Karlsruher Institut für Technologie

Algorithmen für Routenplanung

4. Vorlesung, Sommersemester 2019

Tim Zeitz | 8. Mai 2019

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER

Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken

- Arc-Flags
- SHARC

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ideensammlung

Wie Suche zielgerichtet machen?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Ideensammlung

Wie Suche zielgerichtet machen?

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

Jetzt: letzteres

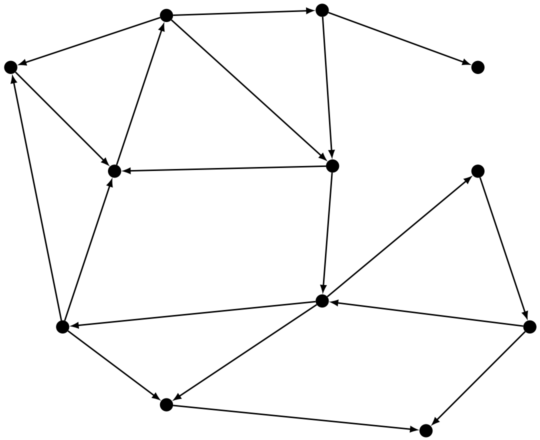
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags: Idee

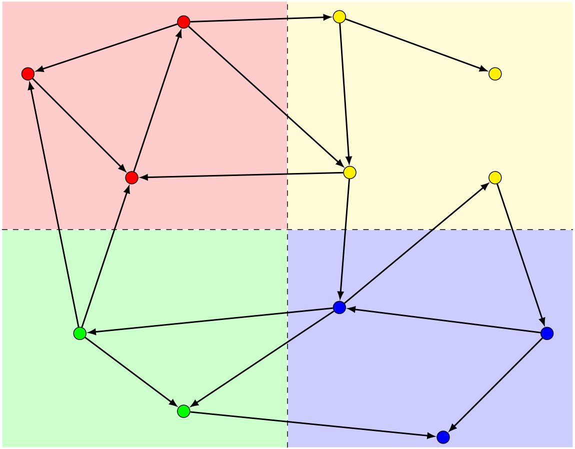
- partitioniere Graphen in k Zellen
- hänge Label mit k Bits an jede Kante
- gibt an, ob Kante e für Ziele in Zelle T benötigt wird



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags: Idee

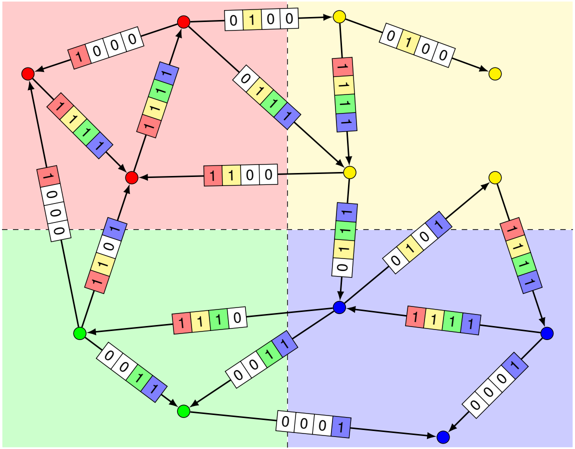
- partitioniere Graphen in k Zellen
- hänge Label mit k Bits an jede Kante
- gibt an, ob Kante e für Ziele in Zelle T benötigt wird



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags: Idee

- partitioniere Graphen in k Zellen
- hänge Label mit k Bits an jede Kante
- gibt an, ob Kante e für Ziele in Zelle T benötigt wird



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Anfragen

Karlsruher Institut für Technologie

$\text{ARC-FLAG-DIJKSTRA}(G = (V, E), s, t, \text{Arc-Flags AF}_{.}(\cdot))$

$d[s] = 0$
$Q.clear(), Q.add(s, 0)$
while $!Q.empty()$ **do**
  $u \leftarrow Q.deleteMin()$
  forall *edges* $e = (u, v) \in E$ **do**
    if $d[u] + \text{len}(e) < d[v]$ *and* $\text{AF}_T(e) = true$ **then**
      $d[v] \leftarrow d[u] + \text{len}(e)$
      if $v \in Q$ **then** $Q.decreaseKey(v, d[v])$
      else $Q.insert(v, d[v])$

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags Vorberechnung

Vorberechnung

Zwei Schritte:

- Partitionierung
- setze korrekte Flaggen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

