Karlsruher Institut für Technologie

Algorithmen für Routenplanung

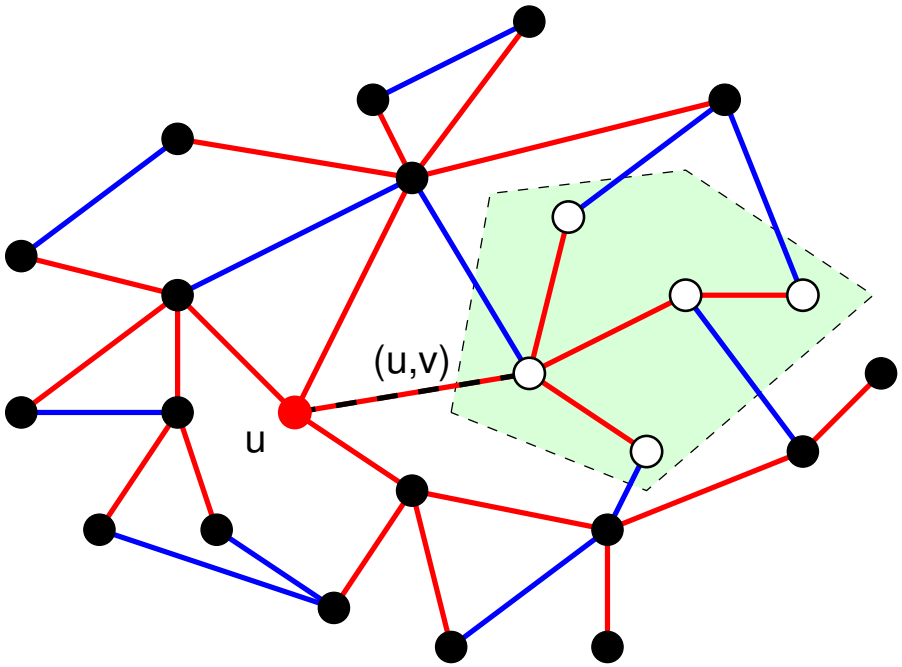
4. Sitzung, Sommersemester 2010

Thomas Pajor | 3. Mai 2010

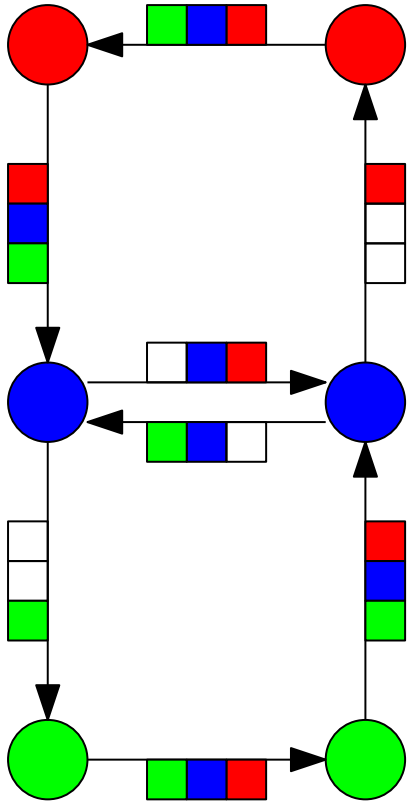
Institut für Theoretische Informatik · Algorithmik I · Prof. Dr. Dorothea Wagner

Letztes Mal

Geometrische Container

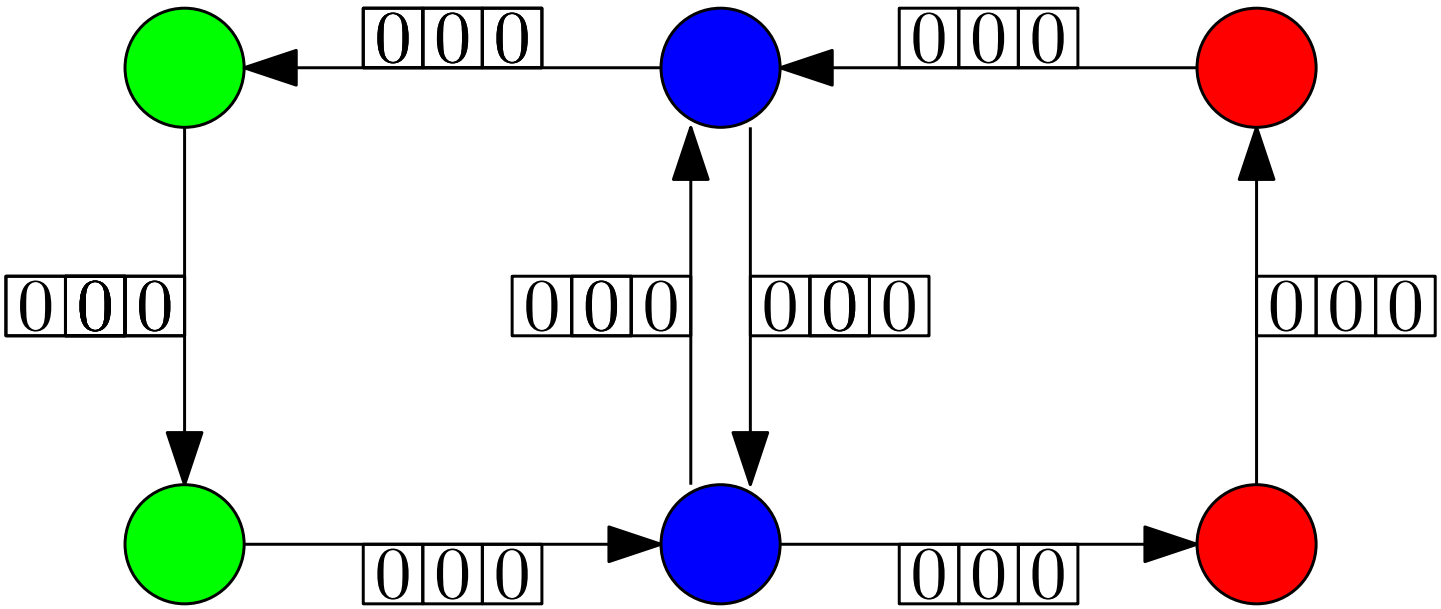


Arc-Flags



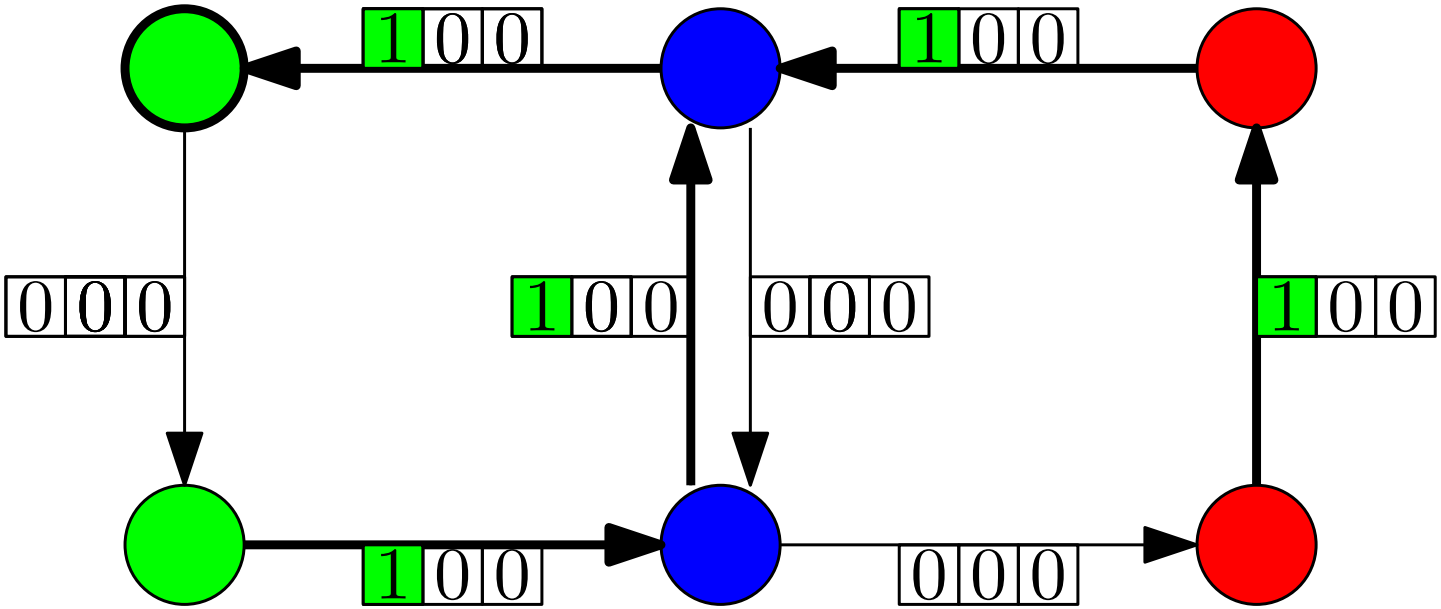
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



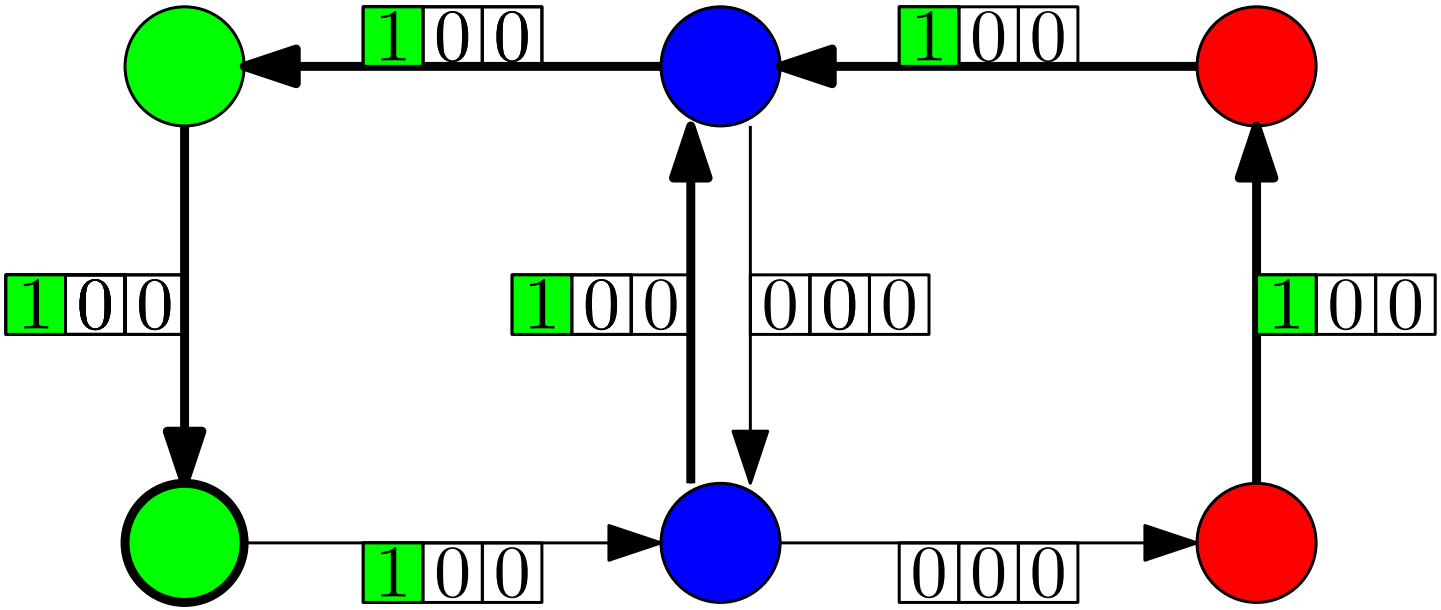
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



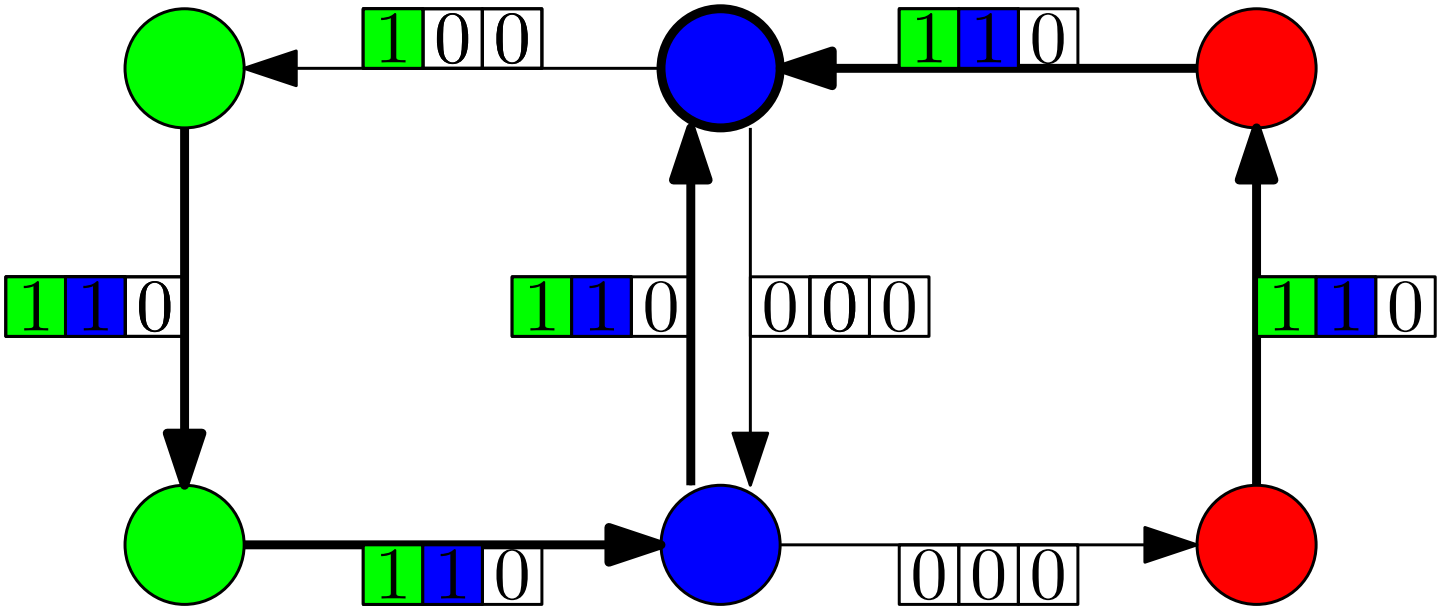
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



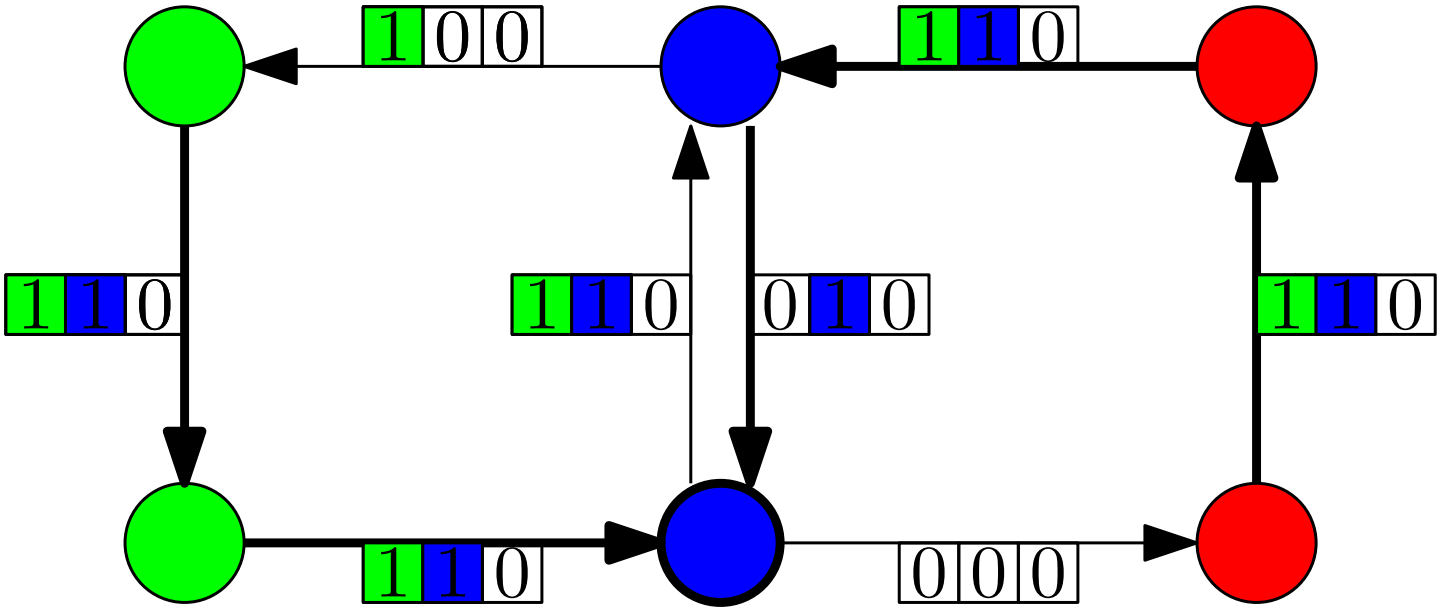
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



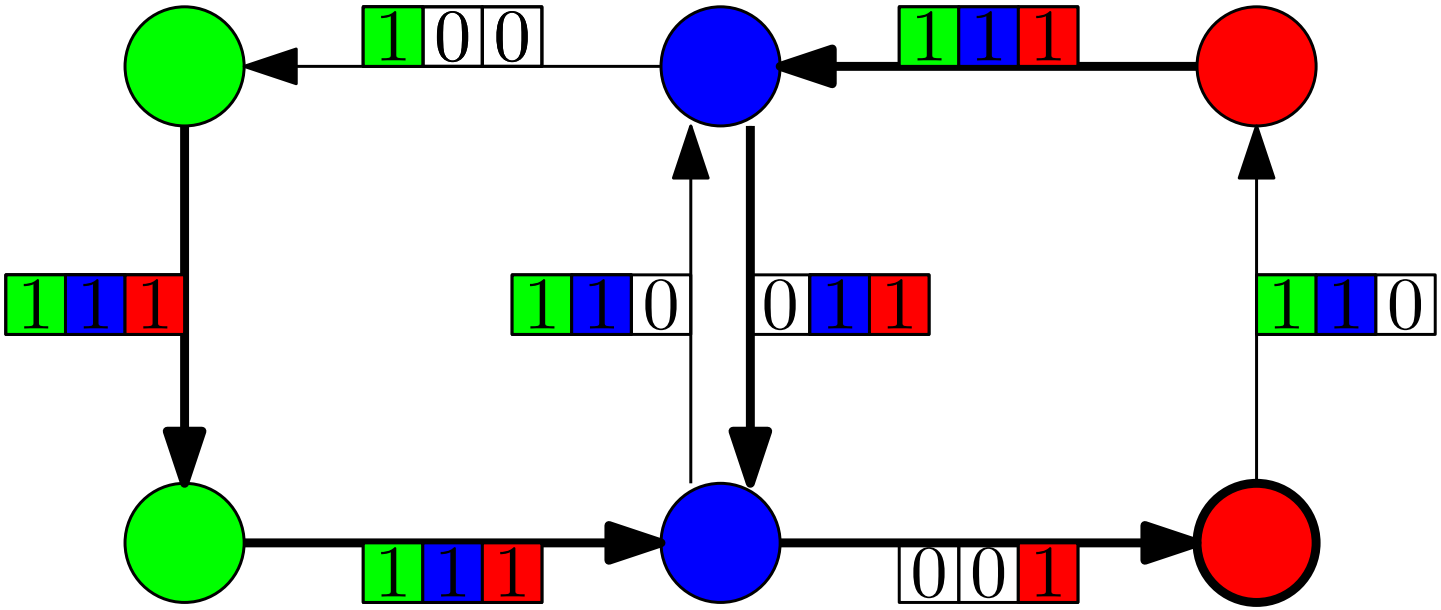
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



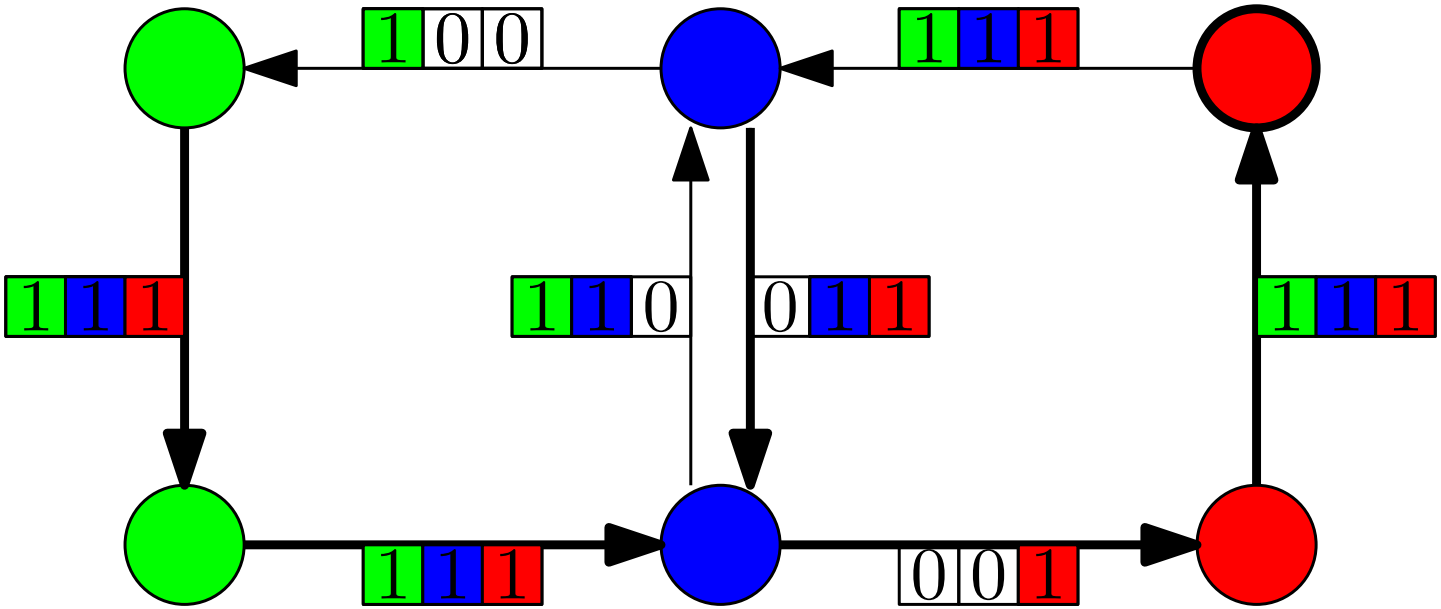
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



