithmus
- Vorberechung basiert auf Dijkstra-Bäumen
- Beschleunigung von ca. 40
- Vorberechnung basiert auf APSP
- knapp 500 Jahre Vorberechnung für Europa

nicht praktikabel

Zusammenfassung

Arc-Flags

- Teile Graphen in k Regionen
- Flaggen zeigen an, ob Kante wichtig für Zielregion ist
- Einfacher Anfrage-Algorithmus
- Vorberechnung
  - Kürzeste-Wege-Baum von jedem Randknoten
  - Manche Flaggen können automatisch gesetzt werden
- Bidirektional und multi-level Erweiterungen
- Beschleunigung von bis zu 7000
- Nahe Anfragen nicht schneller als Dijkstra
- Vorberechnung dauert allerdings immer noch ca. 1 Tag
- Seit kurzem extrem schnelle Vorberechung für Straßennetzwerke

Ende

Literatur (Geometrische Container, Arc-Flags):

- Dorothea Wagner and Thomas Willhalm and Christos Zaroliagis:
  Geometric Containers for Efficient Shortest-Path Computation
  In: *ACM Journal of Experimental Algorithmics, 2005 article 1.3.*
- Moritz Hilger and Ekkehard Köhler and Rolf H. Möhring and Heiko Schilling:
  Fast Point-to-Point Shortest Path Computations with Arc-Flags
  In: *Shortest Paths: Ninth DIMACS Implementation Challenge, 2009*