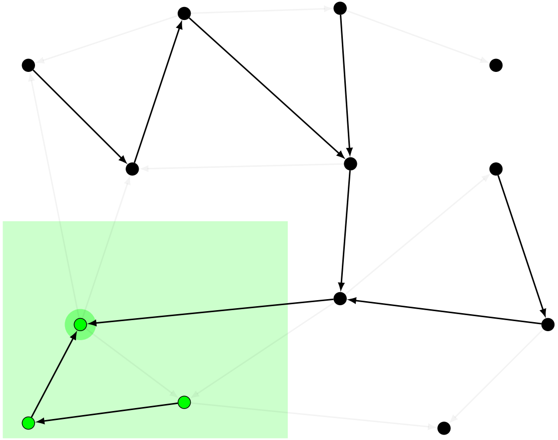
Zwei Schritte:

- Partitionierung
- setze korrekte Flaggen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

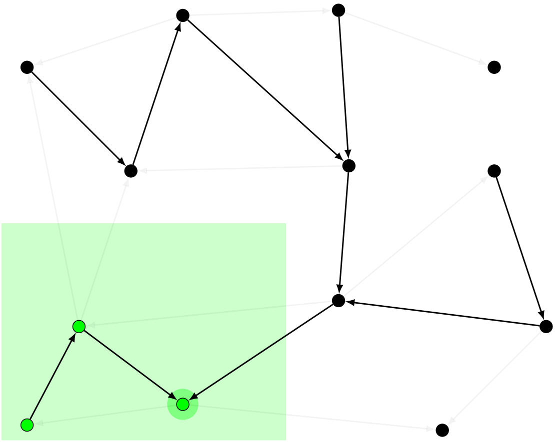
Flaggenberechnung – Naiv

- von jedem Knoten: konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- Berechnung mit all-pairs shortest paths (APSP)



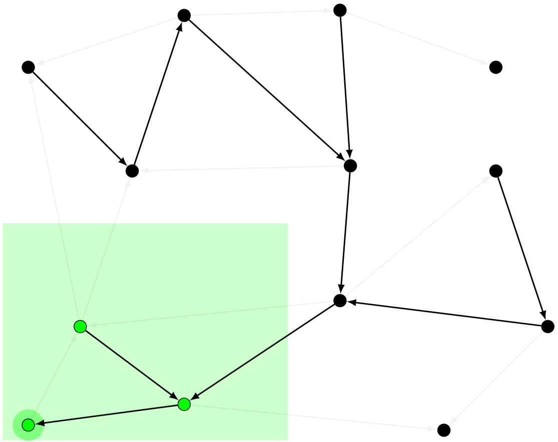
Flaggenberechnung – Naiv

- von jedem Knoten: konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- Berechnung mit all-pairs shortest paths (APSP)



Flaggenberechnung – Naiv

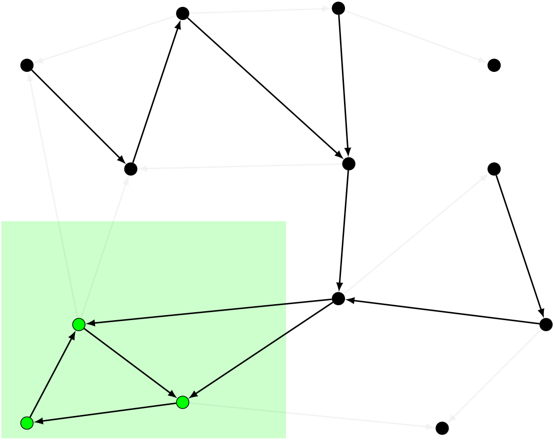
- von jedem Knoten: konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- Berechnung mit all-pairs shortest paths (APSP)



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

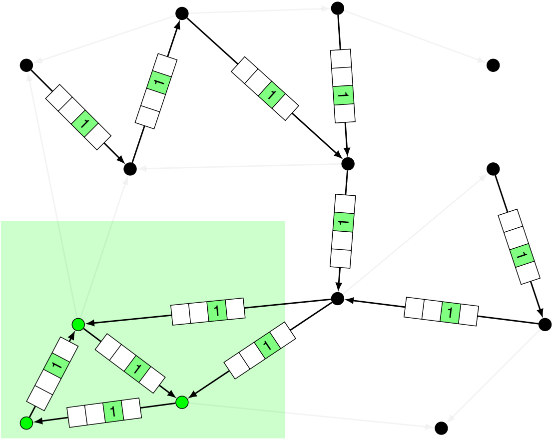
Flaggenberechnung – Naiv

- von jedem Knoten: konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- Berechnung mit all-pairs shortest paths (APSP)



Flaggenberechnung – Naiv

- von jedem Knoten: konstruiere Rückwärts KW-Baum
- setze Flagge der Region der Wurzel für jede Baumkante
- Berechnung mit all-pairs shortest paths (APSP)



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Flaggenberechnung – Randknoten