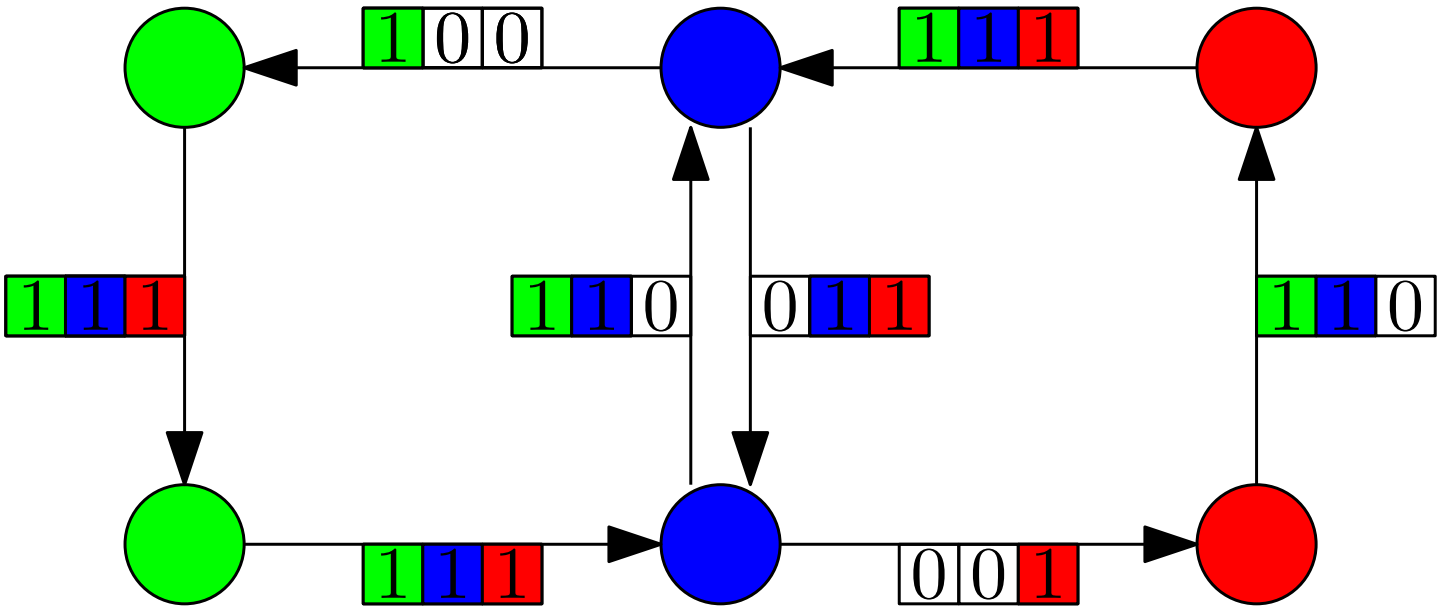
Flaggenberechnung: Erster Versuch

- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



Flaggenberechnung: Erster Versuch

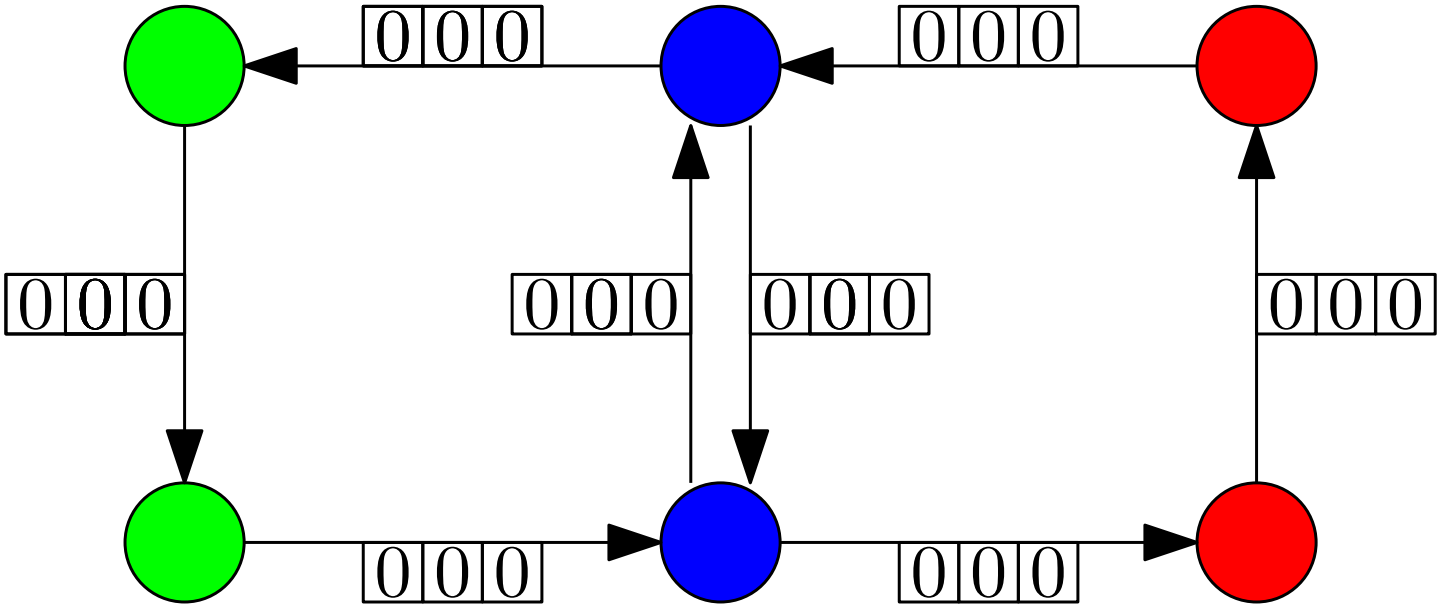
- von jedem Knoten
- konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- wieder APSP und somit $>$ 400 Jahre



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

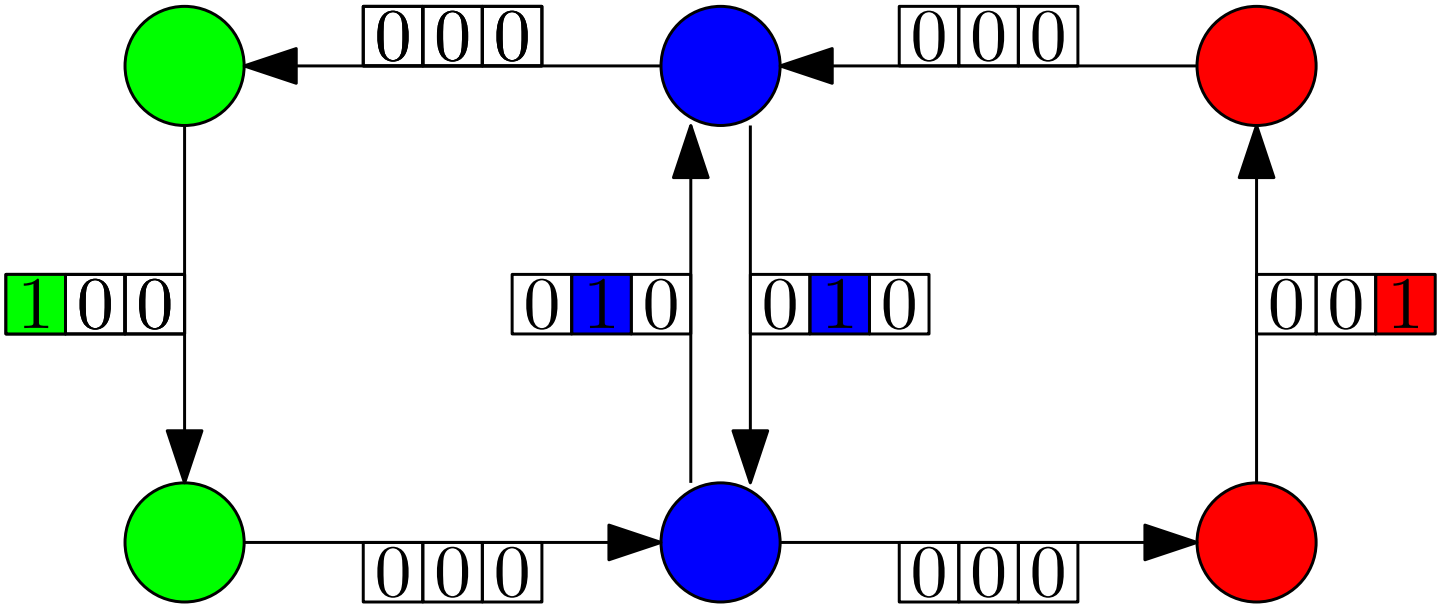
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten b
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn e Baumkante des Baums von b ist



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

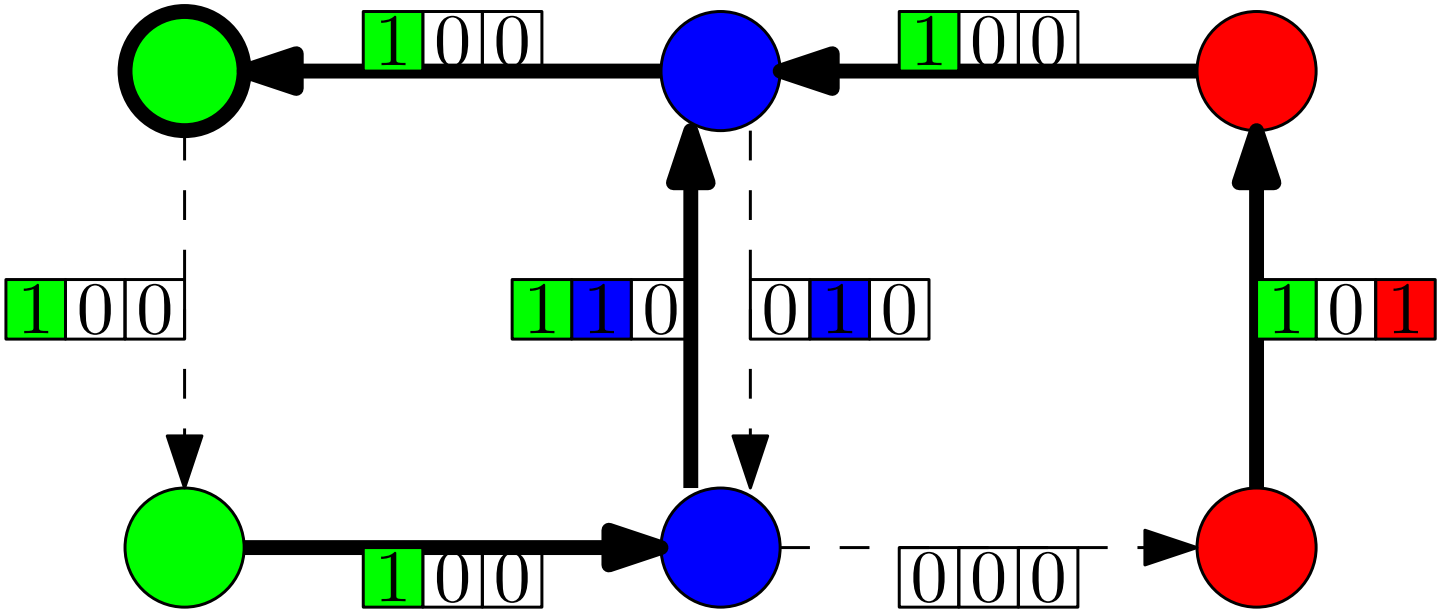
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten b
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn e Baumkante des Baums von b ist



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

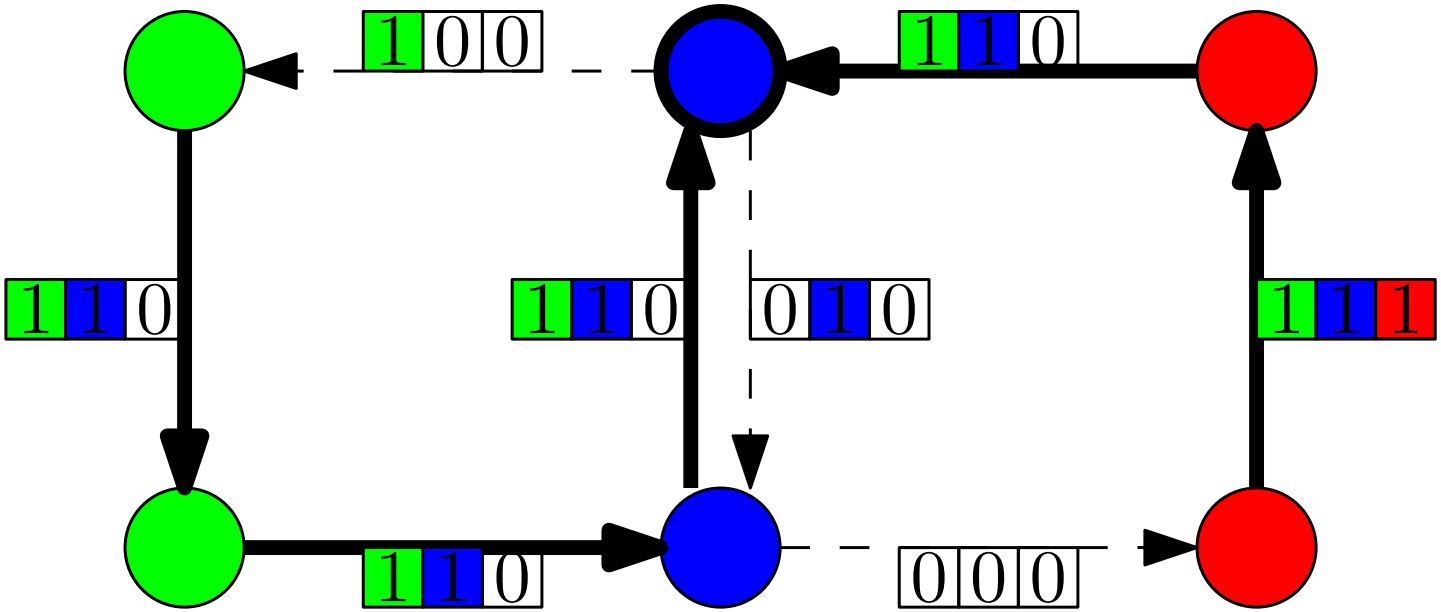
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten *b*
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn *e* Baumkante des Baums von *b* ist



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

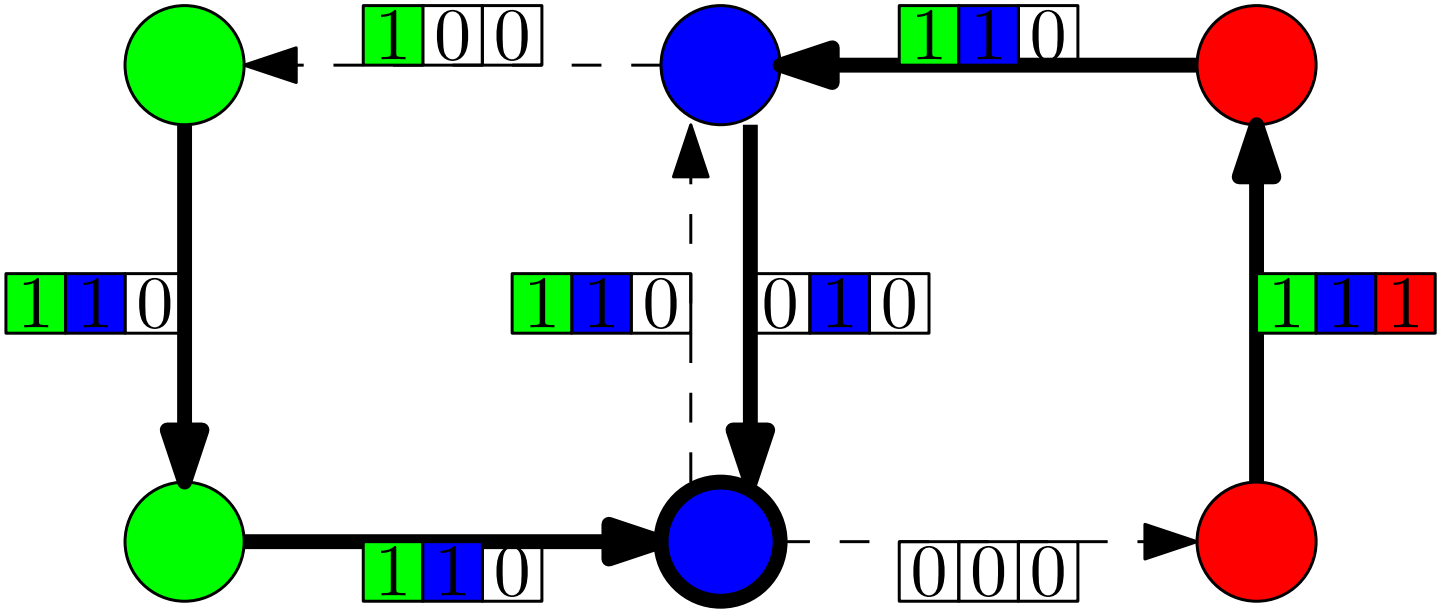
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten *b*
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn *e* Baumkante des Baums von *b* ist



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

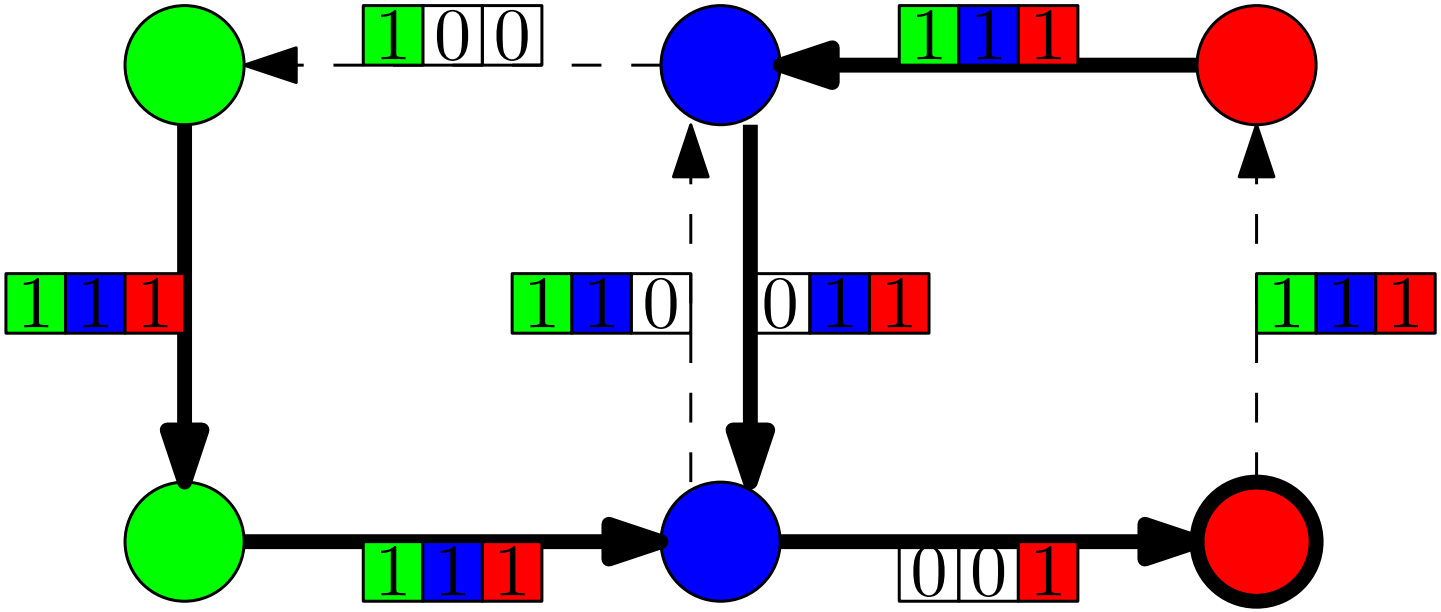
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten *b*
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn *e* Baumkante des Baums von *b* ist



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

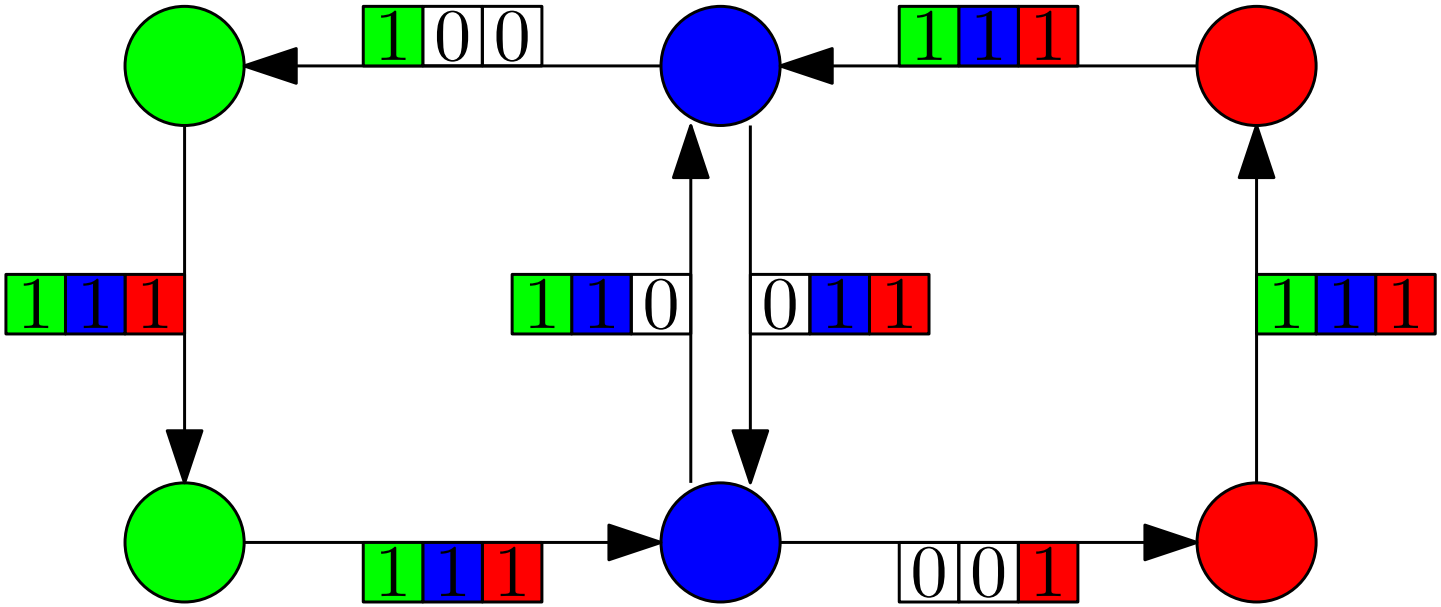
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten b
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn e Baumkante des Baums von b ist



Flaggenberechnung: Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten *b*
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn *e* Baumkante des Baums von *b* ist



Zentralisierte Bäume

verallgemeinere Dijkstra:

- für jede Region r (mit Randknotenmenge B_r)
- hänge Label $l(u)$ der Größe $|B_r|$ an jeden Knoten u
- speichert den Abstand $d(u, b)$ für alle $b \in B_r$
- triviale Intialisierung der Randknoten, füge alle in Queue ein
- nehme Knoten u aus der Queue (key: $\min\{l(u\})$
- relaxiere Kanten (u, v):
  - erzeuge Label l durch $l(u) + \text{len}(u, v)$
  - checke ob l das Label $l(v)$ verbessert
- breche ab, wenn keine Verbesserungen mehr möglich sind
- setze Flagge auf `true` wenn $d(u, b) + \text{len}(u, v) = d(v, b)$ gilt

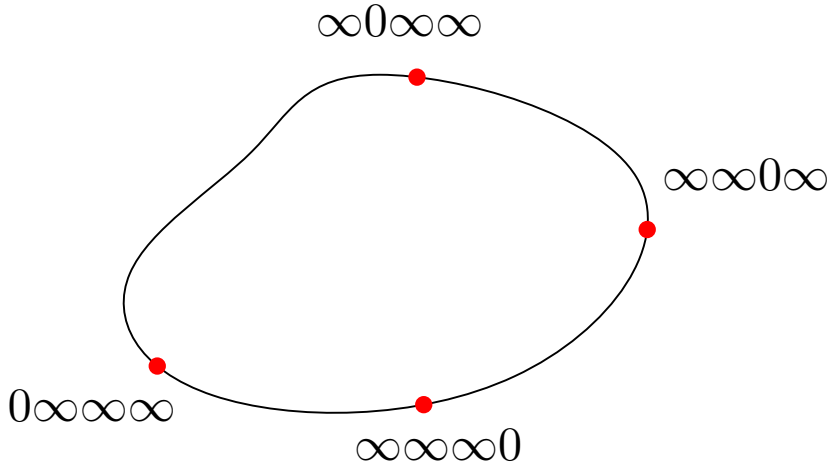
Zentralisierte Bäume: Initialisierung

trivial:

- jeder Randknoten b mit Abstand 0 zu sich und ∞ zu allen anderen

besser:

- von jedem b lokale Suche zu allen anderen Randknoten
- alle Knoten in die Queue, die auf Rand der Schnittmenge liegen



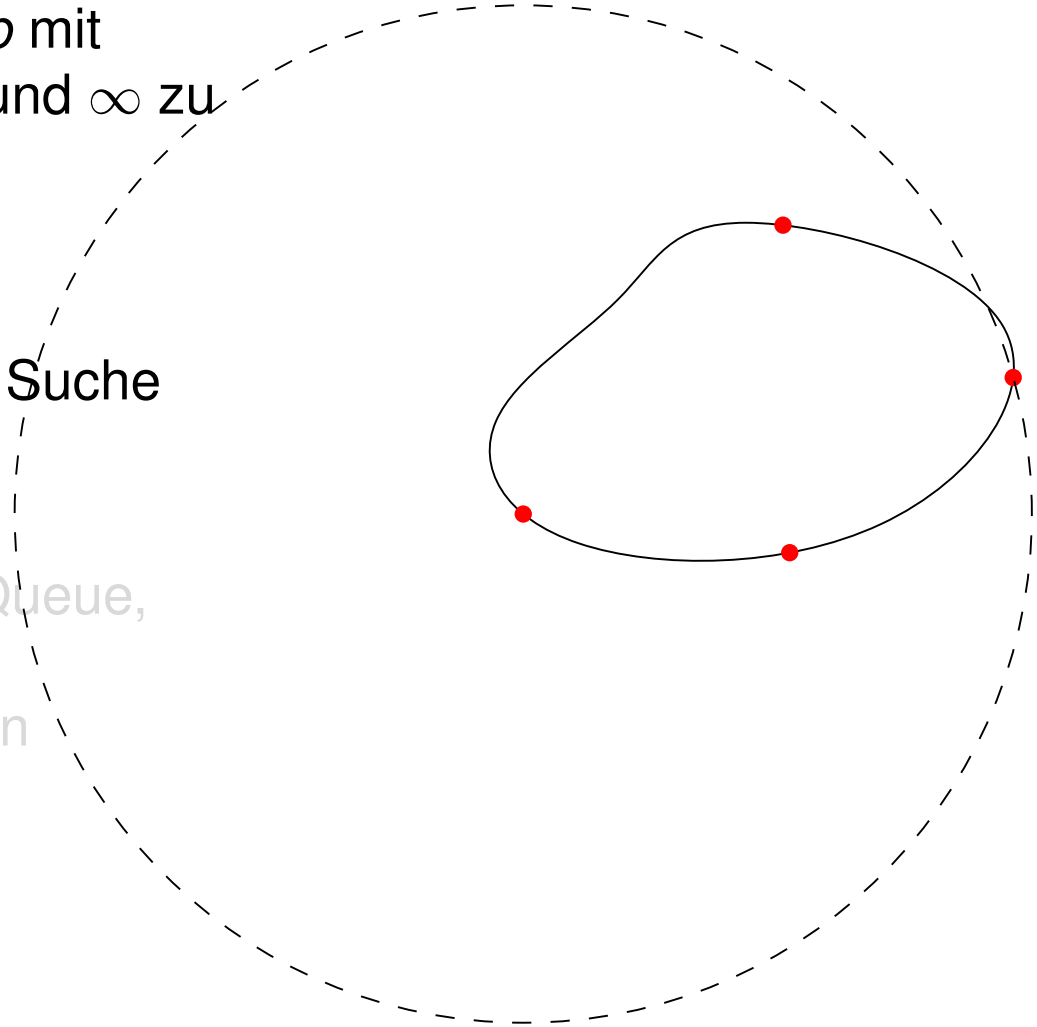
Zentralisierte Bäume: Initialisierung

trivial:

- jeder Randknoten b mit Abstand 0 zu sich und ∞ zu allen anderen

besser:

- von jedem b lokale Suche zu allen anderen Randknoten
- alle Knoten in die Queue, die auf Rand der Schnittmenge liegen



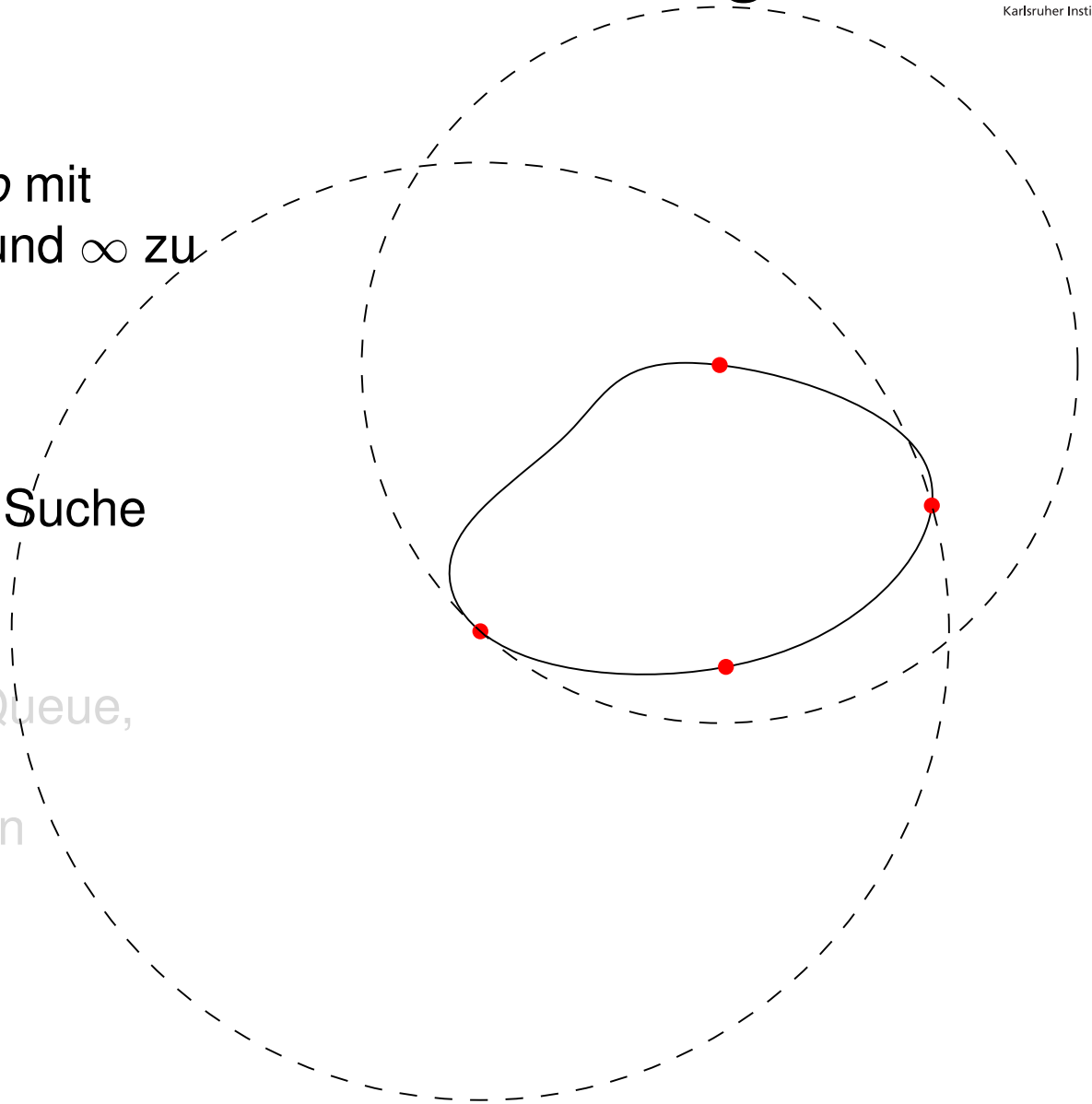
Zentralisierte Bäume: Initialisierung

trivial:

- jeder Randknoten b mit Abstand 0 zu sich und ∞ zu allen anderen

besser:

- von jedem b lokale Suche zu allen anderen Randknoten
- alle Knoten in die Queue, die auf Rand der Schnittmenge liegen



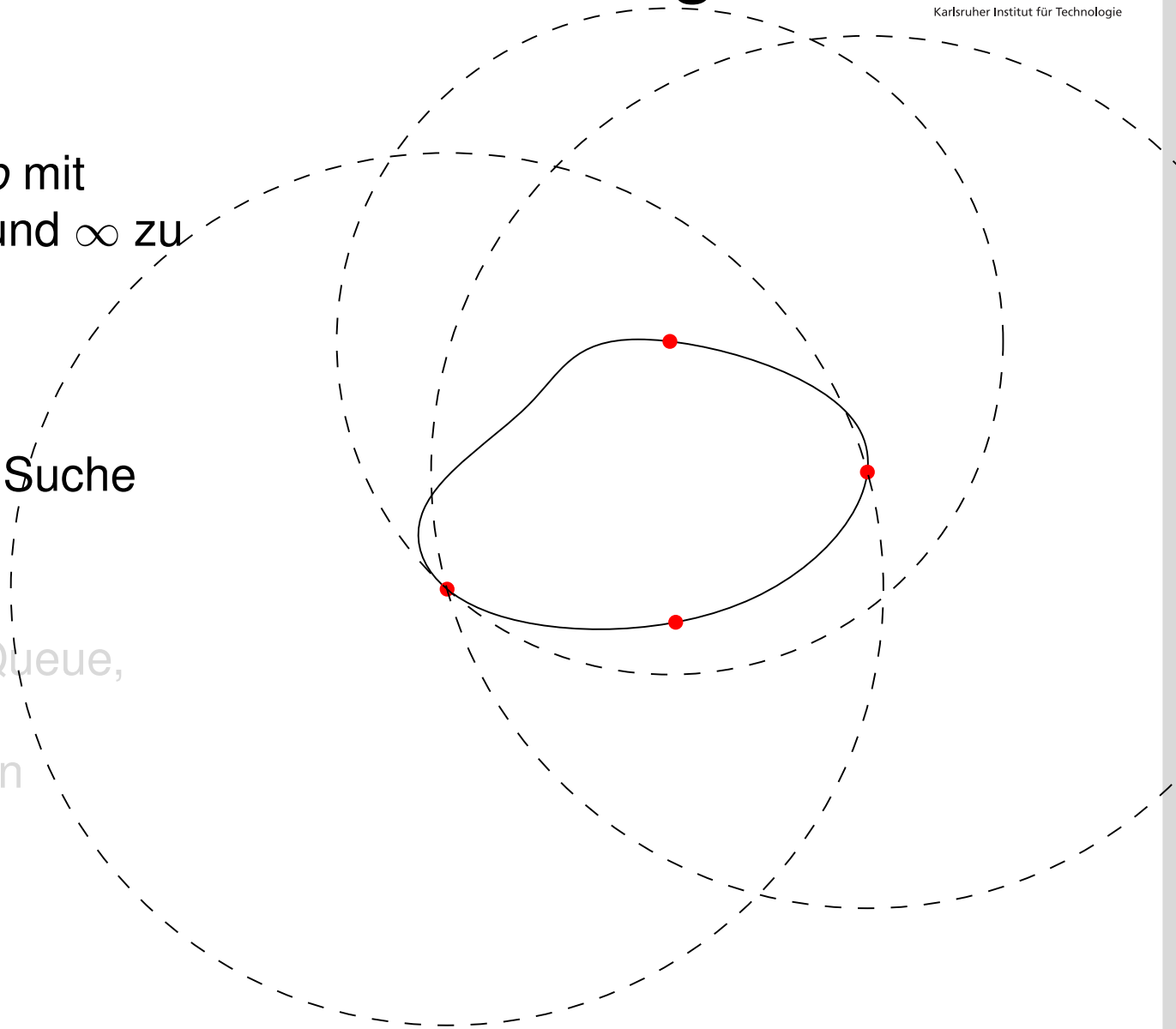
Zentralisierte Bäume: Initialisierung

trivial:

- jeder Randknoten b mit Abstand 0 zu sich und ∞ zu allen anderen

besser:

- von jedem b lokale Suche zu allen anderen Randknoten
- alle Knoten in die Queue, die auf Rand der Schnittmenge liegen



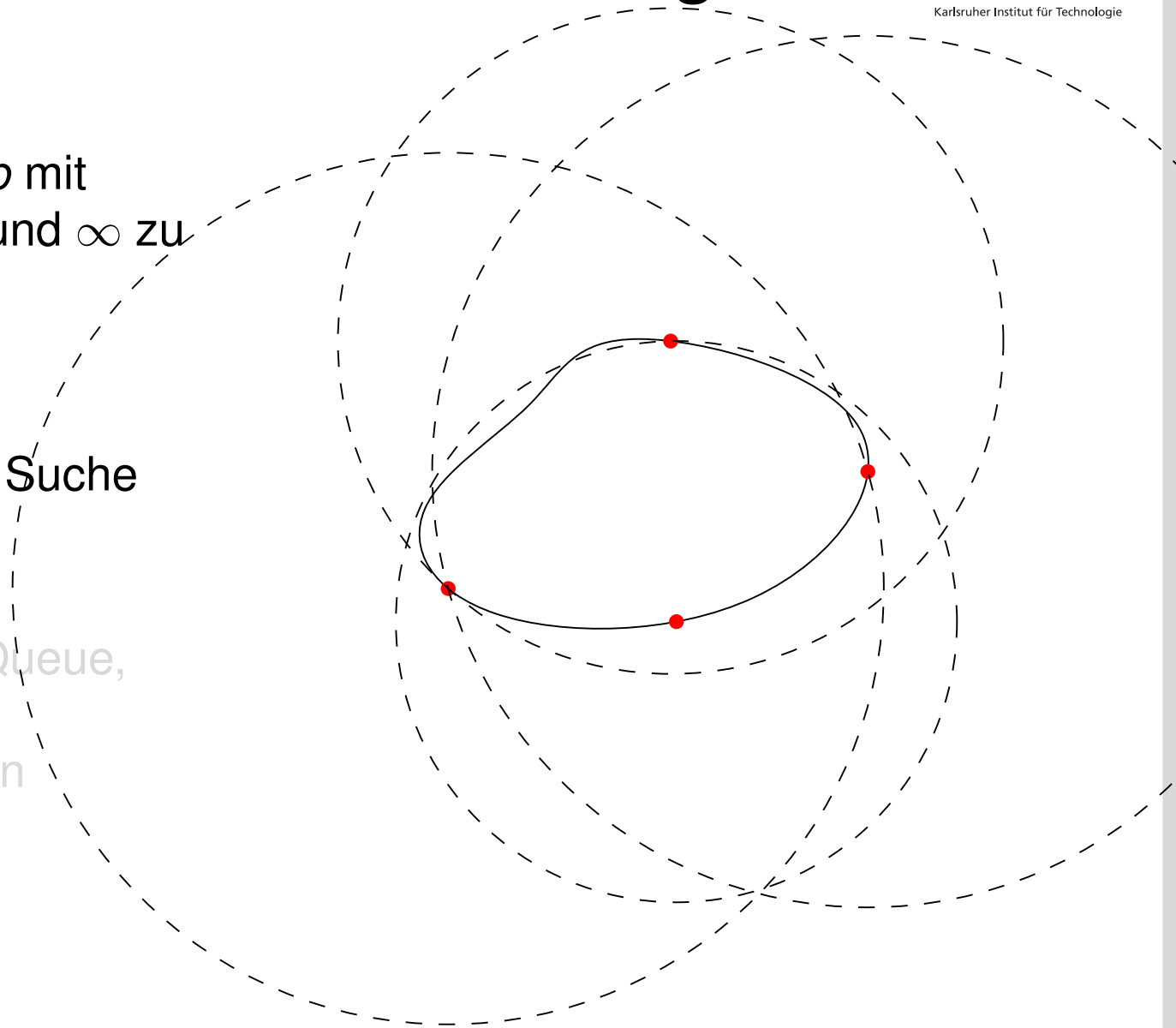
Zentralisierte Bäume: Initialisierung

trivial:

- jeder Randknoten b mit Abstand 0 zu sich und ∞ zu allen anderen

besser:

- von jedem b lokale Suche zu allen anderen Randknoten
- alle Knoten in die Queue, die auf Rand der Schnittmenge liegen



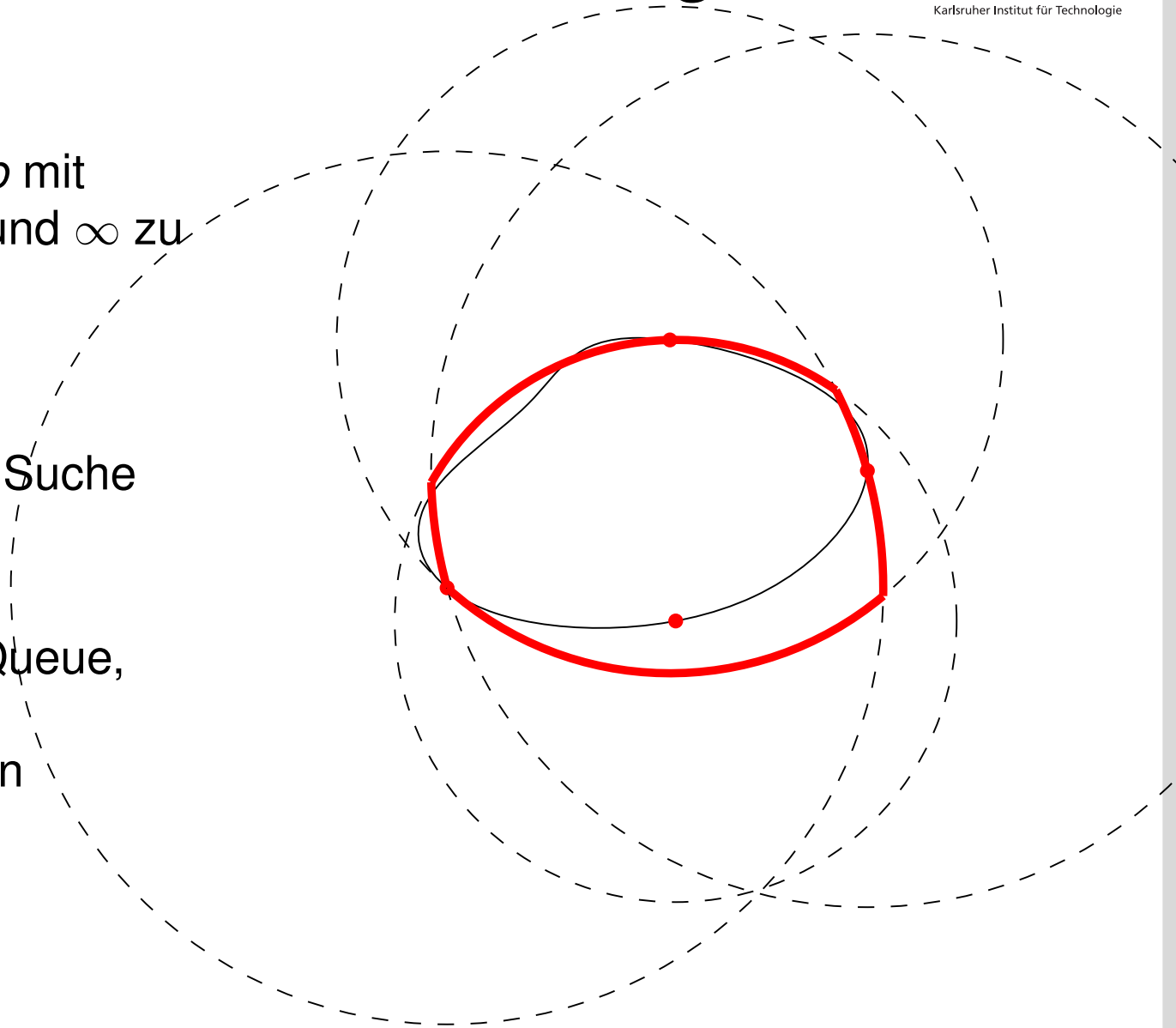
Zentralisierte Bäume: Initialisierung

trivial:

- jeder Randknoten *b* mit Abstand 0 zu sich und ∞ zu allen anderen

besser:

- von jedem *b* lokale Suche zu allen anderen Randknoten
- alle Knoten in die Queue, die auf Rand der Schnittmenge liegen



Zentralisierte Bäume

Bemerkungen:

- Knoten können mehrfach besucht werden (label-correcting)
- Prioritäten der Knoten beliebig wählbar
  - z.B. Minimum über alle Einträge von u oder auch nur der vorläufigen
  - hängt von Eingabe ab
- hoher Speicherverbrauch durch n Label der Größe $|B_r|$
  - 18 Mio. Knoten und 1000 Randknoten $\Rightarrow$ 72 GB
  - teile Arbeit in mehrere Schritte mit maximal 100 Randknoten
  - meist Beschleunigung um Faktor 4 gegenüber Randknotenansatz

Partitionierung

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Multi-Way Arc-Separator:

- benutzt keine Einbettung
- teilt rekursiv Graphen in k Teile mit kleinem Schnitt
- prädestiniert für Arc-Flags

Partitionierung

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Multi-Way Arc-Separator:

- benutzt keine Einbettung
- teilt rekursiv Graphen in k Teile mit kleinem Schnitt
- prädestiniert für Arc-Flags

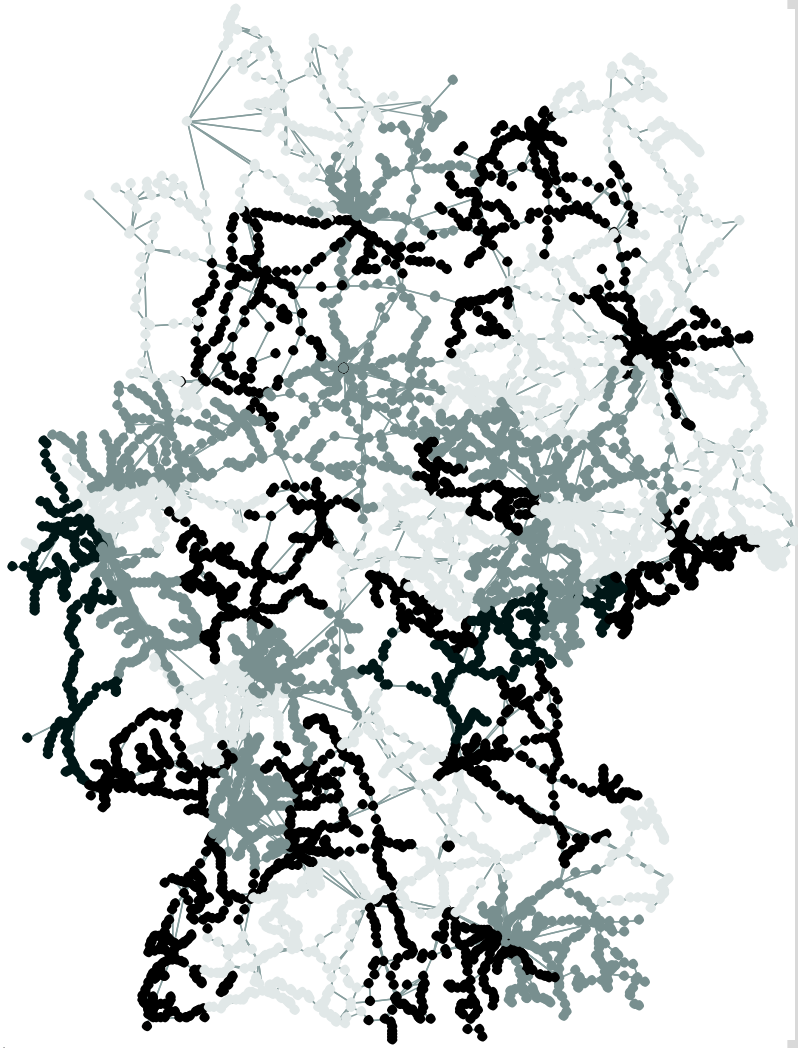
Partitionierung

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Multi-Way Arc-Separator:

- benutzt keine Einbettung
- teilt rekursiv Graphen in k Teile mit kleinem Schnitt
- prädestiniert für Arc-Flags



Anfragen

ARC-FLAG-DIJKSTRA($G = (V, E)$, s, t)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      forall edges e = (u, v) ∈ E do
6          if AF_T(e) = false 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])
```

Korrektheit

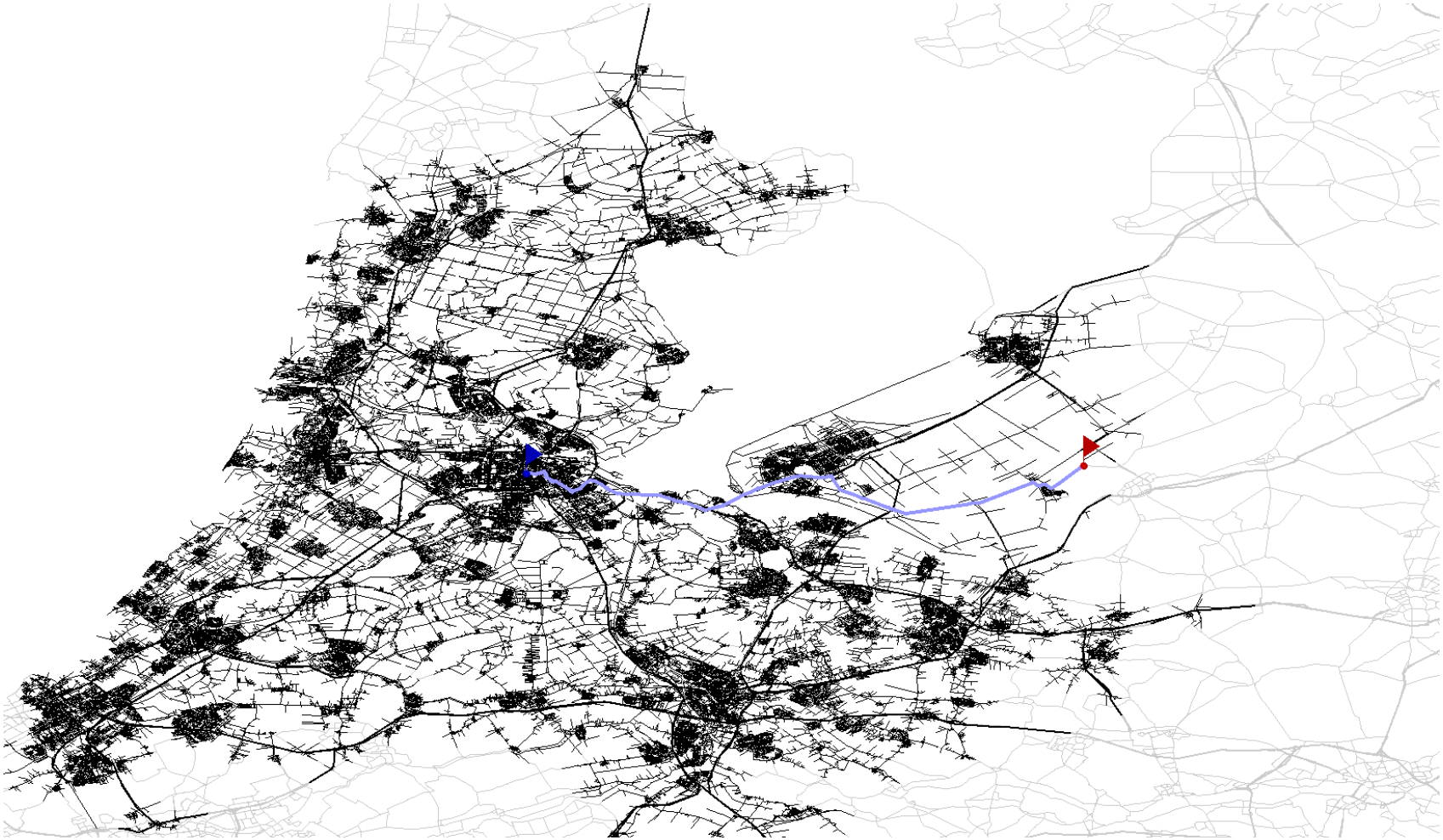
Korrektheit

`ArcFlagsDijsktra` ist korrekt

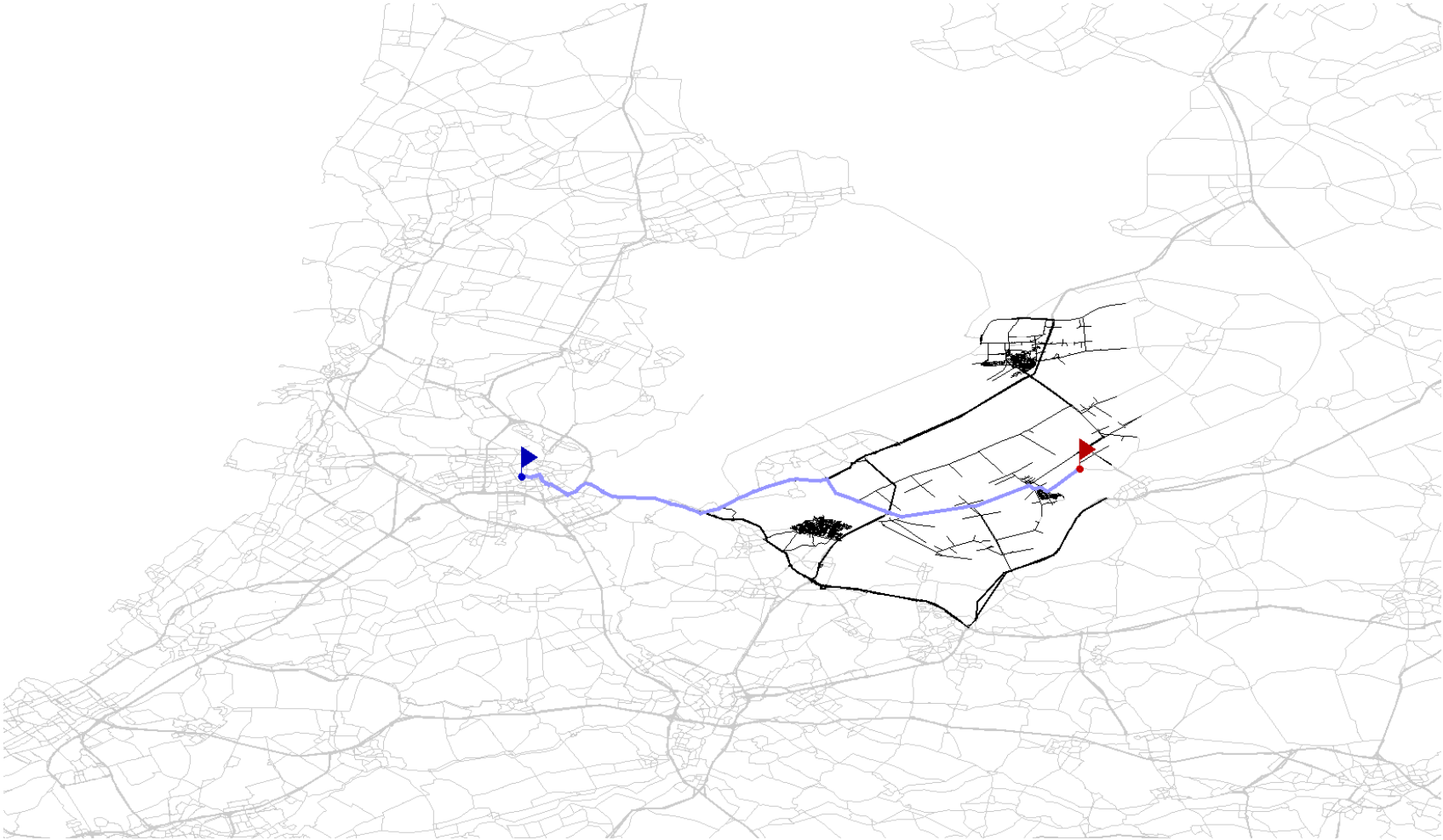
zu zeigen: für jeden kürzesten s-t-Weg $P = (s = u_1, \ldots, u_k = t)$ haben alle Kanten (u_i, u_{i+1}) die Flagge $AF_T(u_i, u_{i+1})$ gesetzt.

- für alle (u_i, u_{i+1}) in T trivial
- sei u_j der letzte Knoten auf P in T
- u_j also Randknoten und somit Rückwärtsbaum gebaut von u_i (einzeln oder zentralisiert)
- $P' = (u_1, \ldots, u_j)$ ist Subpfad von P und somit ein kürzester Weg von u_1 nach u_j
- somit Baumkanten mit gesetzten Flaggen

Suchraum



Suchraum



Diskussion

Vorteile:

- einfacher Anfrage Algorithmus
- Vorberechnung basiert nicht auf APSP

Nachteile:

- Randknotenansatz: lange Vorberechnung, ca. 3 Tage für Europa
- zentralisierter Ansatz:
  - hoher Speicherverbrauch währende Vorberechnung
  - moderate Dauer (aber immer noch ca. 18 Stunden für Europa)
- mehr Flaggen gesetzt, wenn nah an der Zielzelle
- alle Flaggen in Zielzelle gesetzt
- Anfragen in einer Zelle ohne Beschleunigung

Diskussion

Vorteile:

- einfacher Anfrage Algorithmus
- Vorberechnung basiert nicht auf APSP

Nachteile:

- Randknotenansatz: lange Vorberechnung, ca. 3 Tage für Europa
- zentralisierter Ansatz:
  - hoher Speicherverbrauch währende Vorberechnung
  - moderate Dauer (aber immer noch ca. 18 Stunden für Europa)
- mehr Flaggen gesetzt, wenn nah an der Zielzelle
- alle Flaggen in Zielzelle gesetzt
- Anfragen in einer Zelle ohne Beschleunigung

Diskussion

Vorteile:

- einfacher Anfrage Algorithmus
- Vorberechnung basiert nicht auf APSP

Nachteile:

- Randknotenansatz: lange Vorberechnung, ca. 3 Tage für Europa
- zentralisierter Ansatz:
  - hoher Speicherverbrauch währende Vorberechnung