Karlsruher Institut für Technologie

Beobachtung: Man muss durch Randknoten in die Zelle

Flaggenberechnung – Randknoten

Beobachtung: Man muss durch Randknoten in die Zelle

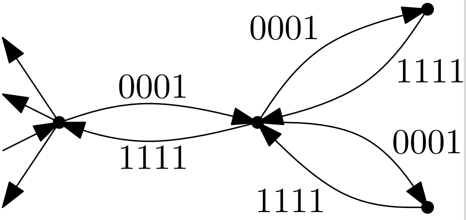
- setze Intra-Zellen Kanten auf `true`
- einen Rückwärts Dijkstra-Baum pro Randknoten b
- setze Flagge $AF_{region(b)}(e) = \texttt{true}$ wenn e Baumkante des Baums von b ist

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Flaggenberechnung: Angehängte Bäume

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: Entferne Bäume vor Arcs-Flag-Berechnung aus Graph
- Knotenzahl verringert sich um ein Drittel

Anmerkung: Alle Knoten eines angehangenen Baumes müssen zur gleichen Zelle wie dessen Wurzel gehören.

Übersicht Vorberechungsarten

In aufsteigender Güte:

- Kürzeste-Wege-Baum von jedem Knoten
- Kürzeste-Wege-Baum von jedem Randknoten
- „Zentralisierte Bäume“
  (nicht Stoff der Vorlesung, ca. Faktor 4 schneller)
- Beschleunigte KW-Baum-Berechnung
  (PHAST, kommt in späterer VL)

Vorberechnung

Zwei Schritte:

- Partitionierung
- setze korrekte Flaggen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Zwei Schritte:

- Partitionierung
- setze korrekte Flaggen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Partitionierung

Wie sollte der Graph partitioniert werden?

- Arc-Flags Algorithmus ist korrekt für beliebige Partition.
- wie sieht eine “gute” Partition aus?
- Ziel: Möglichst schnelle Anfragen.
- formal?

Definiere ein Maß, um die Güte einer Partition zu bewerten.

Annahmen:

- Kanten sind kürzeste Wege zwischen ihren Endknoten.
- kürzeste Wege sind eindeutig.

Damit bestimmt die Partition eindeutig, welche Flags gesetzt werden.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Optimierungsproblem

Karlsruher Institut für Technologie

Definition (Suchraumgröße)

Sei $G = (V, E)$ ein Graph mit Gewichtsfunktion $\mathsf{len}\colon E \to \mathbb{R}_{\geq 0}$, $\mathcal{C}$ eine Partition von G, und $s, t \in V$ zwei Knoten. Dann ist

- $\mathsf{S}(s, t) :=$ Anzahl abgearbeiteter Knoten einer s-t-Anfrage, mit Arc-Flags induziert durch die Partition $\mathcal{C}$.
- $\mathsf{S}_{\max} := \max_{s,t \in V} \mathsf{S}(s, t)$.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Optimierungsproblem

Definition (Suchraumgröße)

Sei $G = (V, E)$ ein Graph mit Gewichtsfunktion $\mathrm{len}\colon E \to \mathbb{R}_{\geq 0}$, $\mathcal{C}$ eine Partition von G, und $s, t \in V$ zwei Knoten. Dann ist

- $\mathsf{S}(s, t) :=$ Anzahl abgearbeiteter Knoten einer s-t-Anfrage, mit Arc-Flags induziert durch die Partition $\mathcal{C}$.
- $\mathsf{S}_{\max} := \max_{s,t \in V} \mathsf{S}(s, t)$.

MinWorstCasePartition

Gegeben: Graph $G = (V, E, \mathrm{len})$, natürliche Zahl k.
Gesucht: Partition $\mathcal{C}$, $|\mathcal{C}| \leq k$, so dass $\mathsf{S}_{\max}$ minimiert wird.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Optimierungsproblem

Definition (Suchraumgröße)

Sei $G = (V, E)$ ein Graph mit Gewichtsfunktion $\mathrm{len}\colon E \to \mathbb{R}_{\geq 0}$, $\mathcal{C}$ eine Partition von G, und $s, t \in V$ zwei Knoten. Dann ist

- $\mathsf{S}(s, t) :=$ Anzahl abgearbeiteter Knoten einer s-t-Anfrage, mit Arc-Flags induziert durch die Partition $\mathcal{C}$.
- $\mathsf{S}_{\max} := \max_{s,t \in V} \mathsf{S}(s, t)$.

MinWorstCasePartition

Gegeben: Graph $G = (V, E, \mathrm{len})$, natürliche Zahl k.
Gesucht: Partition $\mathcal{C}$, $|\mathcal{C}| \leq k$, so dass $\mathsf{S}_{\max}$ minimiert wird.

Anmerkung:
Ohne festen Parameter k ist die optimale Lösung immer eine Partition $\mathcal{C} = \{\{v\} \mid v \in V\}$ — dies führt aber zu Speicherverbrauch $\mathcal{O}(nm)$.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Eigenschaften von $\mathrm{S}_{\max}$

Für Graph $G = (V, E)$, Gewichtsfunktion len, Partition $\mathcal{C}$ von G:

Lemma

Für jedes $s \in V$, $T \in \mathcal{C}$ gibt einen Knoten $t \in V$, so dass eine s-t-Anfrage alle (erreichbaren) Knoten aus T abarbeitet (auch bei aktiviertem Stoppkriterium).

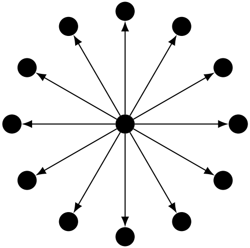
Beweis:
Annahme: Eindeutiges tie breaking für priority queue

- für festes s werden alle Knoten aus T in einer festen Reihenfolge abgearbeitet
- setze $t :=$ letzter Knoten in dieser Reihenfolge

$S_{\max}$ für Sterne

Erster Schritt: Sterne, Kanten von der Wurzel weggerichtet.

Wie groß ist $S_{\max} := \max_{s,t \in V}$ (für beliebige Partition)?



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

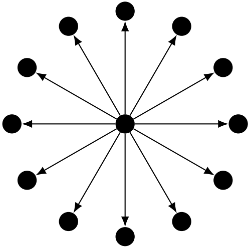
$\mathrm{S}_{\max}$ für Sterne

Erster Schritt: Sterne, Kanten von der Wurzel weggerichtet.

Wie groß ist $\mathrm{S}_{\max} := \max_{s,t \in V}$ (für beliebige Partition)?

Fallunterscheidung:

1. Knoten s ist Blatt: $S(s,t) = 1$ für beliebige t.
2. Knoten s ist die Wurzel: für jede Zelle T gibt es eine s-t-Query, so dass alle Knoten aus T abgearbeitet werden.



$\Rightarrow \mathrm{S}_{\max} = 1 + \max_{T \in \mathcal{C}} |T \setminus \{r\}|$ (für Stern mit Wurzel r).
$\Rightarrow$ Minimal wenn Blätter gleichmäßig auf Zellen der Partition verteilt.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

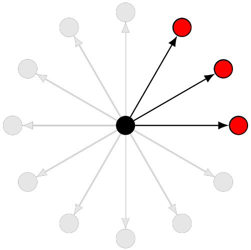
$S_{\max}$ für Sterne

Erster Schritt: Sterne, Kanten von der Wurzel weggerichtet.

Wie groß ist $S_{\max} := \max_{s,t \in V}$ (für beliebige Partition)?

Fallunterscheidung:

1. Knoten s ist Blatt: $S(s, t) = 1$ für beliebige t.
2. Knoten s ist die Wurzel: für jede Zelle T gibt es eine s-t-Query, so dass alle Knoten aus T abgearbeitet werden.



$\Rightarrow$ $S_{\max} = 1 + \max_{T \in \mathcal{C}} |T \setminus \{r\}|$ (für Stern mit Wurzel r).
$\Rightarrow$ Minimal wenn Blätter gleichmäßig auf Zellen der Partition verteilt.

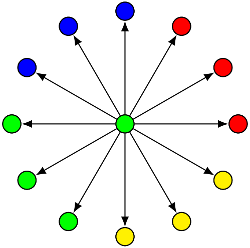
$S_{\max}$ für Sterne

Erster Schritt: Sterne, Kanten von der Wurzel weggerichtet.

Wie groß ist $S_{\max} := \max_{s,t \in V}$ (für beliebige Partition)?

Fallunterscheidung:

1. Knoten s ist Blatt: $S(s,t) = 1$ für beliebige t.
2. Knoten s ist die Wurzel: für jede Zelle T gibt es eine s-t-Query, so dass alle Knoten aus T abgearbeitet werden.



$\Rightarrow S_{\max} = 1 + \max_{T \in \mathcal{C}} |T \setminus \{r\}|$ (für Stern mit Wurzel r).
$\Rightarrow$ Minimal wenn Blätter gleichmäßig auf Zellen der Partition verteilt.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bäume mit Höhe 2

Satz

MinWorstCasePartition ist $\mathcal{NP}$-schwer für gerichtete Bäume mit Höhe 2, sogar wenn alle Kanten Einheitsgewicht haben.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bäume mit Höhe 2

Satz

MinWorstCasePartition ist $\mathcal{NP}$-schwer für gerichtete Bäume mit Höhe 2, sogar wenn alle Kanten Einheitsgewicht haben.