  - moderate Dauer (aber immer noch ca. 18 Stunden für Europa)
- 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

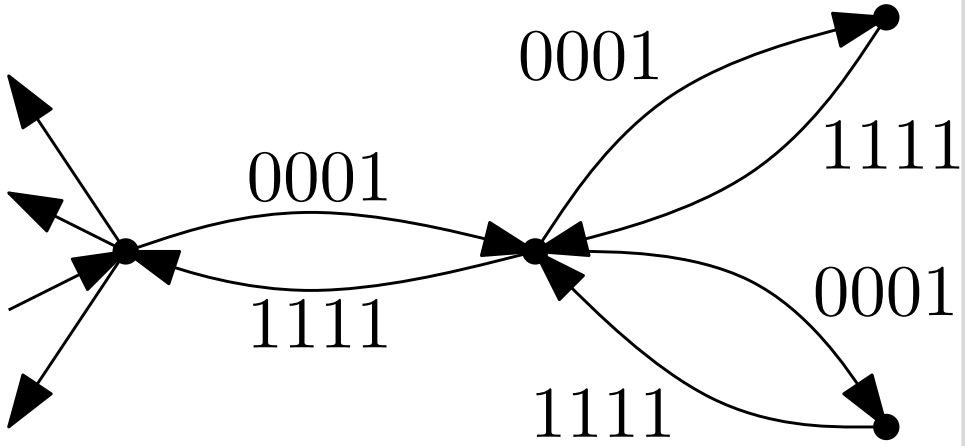
Angehangene Bäume:

- 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

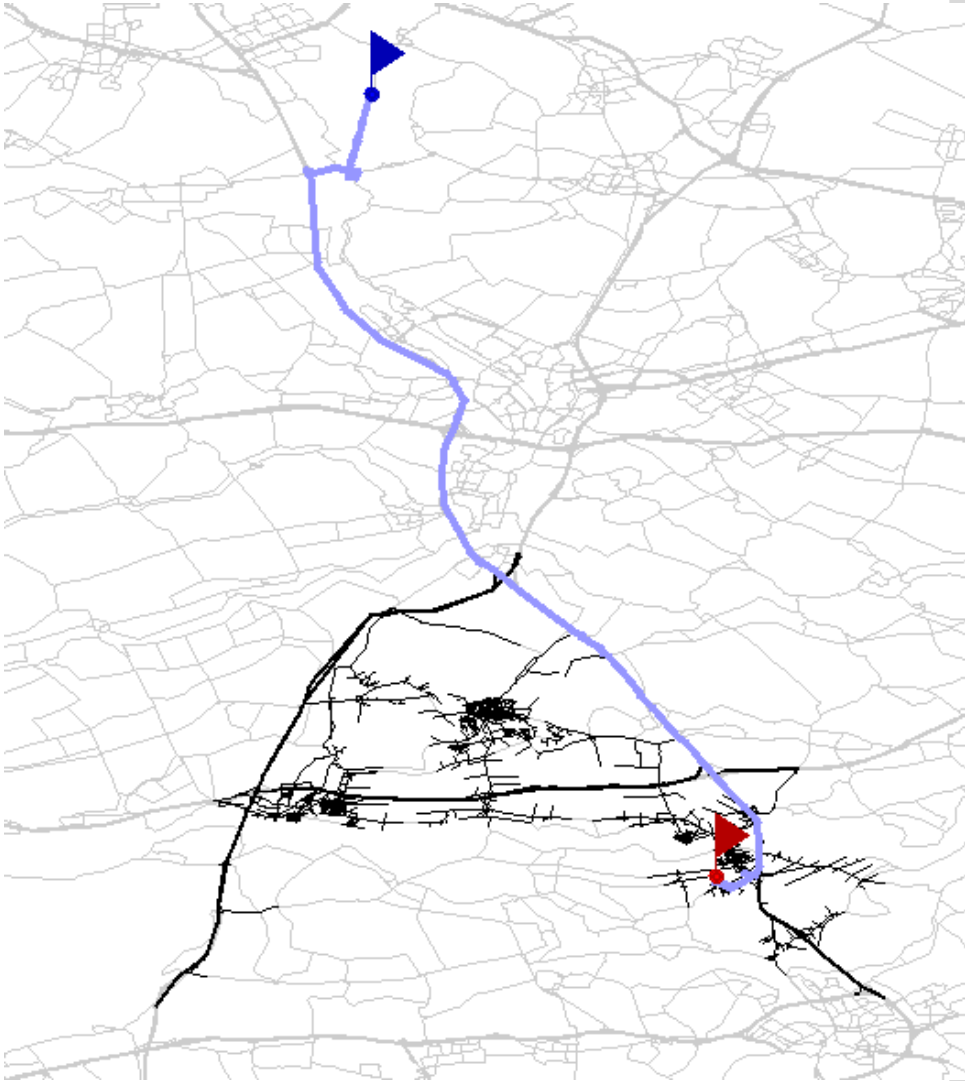
Coning-Effekt

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



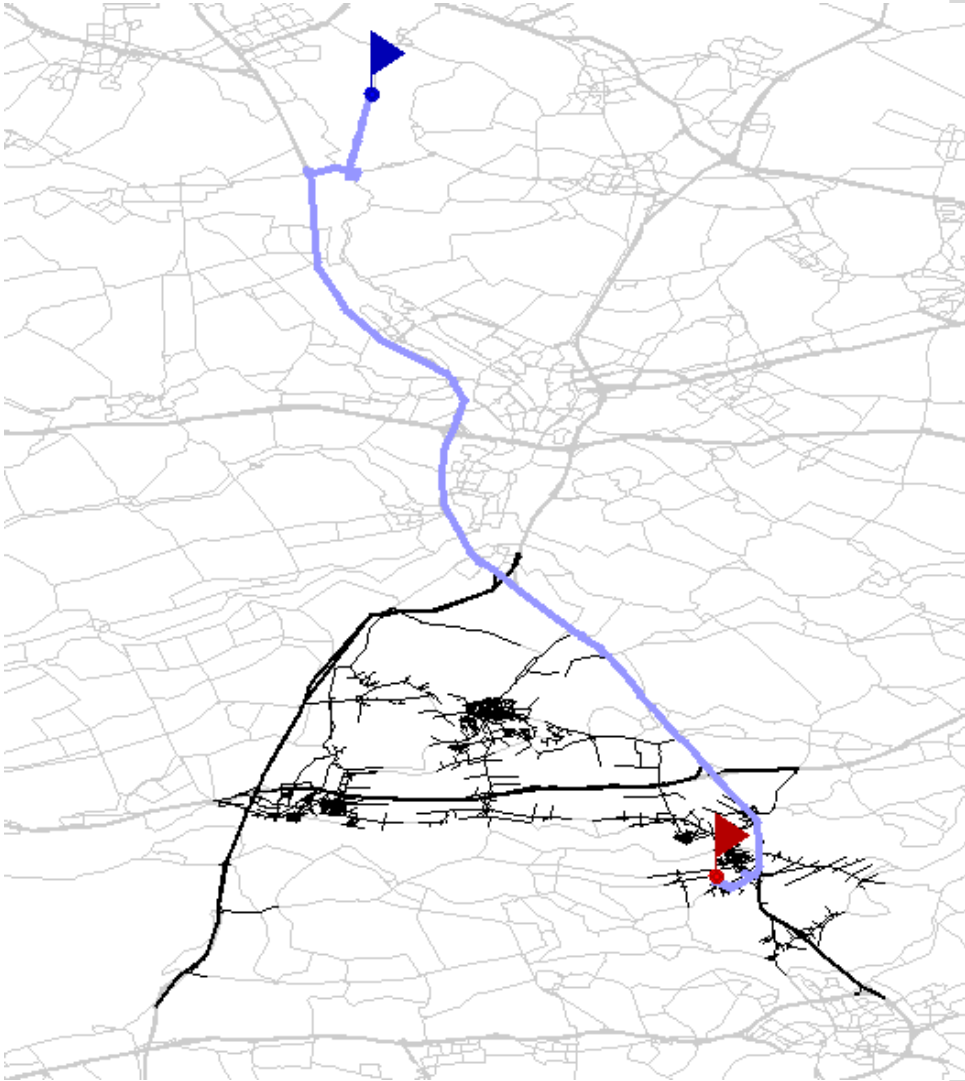
Coning-Effekt

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

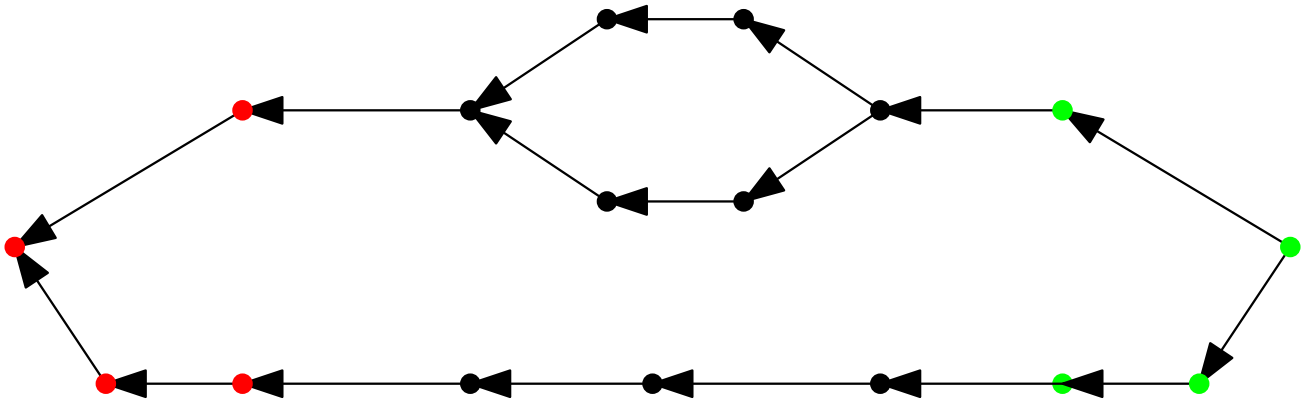
Anfrage:

- 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



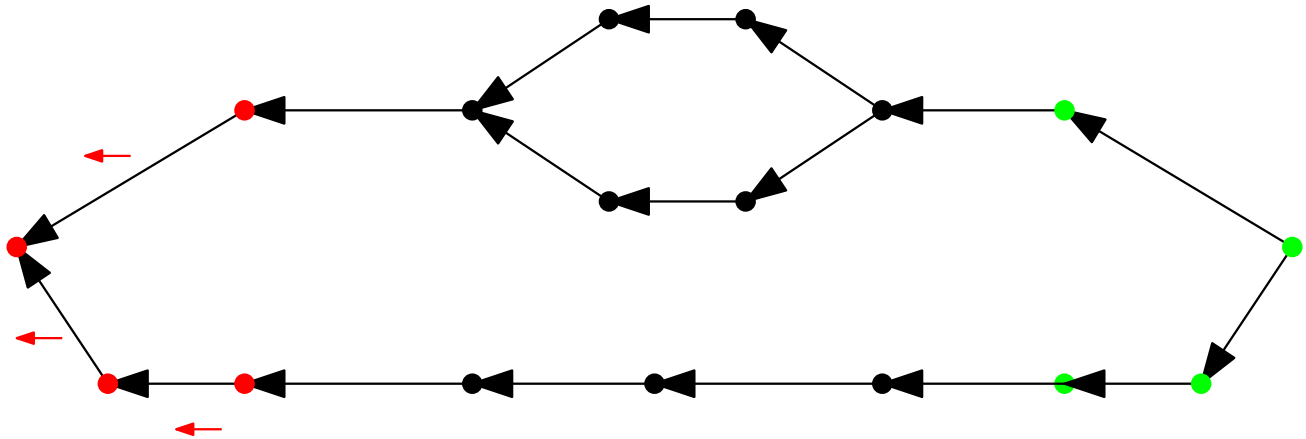
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



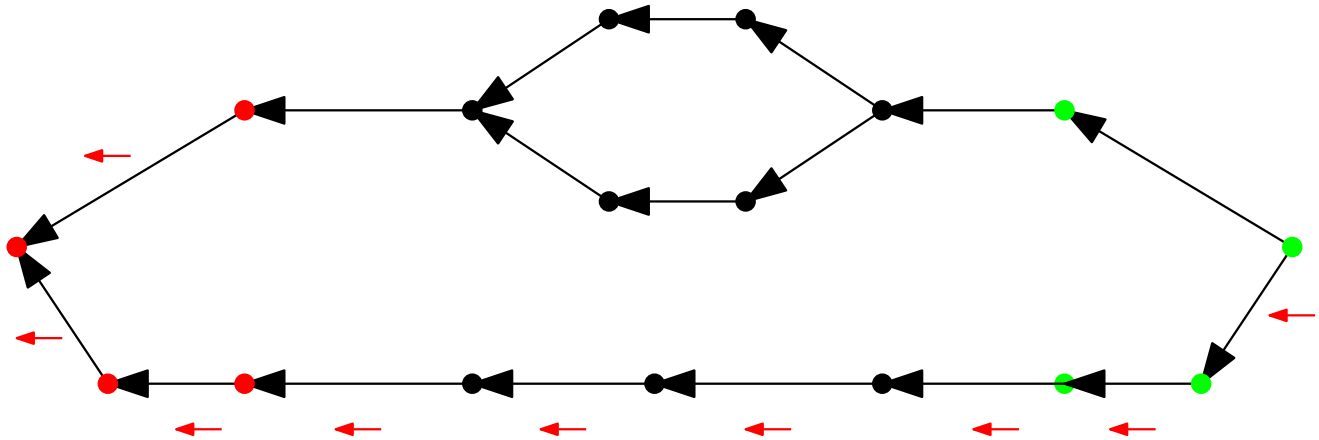
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



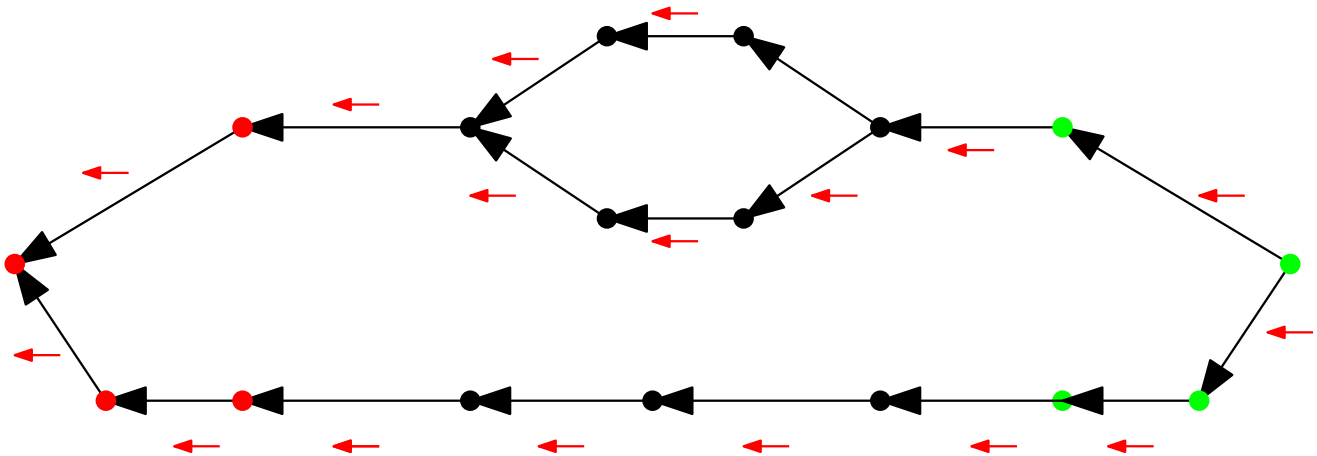
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



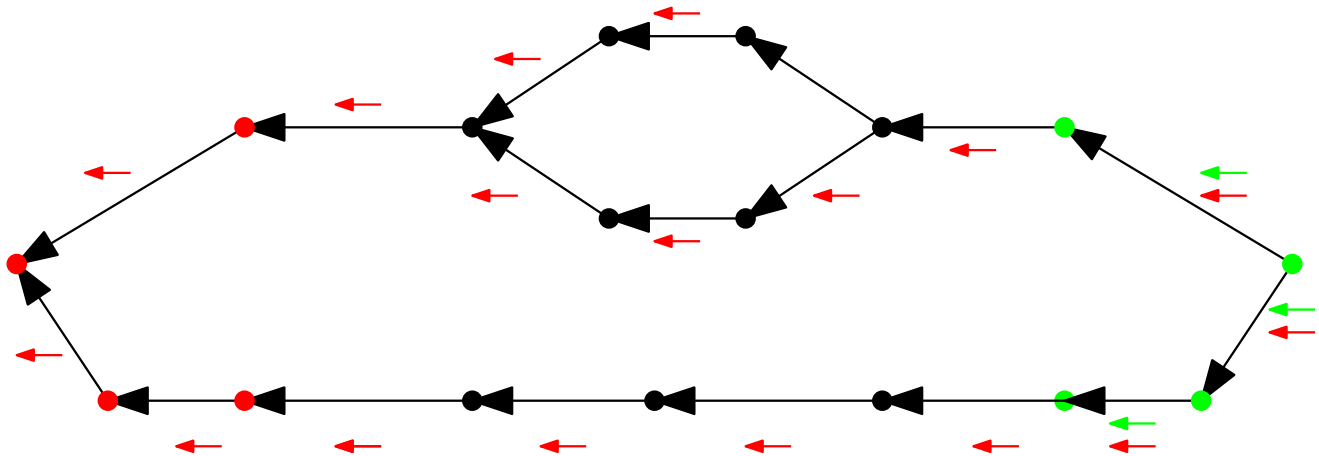
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



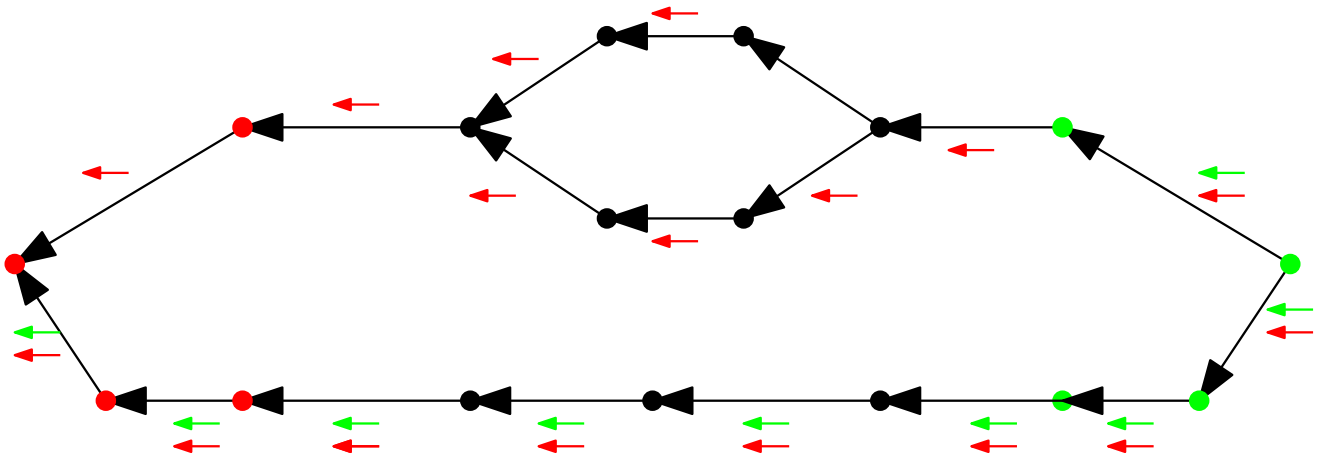
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



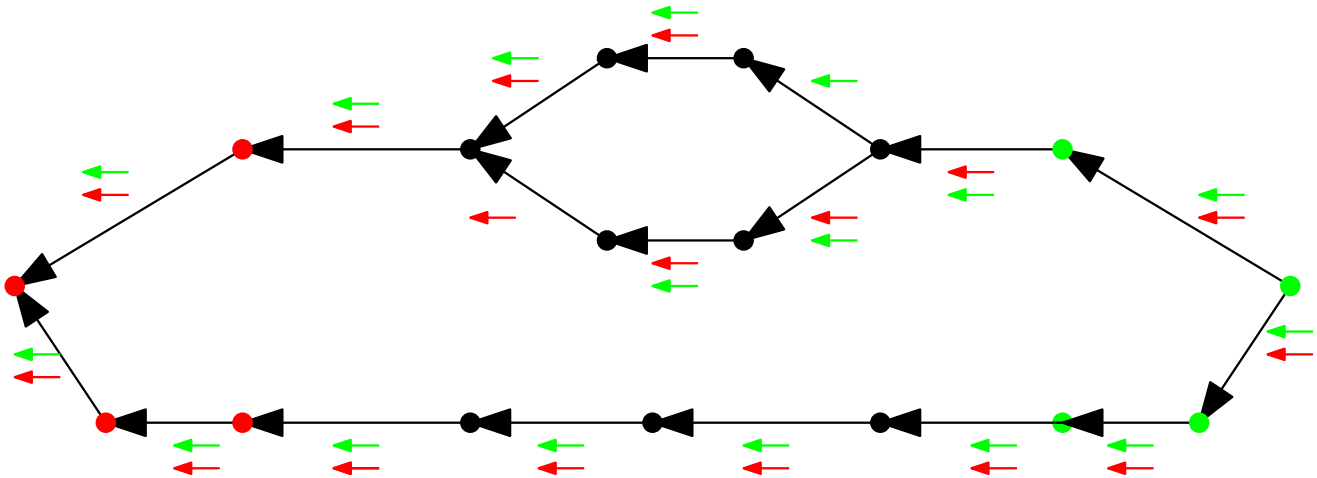
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



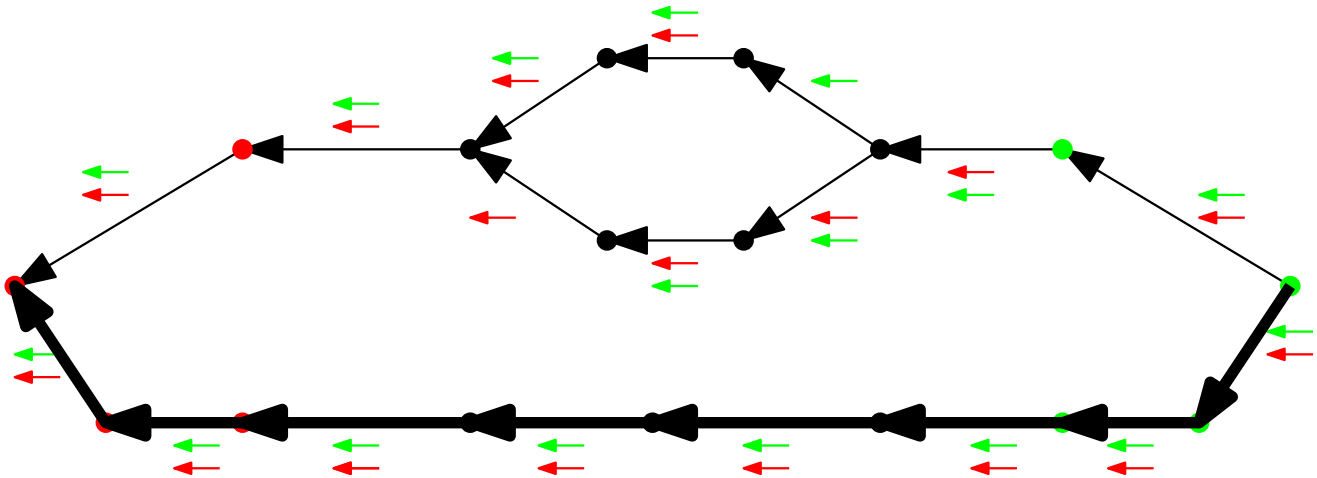
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



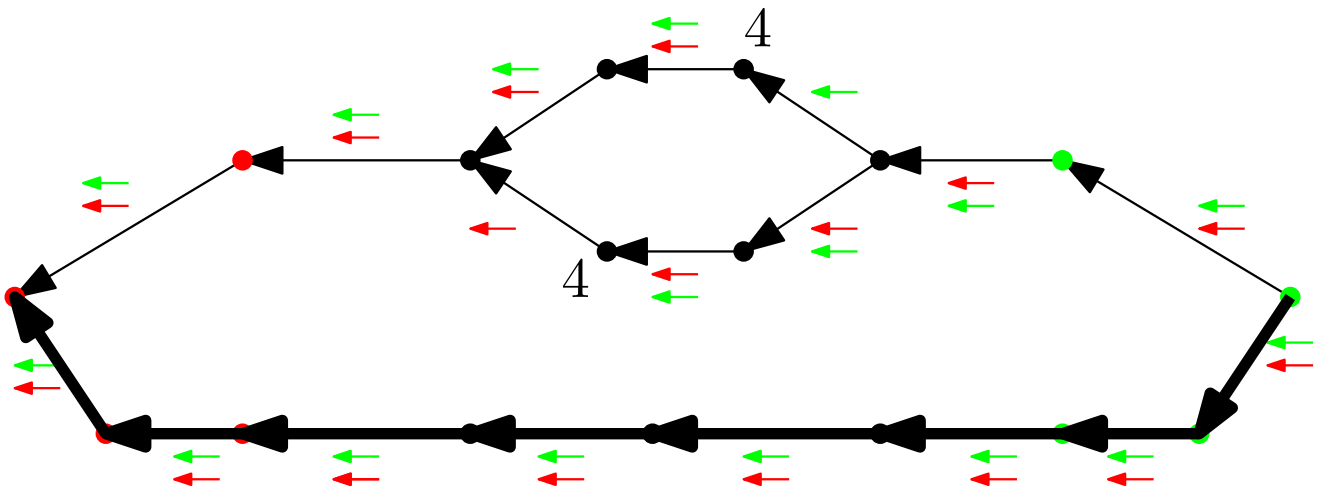
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



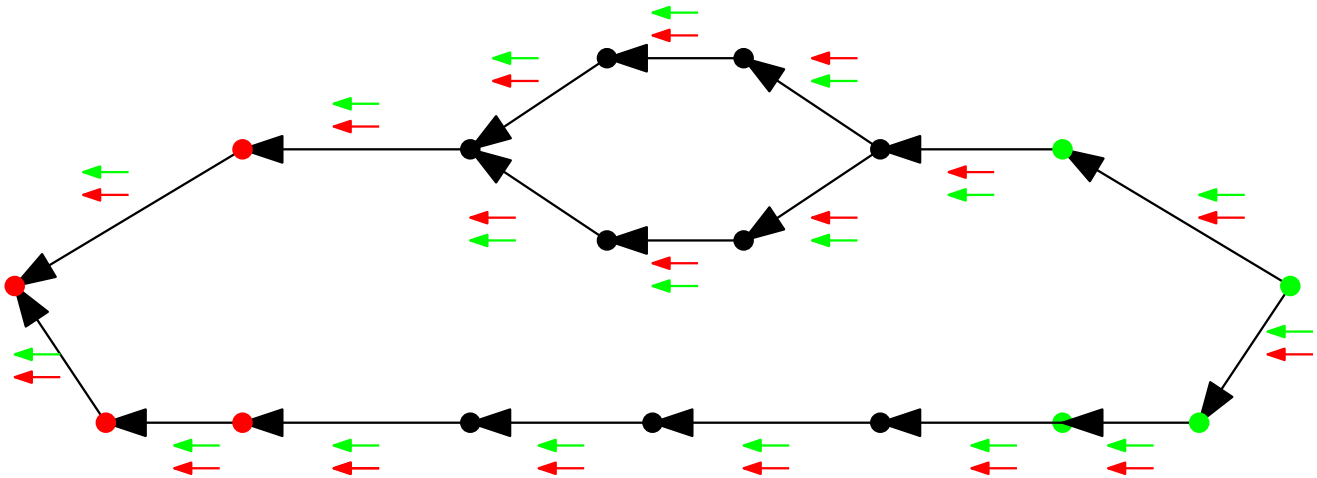
Lösung:

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

Bidirektionale Arc-Flags

Problem:

- Eindeutigkeit der Wege
- eventuell nicht korrekt



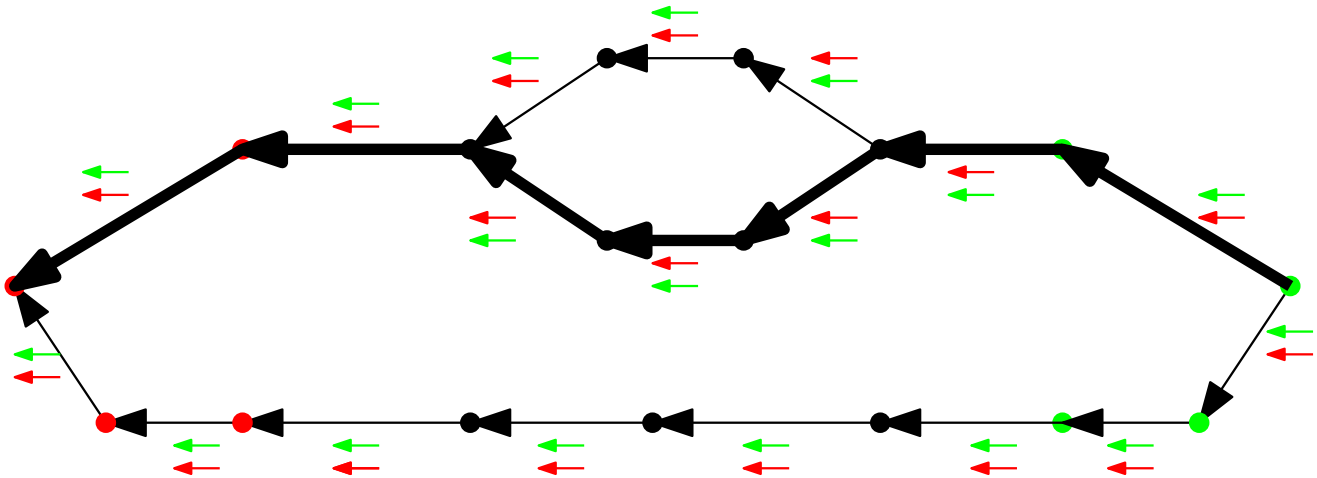
Lösung:

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

Bidirektionale Arc-Flags

Problem:

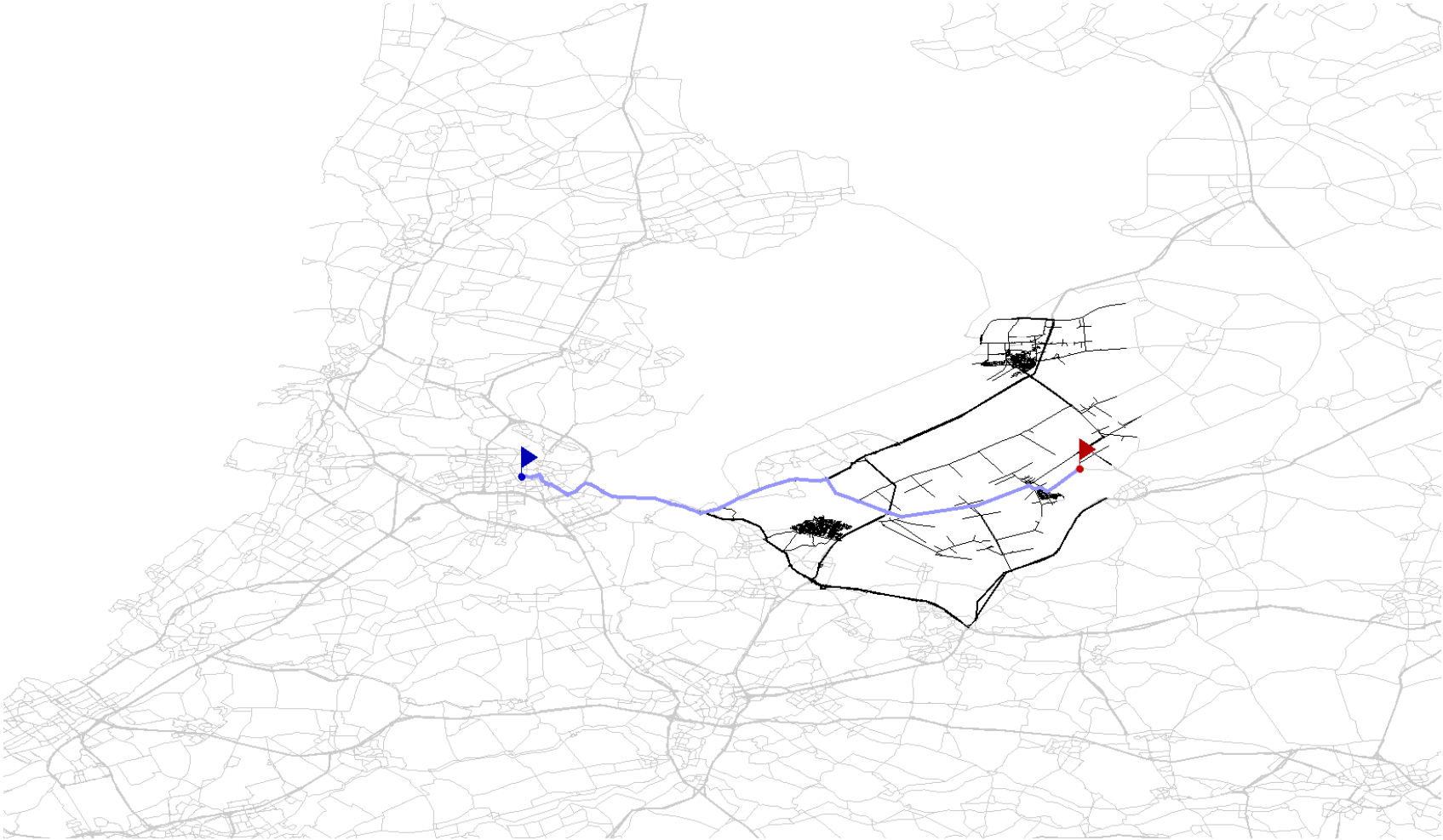
- Eindeutigkeit der Wege
- eventuell nicht korrekt



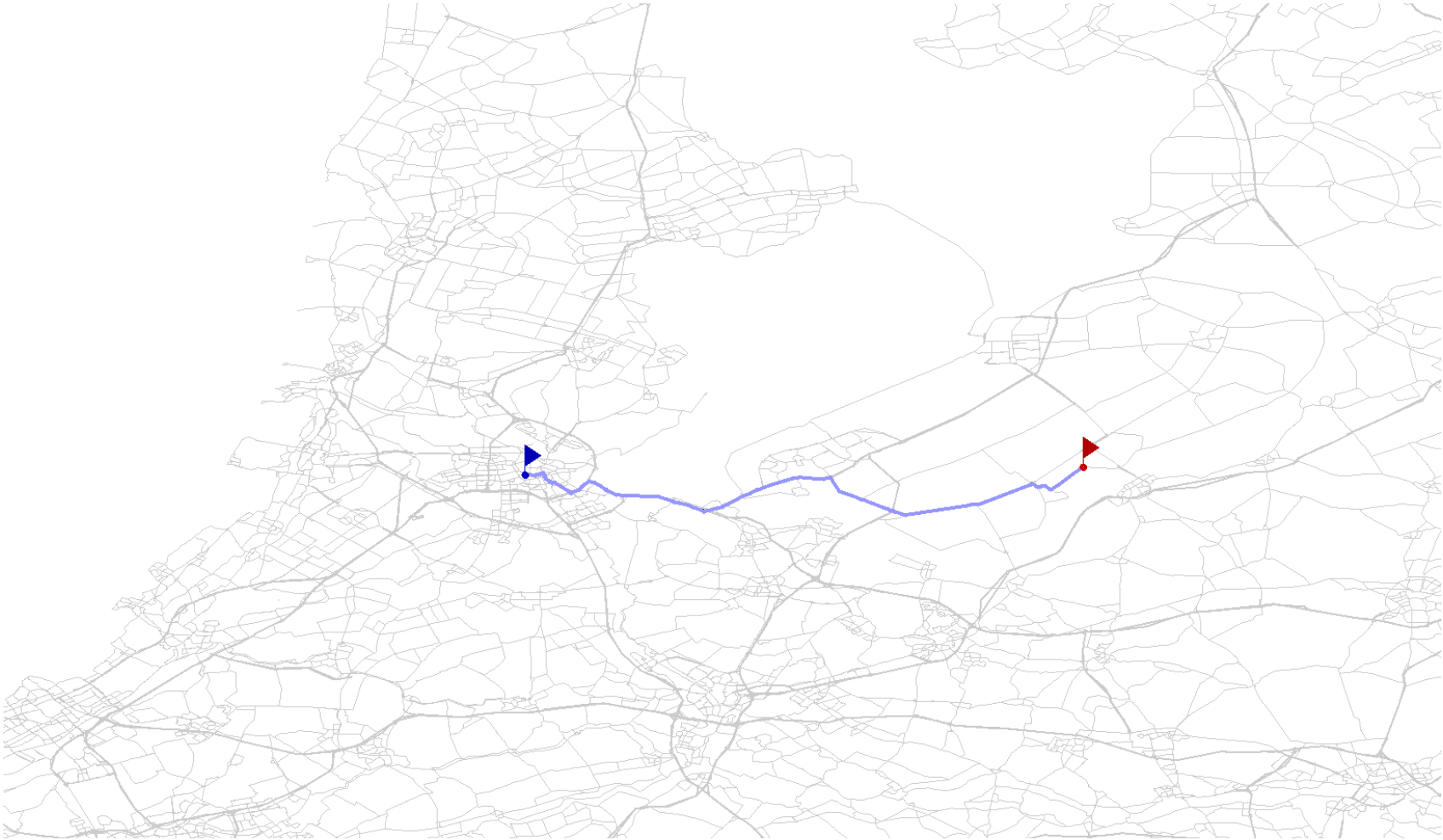
Lösung:

- 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

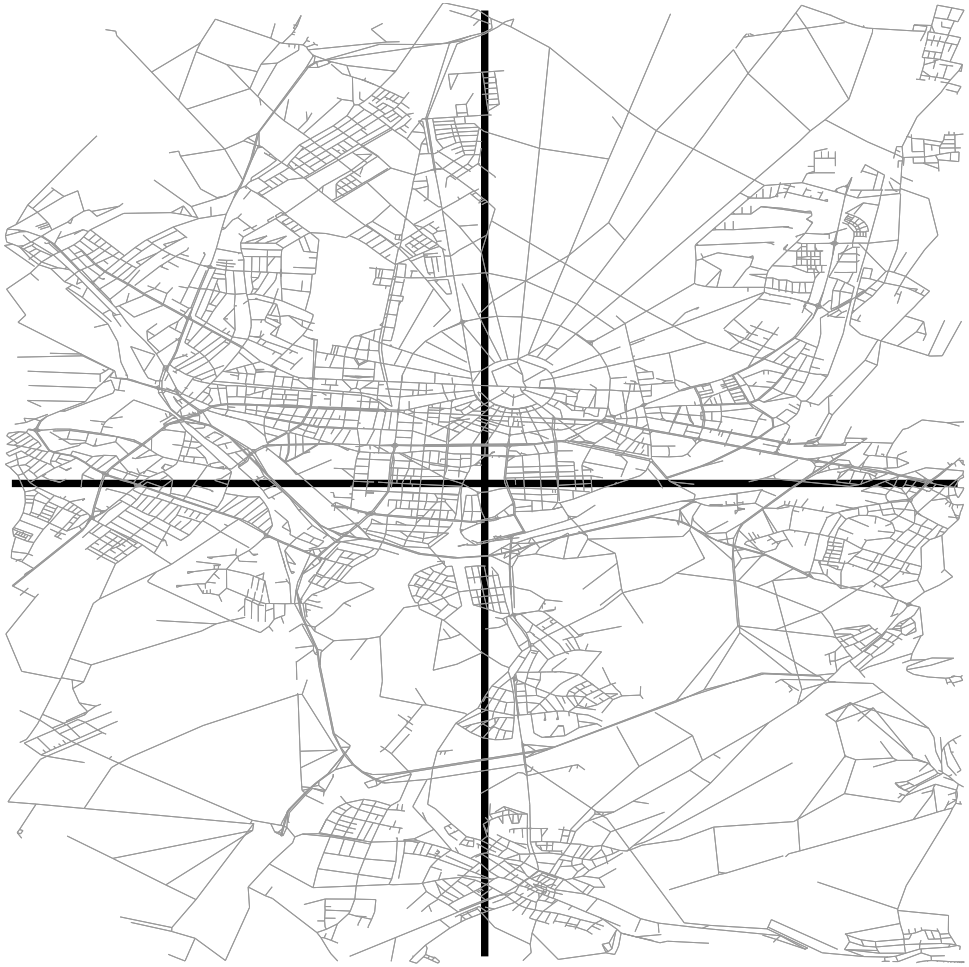
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



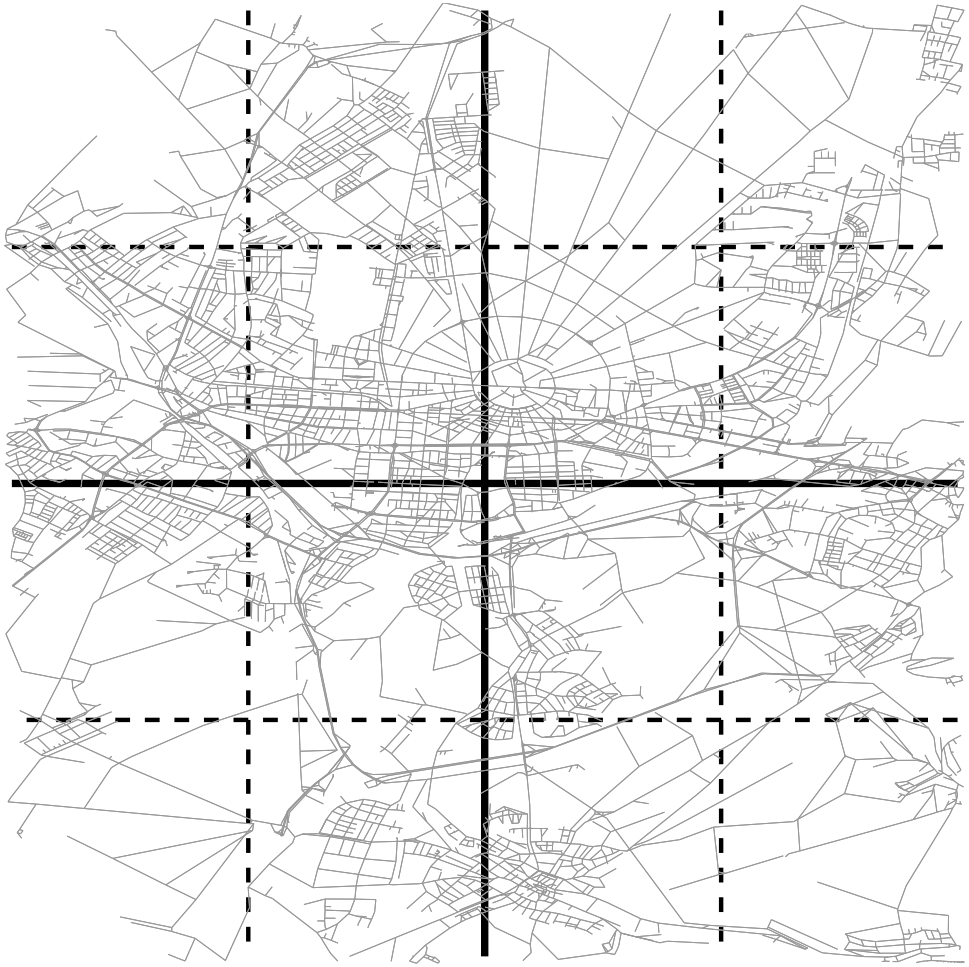
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