Betrachte Entscheidungsproblem:

MinWorstCasePartition-Dec

Gegeben: Graph $G = (V, E, \text{len})$, natürliche Zahlen k, S^*.
Gesucht: Existiert $\mathcal{C}$, $|\mathcal{C}| \leq k$, so dass $\text{S}_{\max} \leq S^*$?

Beweis

Reduktion vom stark $\mathcal{NP}$-schweren Entscheidungsproblem 3-PARTITION.

3-PARTITION

Gegeben:

- Menge $S = \{s_1, \ldots, s_{3m}\}$,
- natürliche Zahl B,
- Gewichtsfunktion $\omega \colon S \to \{1, \ldots, B\}$, so dass $\sum_{i=1}^{3m} \omega(s_i) = mB$.

Gesucht: Kann S in m Mengen S_i partitioniert werden, so dass $|S_i| = 3$ für alle i und $\sum_{s \in S_i} \omega(s) = B$?

Sogar $\mathcal{NP}$-schwer, wenn $\frac{B}{4} < \omega(s_i) < \frac{B}{2}$ für alle s_i.

Beispielinstanz:
$m = 2$, $|S| = 6$, $B = 11$ und Gewichte 3, 3, 3, 4, 4, 5.

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beweis

Reduktion vom stark $\mathcal{NP}$-schweren Entscheidungsproblem 3-PARTITION.

3-PARTITION

Gegeben:

- Menge $S = \{s_1, \dots, s_{3m}\}$,
- natürliche Zahl B,
- Gewichtsfunktion $\omega\colon S \to \{1, \dots, B\}$, so dass $\sum_{i=1}^{3m} \omega(s_i) = mB$.

Gesucht: Kann S in m Mengen S_i partitioniert werden, so dass $|S_i| = 3$ für alle i und $\sum_{s \in S_i} \omega(s) = B$?

Sogar $\mathcal{NP}$-schwer, wenn $\frac{B}{4} < \omega(s_i) < \frac{B}{2}$ für alle s_i.

Beispielinstanz:
$m = 2, |S| = 6, B = 11$ und Gewichte 3, 3, 3, 4, 4, 5.

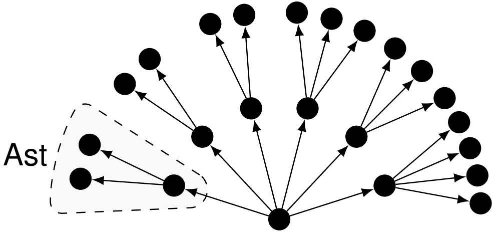
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beweis

Reduktion: Erzeuge Baum mit

- Wurzel r,
- einem Mittelknoten pro Element s_i aus S,
- an jedem Mittelknoten jeweils $\omega(s_i) - 1$ Blätter.

Beispielinstanz: $m = 2$, $|S| = 6$, $B = 11$ und Gewichte 3, 3, 3, 4, 4, 5.



Behauptung: (S, B) ist erfüllbare 3-Partition-Instanz genau dann, wenn $\mathsf{S}_{\max} \leq B + 1$ für Partitionen mit m Zellen.

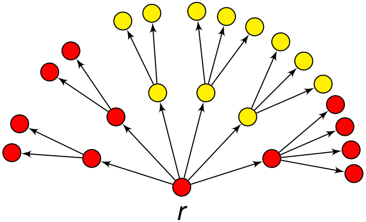
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Beweis

$\Rightarrow$) (S, B) **erfüllbar.**

Zu zeigen: $S_{\max} \leq B + 1$.

- Weise jeder Zelle drei *Äste* zu, die einem Tripel der Lösung entsprechen.
- Im Beispiel: $\{3, 3, 5\}, \{3, 4, 4\}$.



Queries mit Startknoten $s \neq r : S(s, t) < \frac{B}{2}$.

Queries mit Startknoten r : Nur r selbst und Knoten innerhalb der Zielzelle werden abgearbeitet.

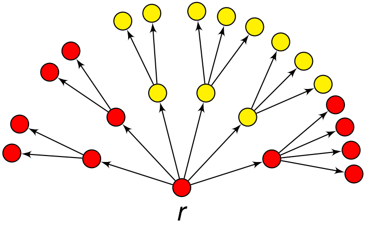
$\Rightarrow S_{\max} \leq B + 1$.

Beweis

$\Leftarrow$) **Es gilt** $\mathsf{S}_{\max} \leq B + 1$.
Zu zeigen: (S, B) erfüllbar.

Wir erhalten eine Lösung für 3-PARTITION, falls ...

- jede Zelle genau B Astknoten enthält.
- jeder Ast genau einer Zelle zugeordnet ist.