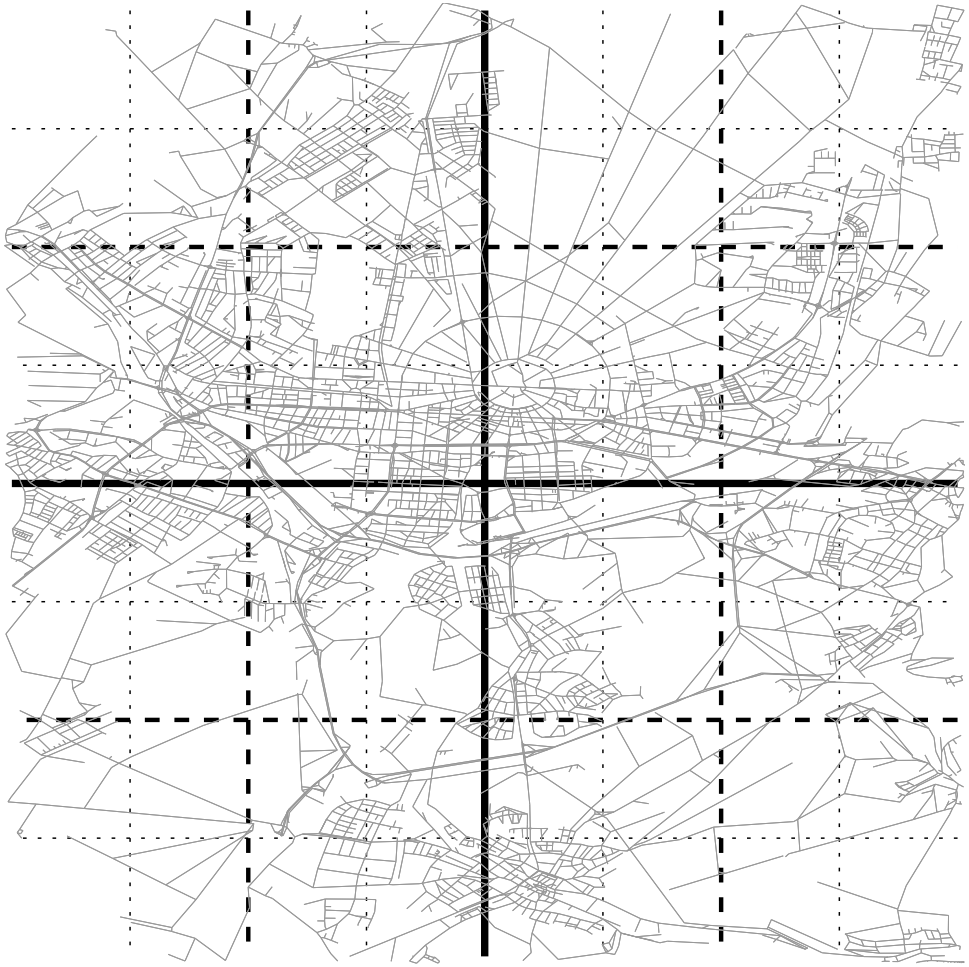
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



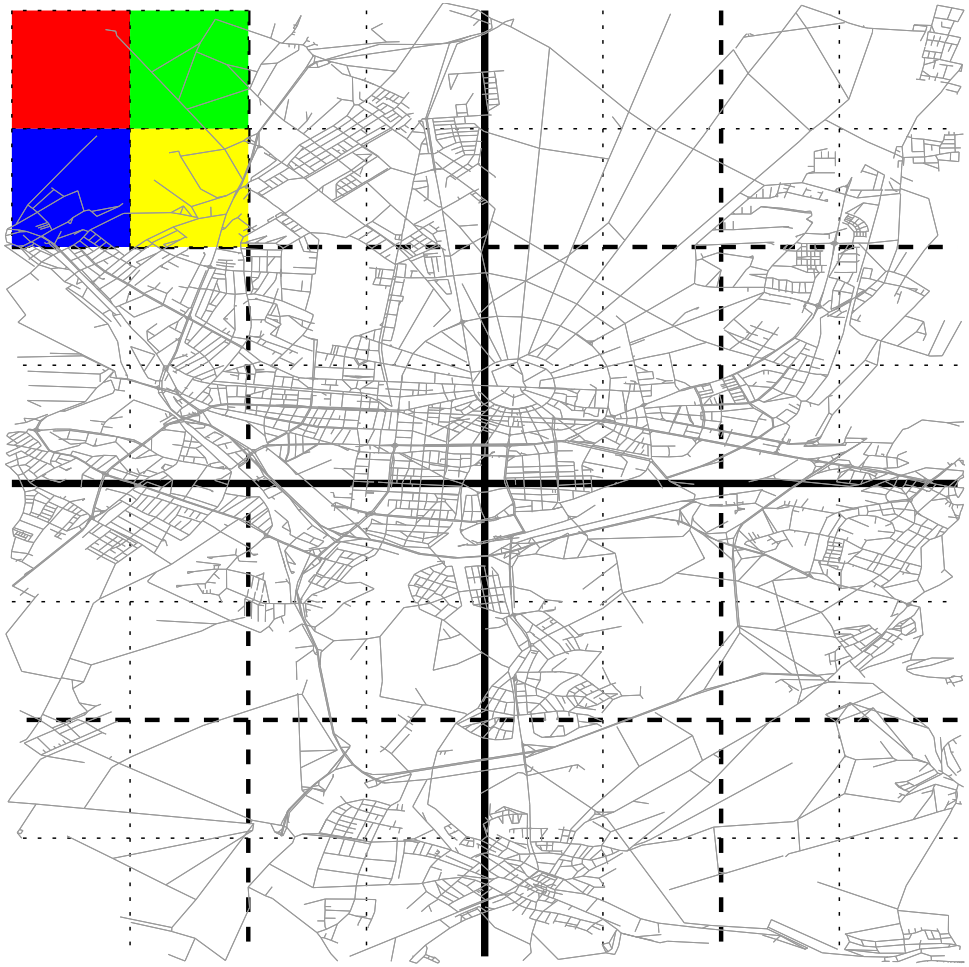
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



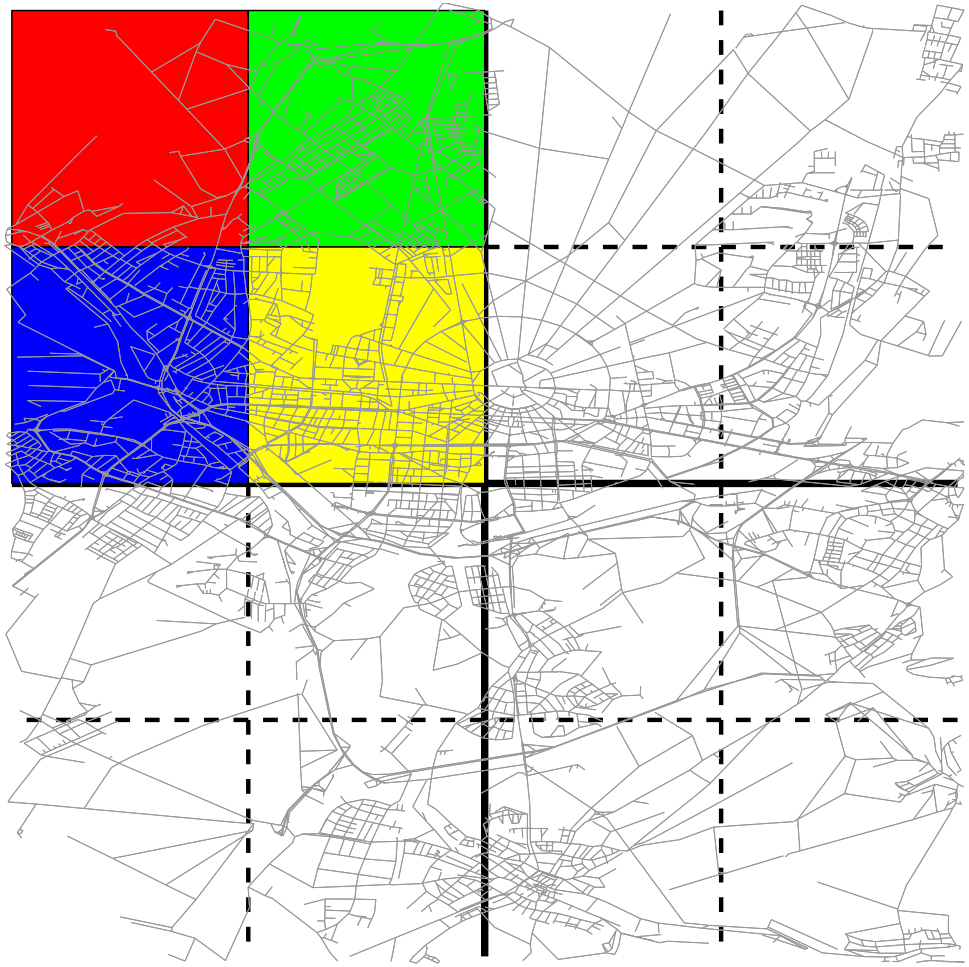
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



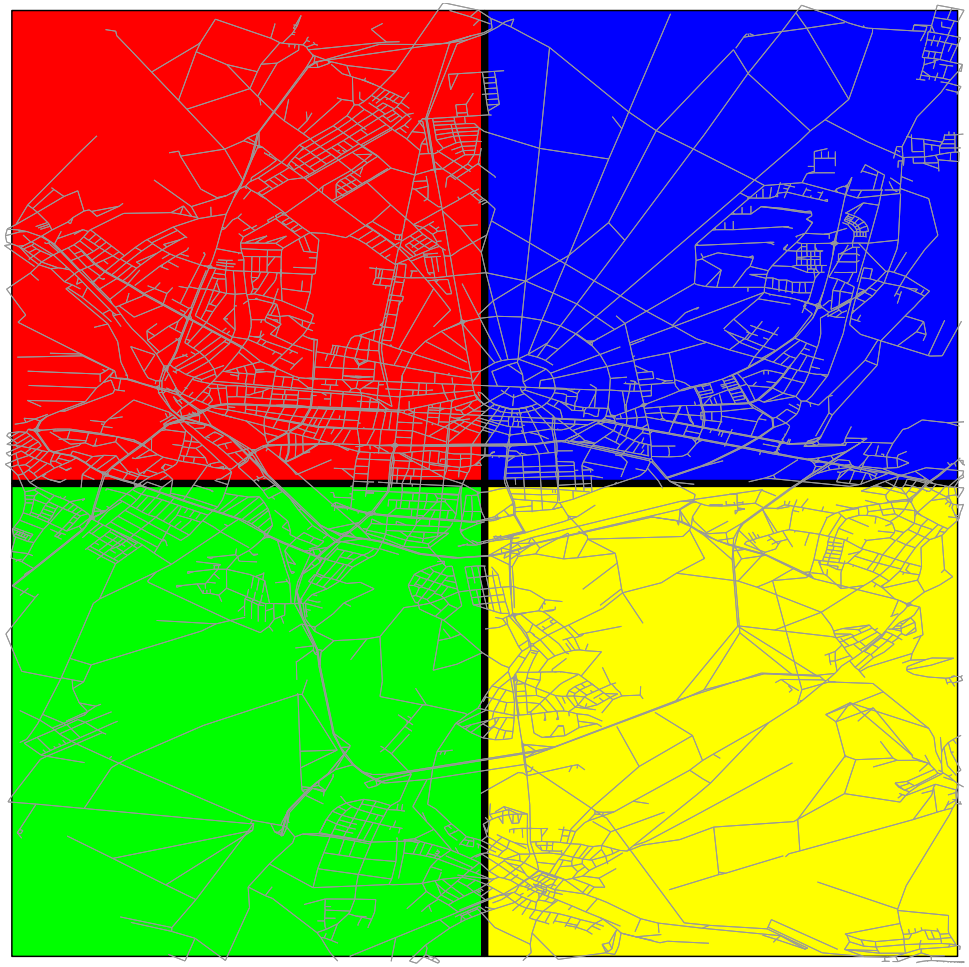
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



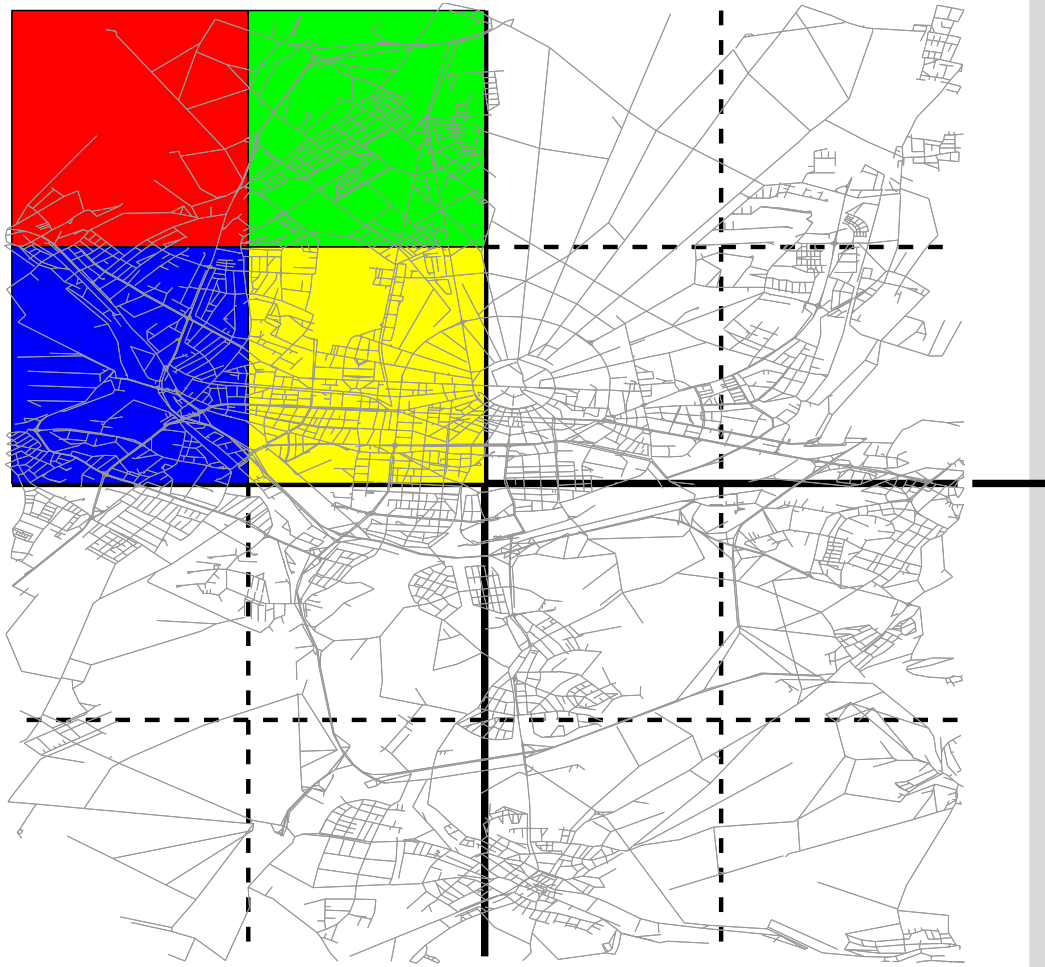
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



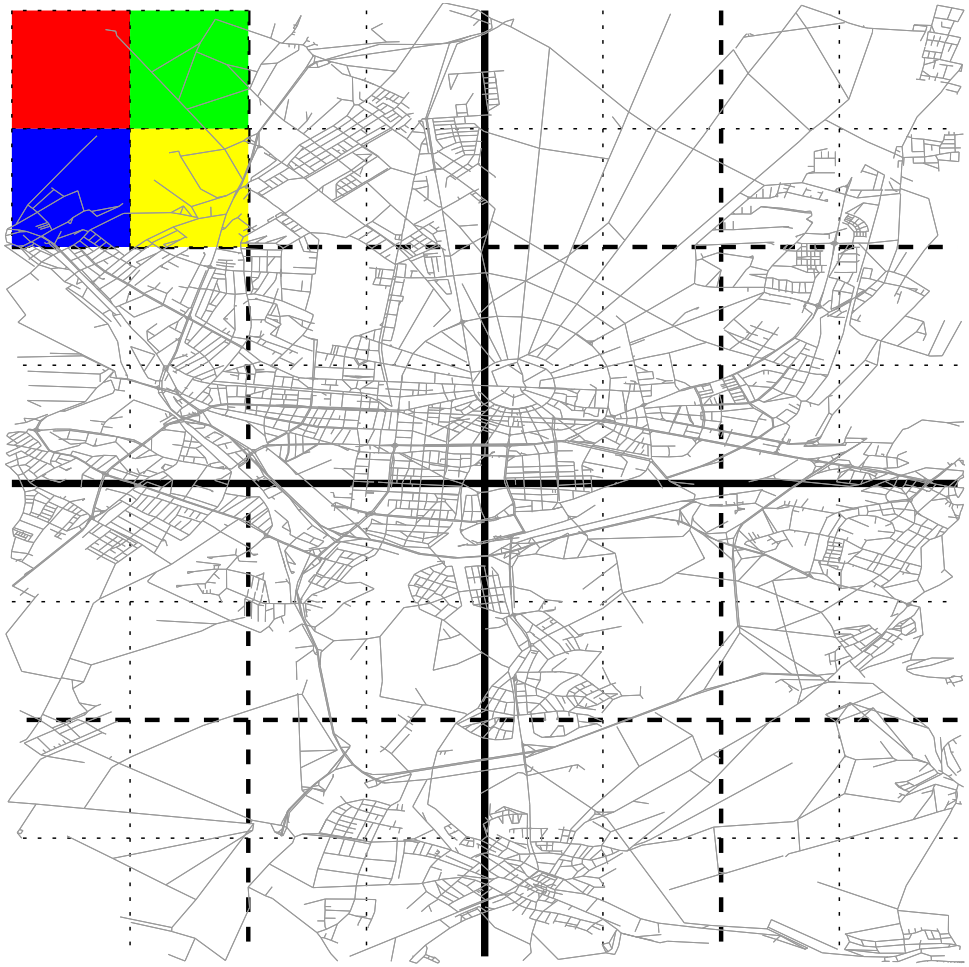
Multi-Level Arc-Flags

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:

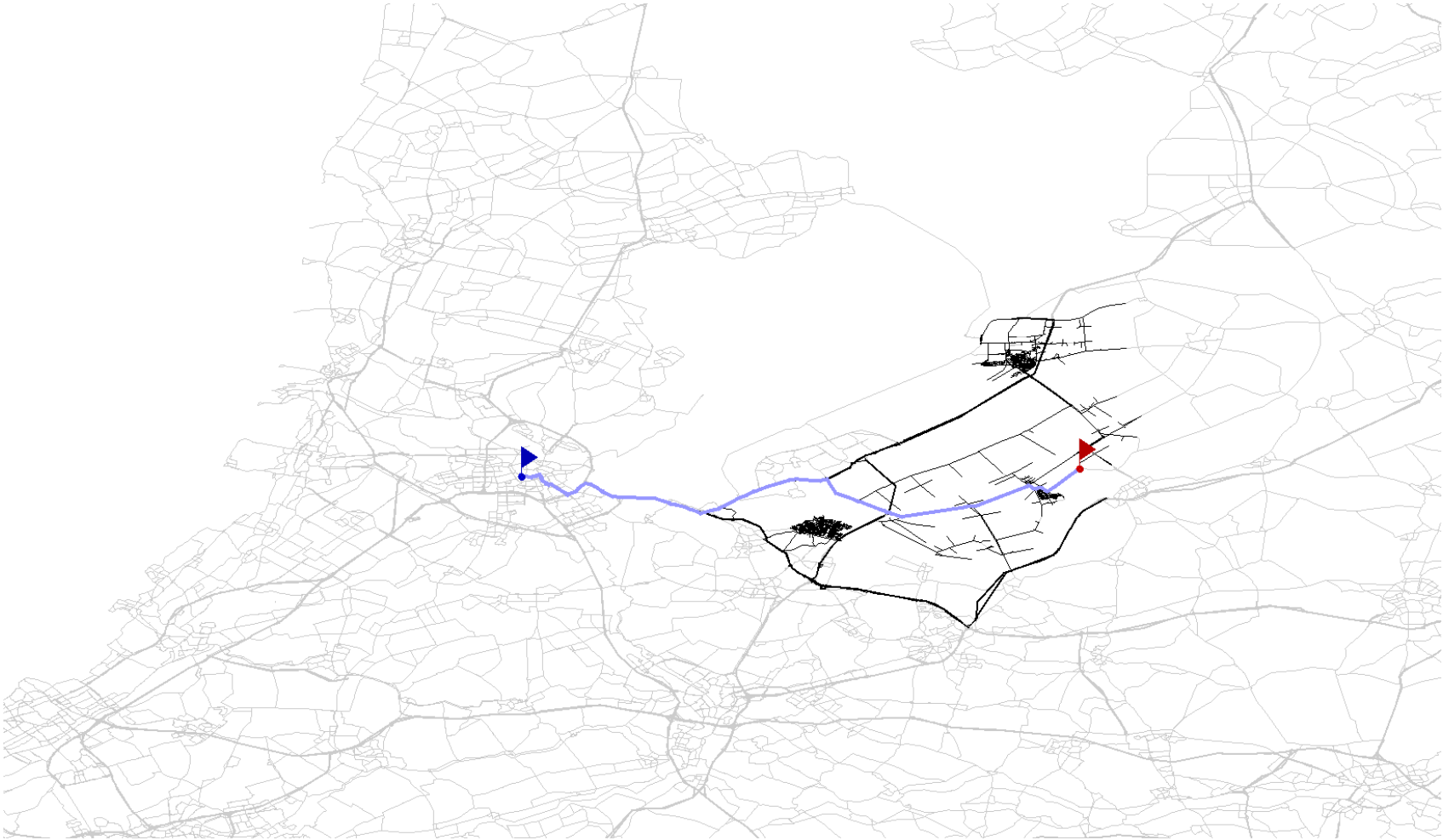
- oberster Level wie gehabt
- auf unteren Leveln:
  - von jedem Randknoten führe Dijkstra aus, bis alle Knoten der Superzelle abgearbeitet worden sind
  - setze Flaggen für alle Kanten (u, v) für die u in Superzelle
  - Hinweis: es reicht nicht, nur den Subgraphen der Superzelle zu betrachten (Übungsaufgabe)

Anfragen:

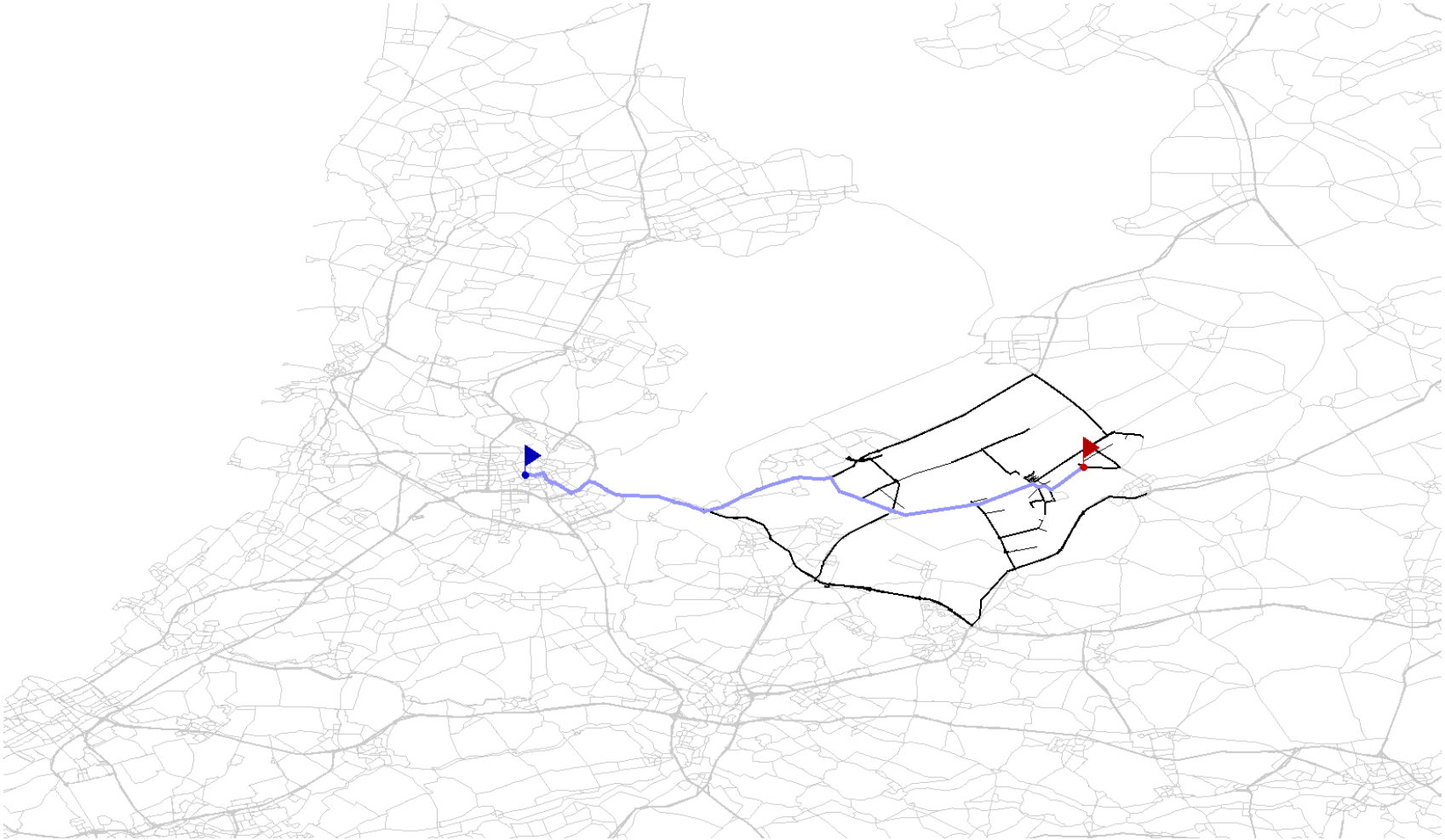
- bestimme gemeinsamen Level l von u und t
- werte Flaggen auf dem Level l aus

Korrektheit: siehe Übung

Suchraum



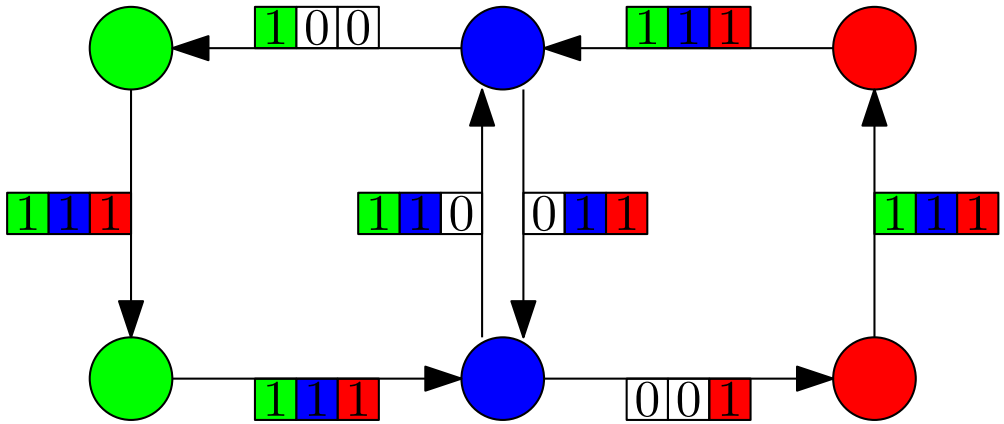
Suchraum



Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



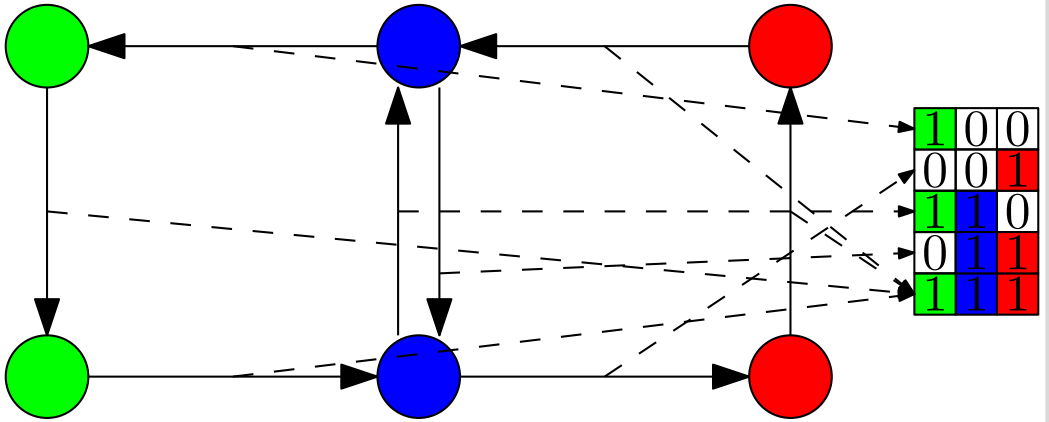
Idee:

- speicher 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



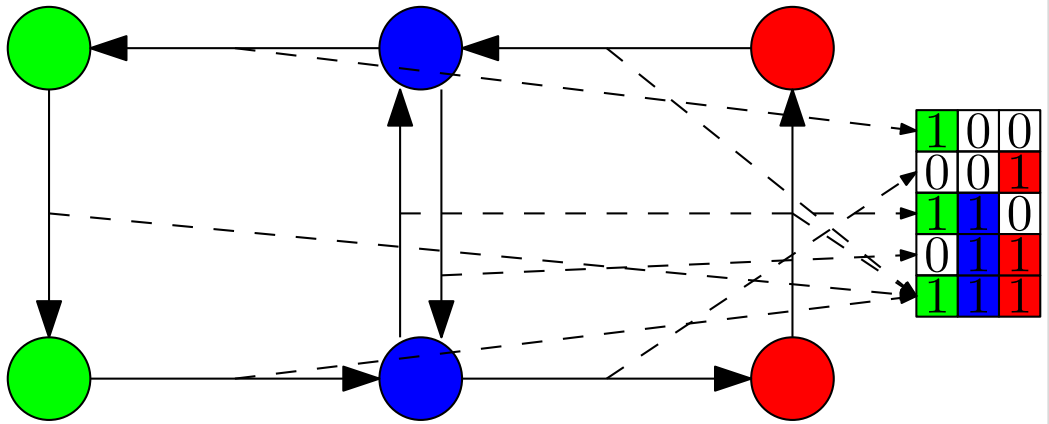
Idee:

- speicher 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)



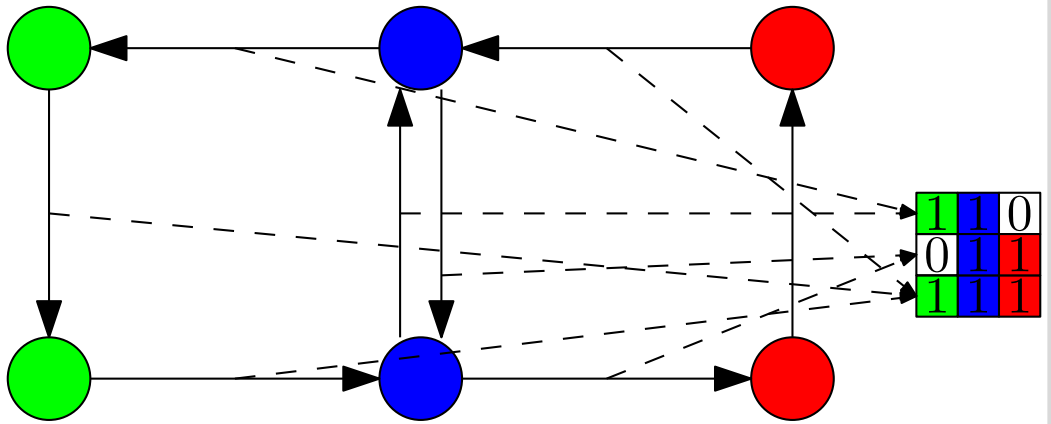
Idee:

- 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

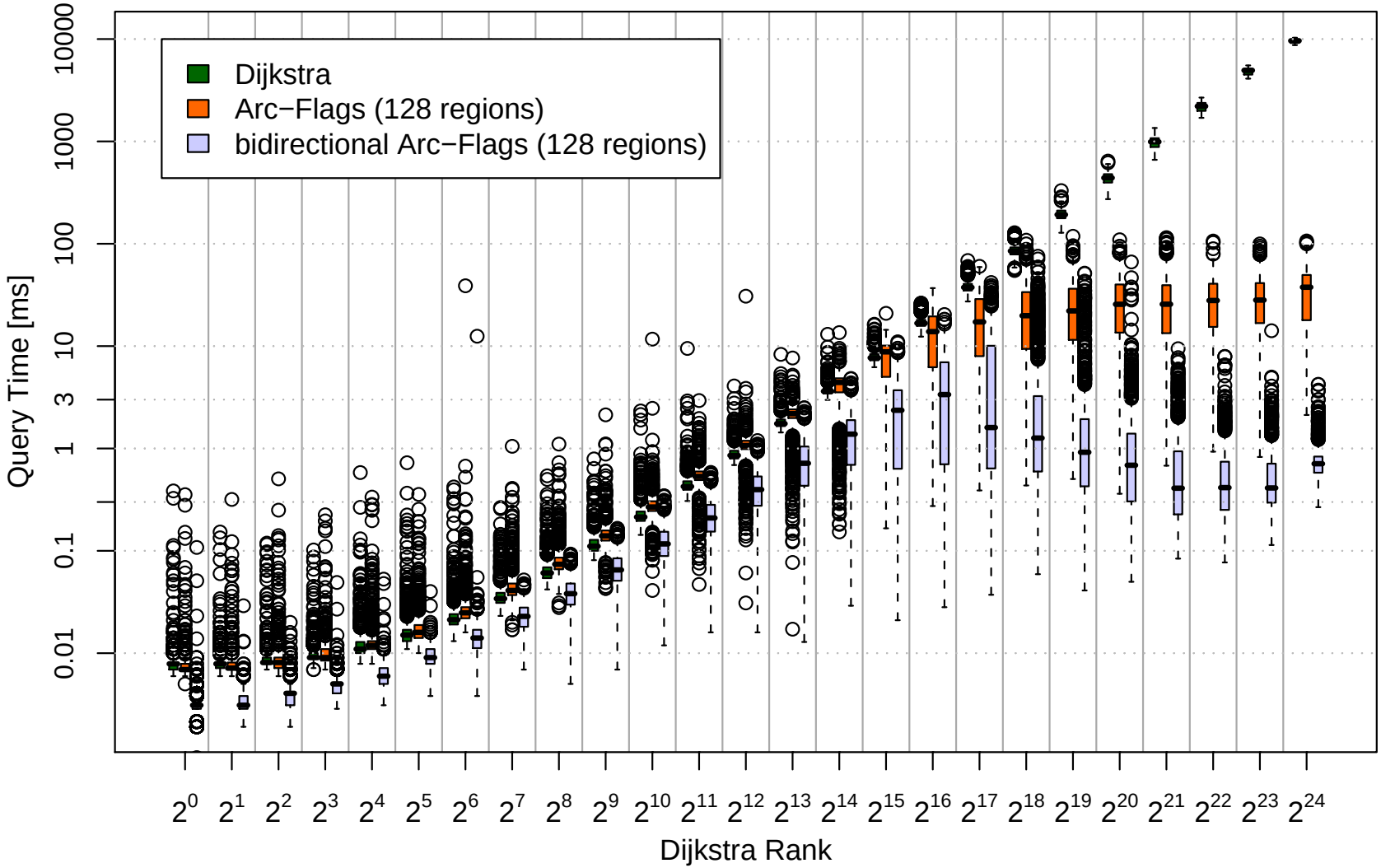
Zufallsanfragen: Anzahl Regionen

	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

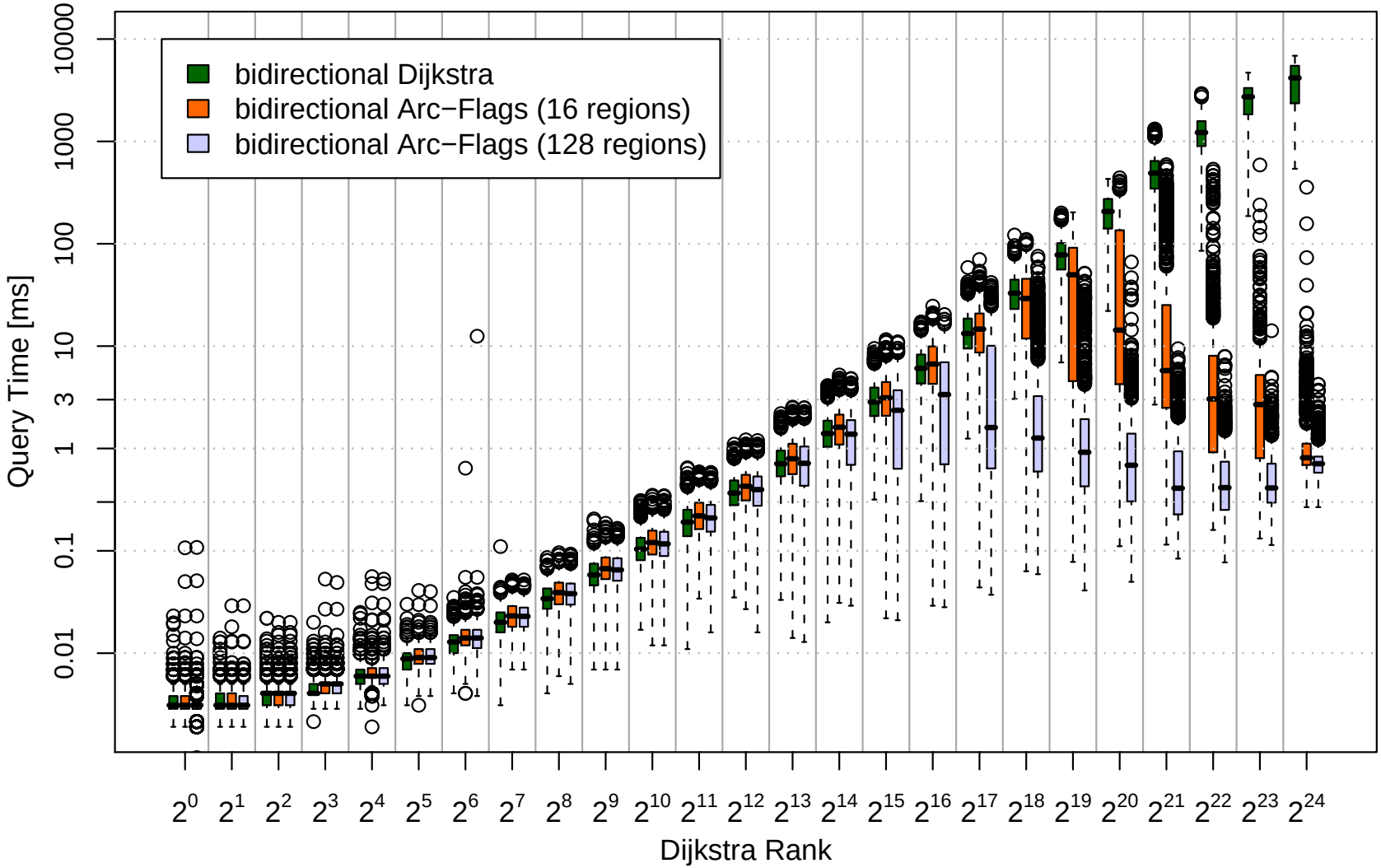
Beobachtungen:

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

Lokale Anfragen Arc-Flags I



Lokale Anfragen Arc-Flags I

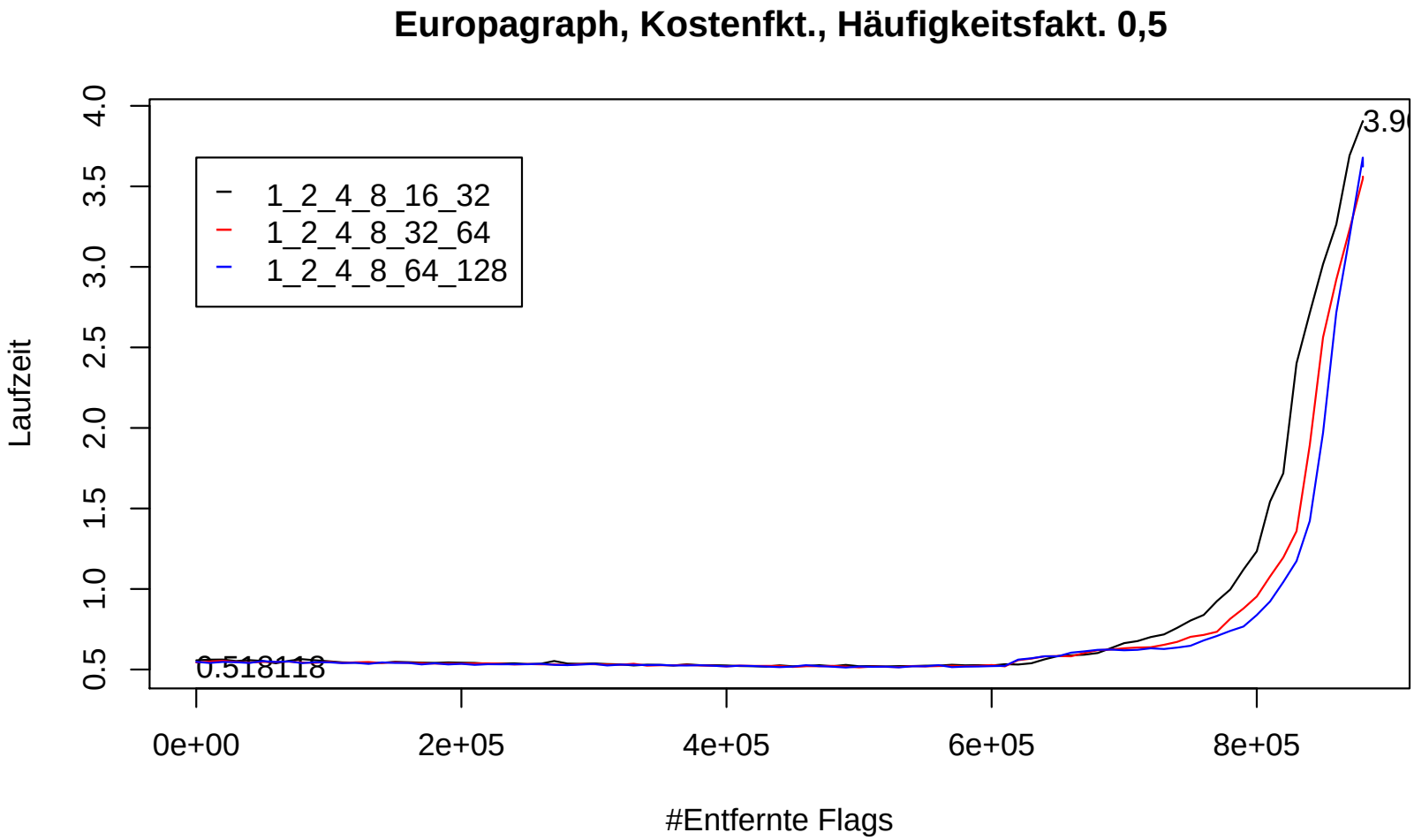


Beobachtungen

- 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

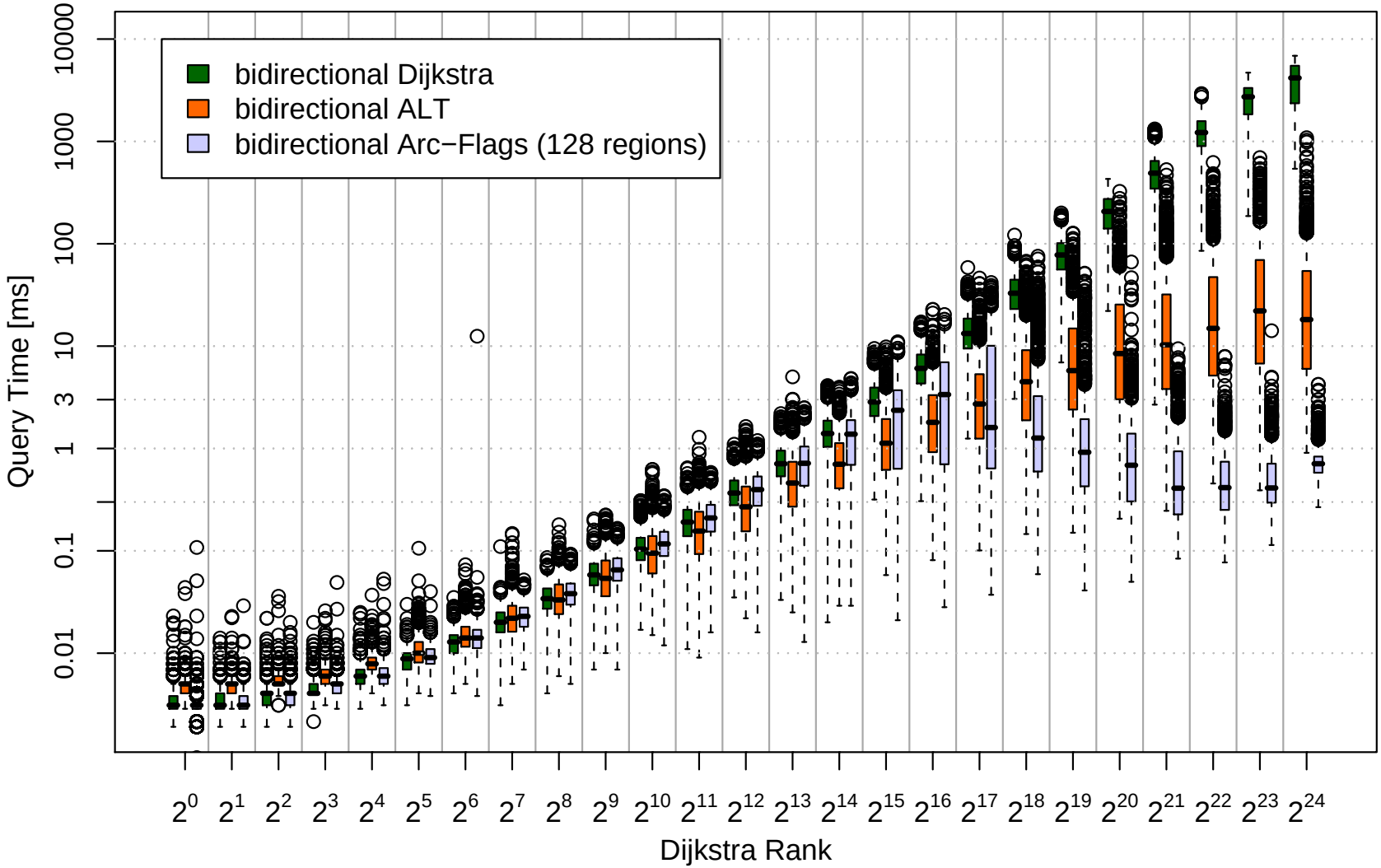
Übersicht: bisherige Techniken

	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