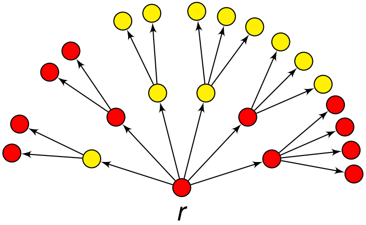
Pro Zelle T gibt es eine Query von r, die T komplett abarbeitet.
$\Rightarrow$ Wegen $\mathsf{S}_{\max} \leq B + 1$ enthält keine Zelle mehr als B Astknoten.

Beweis

$\Leftarrow$) **Es gilt** $\mathrm{S}_{\max} \leq B + 1$.
Zu zeigen: (S, B) erfüllbar.

Wir erhalten eine Lösung für 3-PARTITION, falls ...

- jede Zelle genau B Astknoten enthält.
- jeder Ast genau einer Zelle zugeordnet ist.

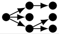
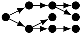


Annahme: ein Ast enthält Knoten aus verschiedenen Zellen.
$\Rightarrow \exists$ Blatt, dessen Zelle T nicht der Zelle des Mittelknoten entspricht.
$\Rightarrow$ Worst-Case Query nach T arbeitet diesen Mittelknoten ab.
$\Rightarrow \mathrm{S}_{\max} > B + 1$. ↯ $\square$

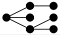
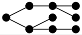
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht: Worst-Case

Gerichtet:

Sterne		$\mathcal{O}(n)$
Bäume ($h \leq 2$)		$\mathcal{NP}$-schwer
Pfade		$\mathcal{O}(n)$
Bäume ($\Delta \leq 3$)		$\mathcal{NP}$-schwer
Kreise		$\mathcal{O}(n)$

Ungerichtet:

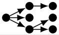
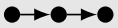
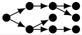
Sterne		$\mathcal{O}(n)$
Bäume ($h \leq 2$)		$\mathcal{NP}$-schwer
Pfade		$\mathcal{O}(n)$
Bäume ($\Delta \leq 3$)		$\mathcal{NP}$-schwer
Kreise		?

- Ähnliche Resultate für andere Graphklassen (beschränkter Grad, ungerichtet).
- Beweisideen in den meisten Fällen ähnlich.
- Problem MinWorstCasePartition ist selbst für stark eingeschränkte Graphklassen $\mathcal{NP}$-schwer.

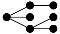
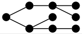
Übersicht: Average-Case

Alternativ: Minimiere erwarteten Suchraum $\mathrm{S}_{\text{avg}} := \sum_{s,t \in V} \mathrm{S}(s,t)/n^2$.

Gerichtet:

Sterne		$\mathcal{O}(n)$
Bäume ($h \leq 2$)		$\mathcal{NP}$-schwer
Pfade		$\mathcal{O}(n)$
Bäume ($\Delta \leq 3$)		?
Kreise		$\mathcal{O}(n)$

Ungerichtet:

Sterne		$\mathcal{O}(n)$
Bäume ($h \leq 2$)		$\mathcal{NP}$-schwer
Pfade		$\mathcal{O}(n)$
Bäume ($\Delta \leq 3$)		?
Kreise		?

- Beweise analog zum Worst Case.
- Auch hier Schwereresultate für stark eingeschränkte Graphen.

$\Rightarrow$ Verwende Heuristiken.

Anmerkung:
Für viele Beschleunigungstechniken, die in der Vorlesung behandelt werden, gibt es vergleichbare Schwereresultate.

Heuristiken

Anforderungen:

- balanciert

mögliche Partitionen:

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

weitere Anforderungen?

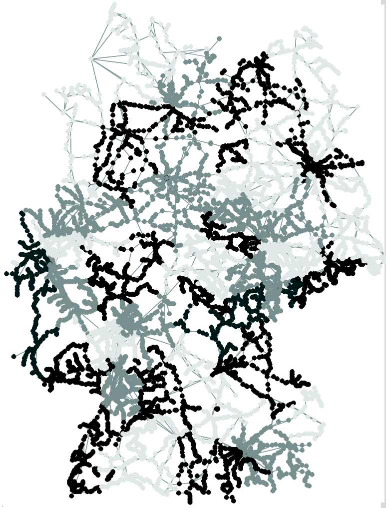
Heuristiken

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Black-Box Partitionierer (METIS, ...):