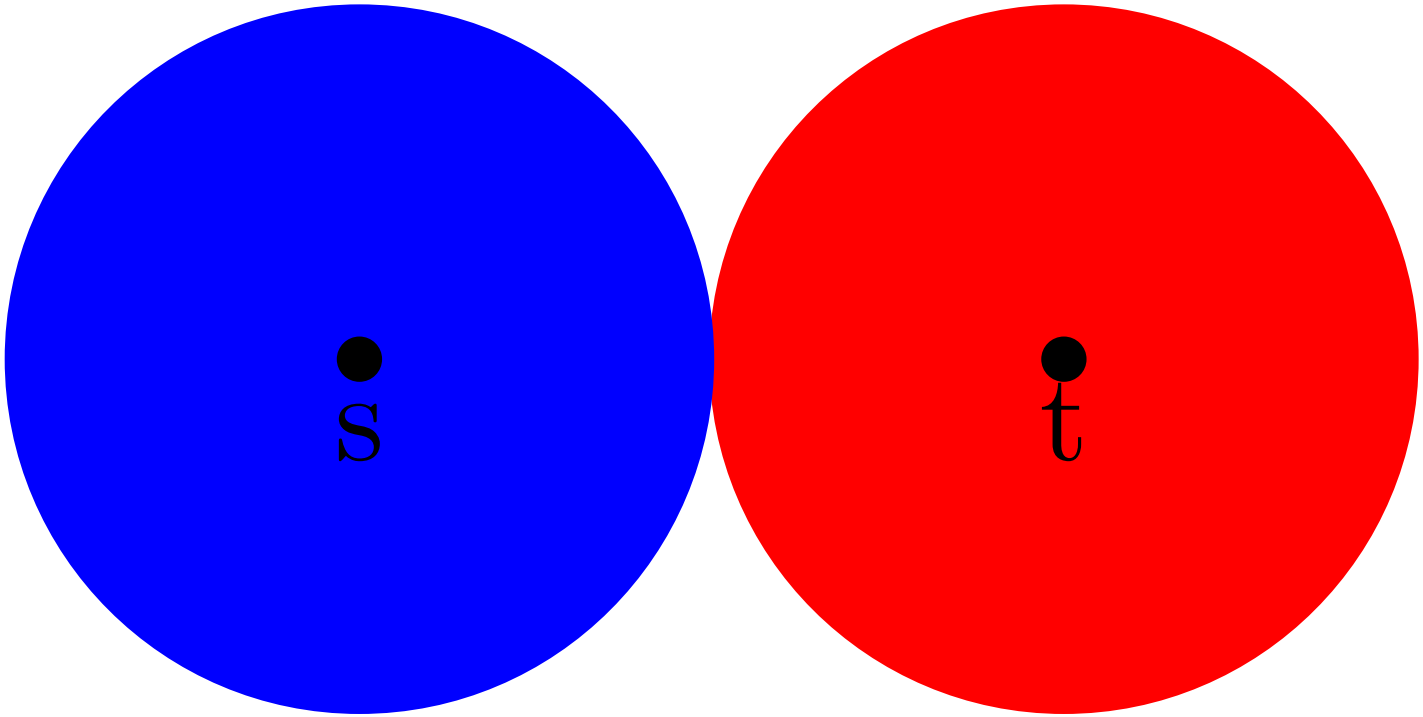
nicht praktikabel

Zusammenfassung

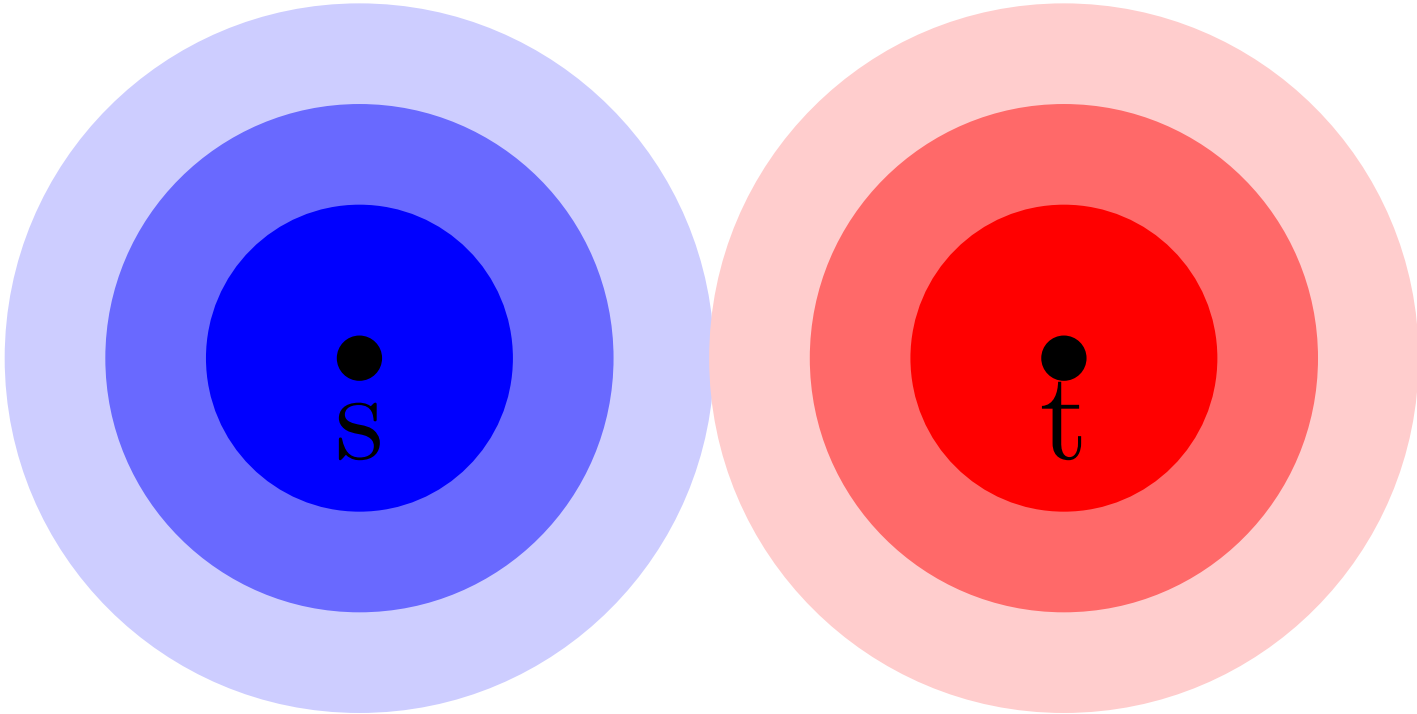
Arc-Flags

- invertiert Geometrische Container
- teile Graphen in k Regionen
- Flaggen zeigen an, ob Kante wichtig für Zielregion ist
- einfacher Anfrage-Algorithmus
- Vorberechnung
  - basiert nicht auf APSP
  - Dijkstra Baum von jedem Randknoten
  - manche Flaggen können automatisch gesetzt werden
  - daher schneller als bei Geometrischen Container
- bidirektional und multi-level Erweiterungen
- Beschleunigung von bis zu 7000
- nahe Anfragen nicht schneller als Dijkstra
- Vorberechnung dauert allerdings immer noch ca. 1 Tag

Hierarchische Techniken



Hierarchische Techniken



Ideensammlung

Wie Suche hierarchisch machen?

- identifiziere wichtige Knoten mit Zentralitätsmaß
- überspringe unwichtige Teile des Graphen

Heute: ersteres

Zentralitätsmaße

Zentralitätsmaße bewerten Wichtigkeit eines Knoten oder einer Kante

Beispiele:

- degree centrality

$$C_D(v) = \frac{\deg(v)}{n-1}$$

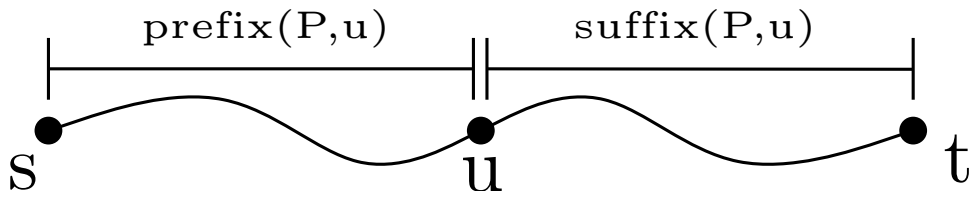
- betweenness centrality

$$C_B(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Idee:

- berechne Zentralitätsmaß
- benutze ZM zum prunen von Knoten oder Kanten

Reach



Definition:

- sei $P = \langle s, \ldots, u, \ldots, t \rangle$ Pfad durch u
- dann Reach von u bezüglich P:

$$r_P(u) := \min\{\text{len}(P_{su}), \text{len}(P_{ut})\}$$

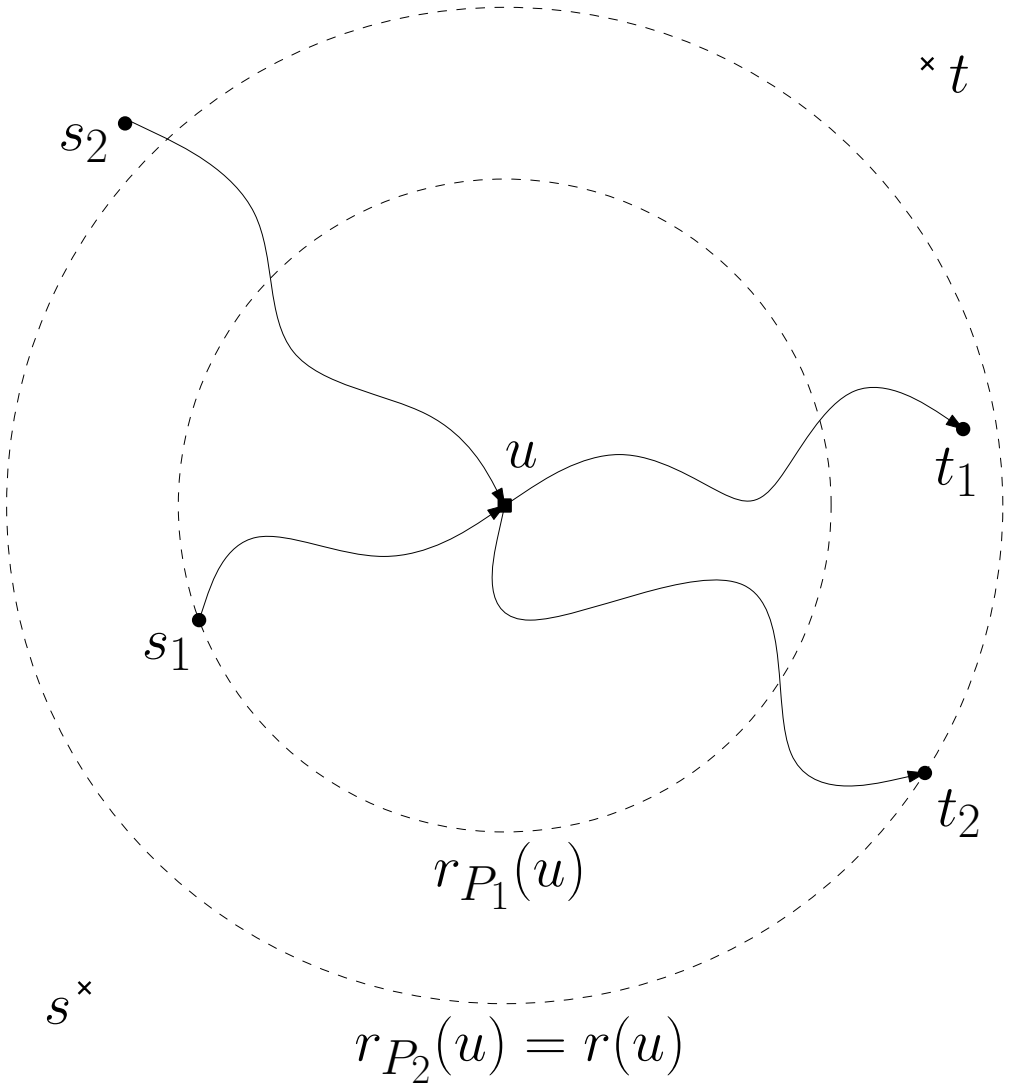
- Reach von u:
  Maximum seiner Reachwerte bezüglich aller kürzesten Pfade durch u:

$$r(u) := \max\{r_P(u) \mid P \text{ kürzester Weg mit } u \in P\}$$

Reach-Pruning

somit:

- Reach $r(u)$ von u gibt Suffix oder Prefix des längsten kürzesten Weges durch u
- wenn für u während Query $r(u) < d(s,u)$ und $r(u) < d(u,t)$ gilt, kann u geprunt werden



Reach Dijkstra - Pseudocode

ReachDijkstra($G = (V, E)$, s, t)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      if r(u) < d[u] and r(u) < d(u, t) then continue
6      forall edges e = (u, v) ∈ E do
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])
```

Probleme

Probleme:

- Abfrage $r(u) < d(u,t)$

Lösung:

- Knotenpotential $\pi(u)$ gibt untere Schranke zum Ziel wenn $\pi(t) = 0$
- benutze A* für Abschätzung (euklidisch oder Landmarken)
- gut kombinierbar mit A*

Probleme

Probleme:

- Abfrage $r(u) < d(u,t)$

Lösung:

- Knotenpotential $\pi(u)$ gibt untere Schranke zum Ziel wenn $\pi(t) = 0$
- benutze A* für Abschätzung (euklidisch oder Landmarken)
- gut kombinierbar mit A*

Potentiale durch Landmarken

Idee:

- wähle Landmarken L aus V (≈ 16)
- berechne Distanzen von und zu allen Landmarken
- dann gilt:

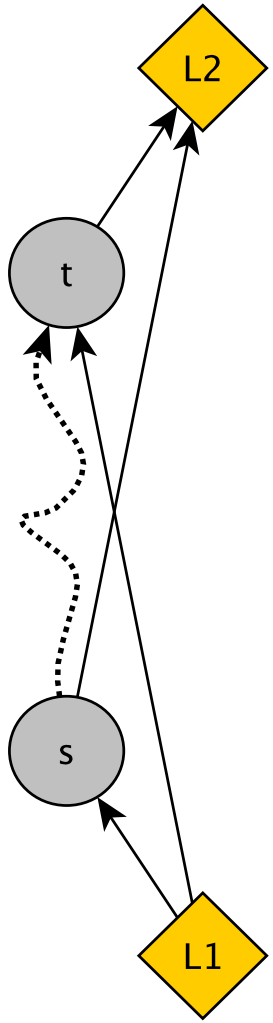
$$\begin{aligned} d(u,t) &\geq d(L_1,t) - d(L_1,u) \\ d(u,t) &\geq d(u,L_2) - d(t,L_2) \end{aligned}$$

  für alle $u \in V$.

- somit ist

$$\pi(u) = \max_{\ell \in L} \{\max\{d(\ell,t) - d(\ell,u), d(u,\ell) - d(t,\ell)\}\}$$

  ein gültiges Potential



A* + Reach

A*Reach($G = (V, E)$, s, t)

```
1  d[s] = 0
2  Q.clear(), Q.add(s, 0)
3  while !Q.empty() do
4      u ← Q.deleteMin()
5      if r(u) < d[u] and r(u) < π(u) then continue
6      forall edges e = (u, v) ∈ E do
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] + π(v))
10             else Q.insert(v, d[v] + π(v))
```

Nachteil A*Reach

Problem:

- Potentiale nicht verfügbar
- Potentiale können schlechte Abschätzung sein

$\Rightarrow$ schwaches Pruning

Lösung:

- benutze bidirektionalen Anfragealgorithmus
- zwei Ansätze:
  - self-bounding
  - distance-bounding

Ende

Nächste Vorlesung:
Montag, 10. Mai, 14:00 Uhr

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*

Username: `routePlanning` Passwort: `ss10`

Ende

Literatur (Reach, RE, REAL):

- Ronald J. Gutman:
  Reach-Based Routing: A New Approach to Shortest Path Algorithms Optimized for Road Networks
  In: *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX'04), 2004 pages 100–111.*
- Andrew V. Goldberg and Haim Kaplan and Renato F. Werneck:
  Reach for A*: Shortest Path Algorithms with Preprocessing
  In: *Shortest Paths: Ninth DIMACS Implementation Challenge, 2009.*

Username: `routePlanning` Passwort: `ss10`