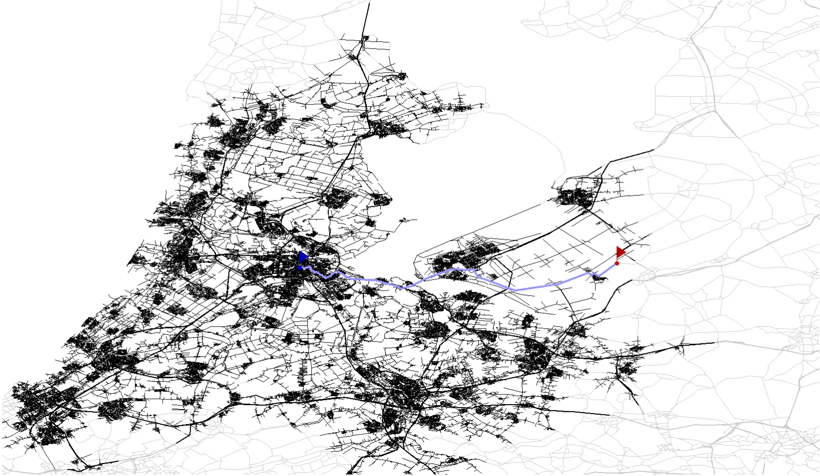
- benutzen keine Einbettung
- oft: teilen rekursiv Graphen in *k* Teile mit kleinem Schnitt
- bringen gute Lösungen für Arc-Flags



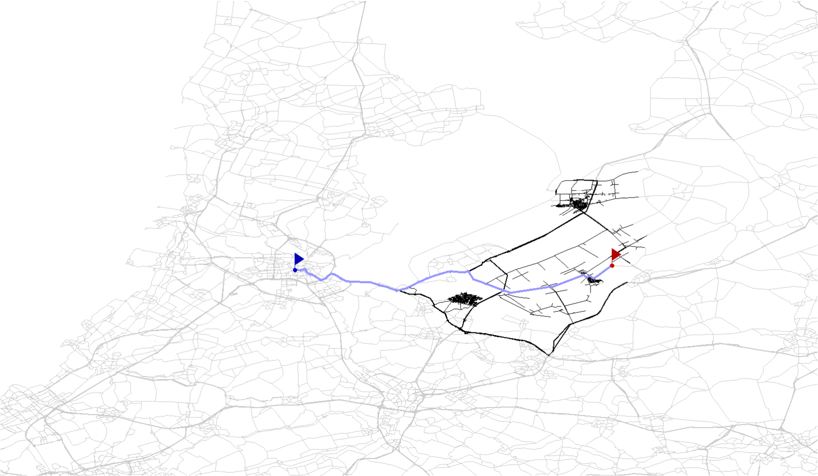
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Arc-Flags Eigenschaften

Suchraum – Dijkstra



Suchraum – Arc-Flags



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

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_j
- $P' = (u_1, \ldots, u_j)$ ist Subpfad von P und somit ein kürzester Weg von u_1 nach u_j
- somit Baumkanten mit gesetzten Flaggen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Diskussion

Vorteile:

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

Nachteile:

- hohe Vorberechnungsdauer
- weitere?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Coning-Effekt

Beobachtung:

- lange Zeit nur eine Kante wichtig
- daher 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

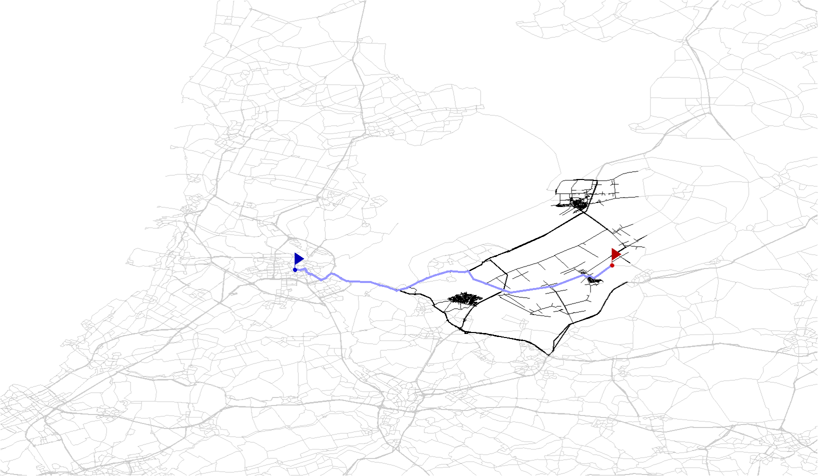
Vorberechung:

- Vorwärts- und Rückwärtsflaggen
- Rückwärtsflaggen analog für eingehende Kanten berechnet
- Vorberechungszeit in gerichteten Graphen: Faktor 2

Anfrage:

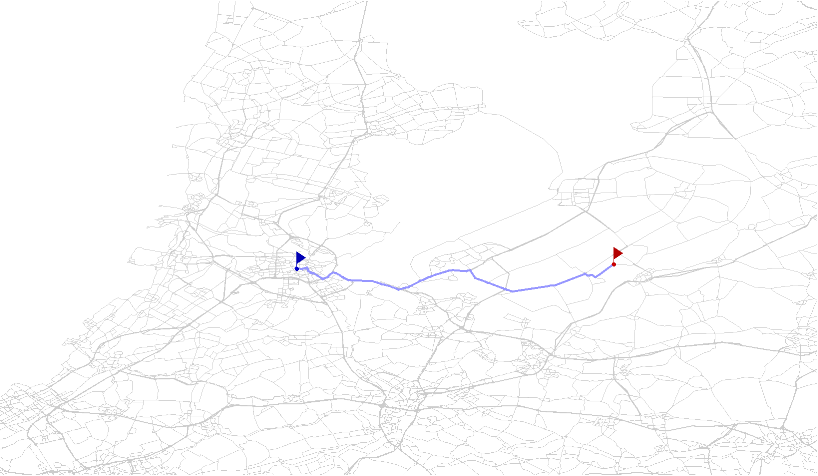
- bidirektional:
  - Vorwärtssuche relaxiert nur Kanten mit Flagge für T
  - Rückwärtssuche nur Kanten mit Flaggen für S
- normales Stopp-Kriterium von bidirektionalem Dijkstra

Suchraum – Arc-Flags



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Suchraum – Bidirektionale Arc-Flags



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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

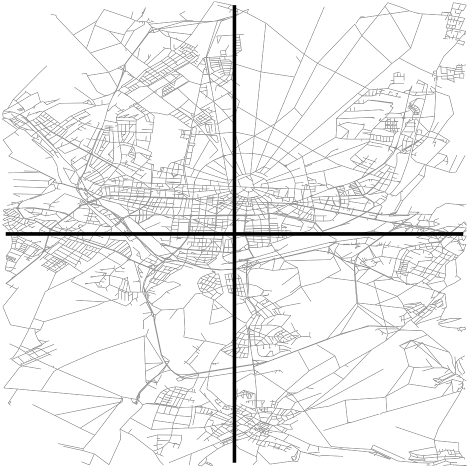
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



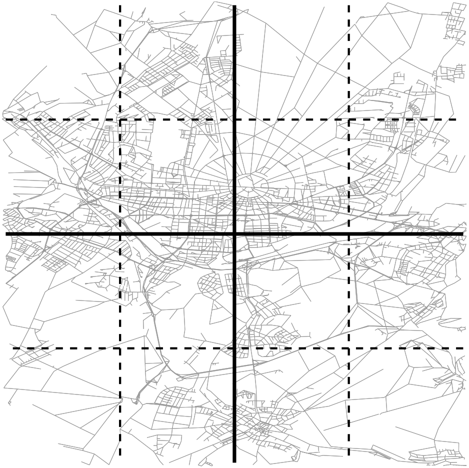
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

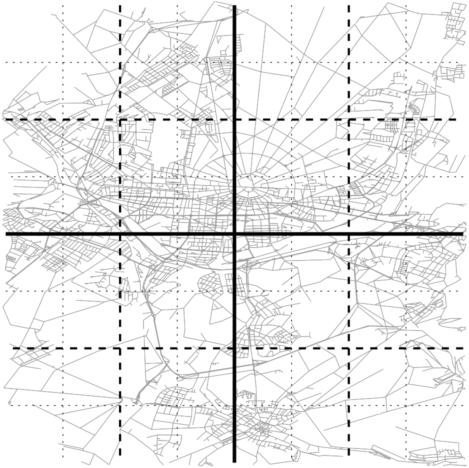
Multi-Level Arc-Flags

Problem:

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

Multi-Level Arc-Flags:

- Multi-Level Partition
- berechne partielle Flaggen

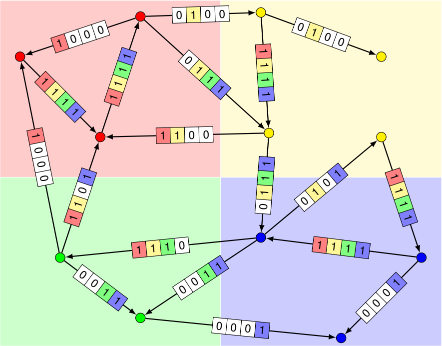


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt

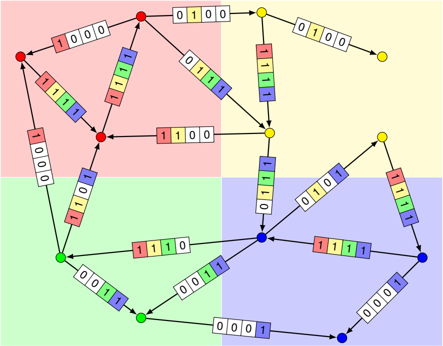


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



Idee:

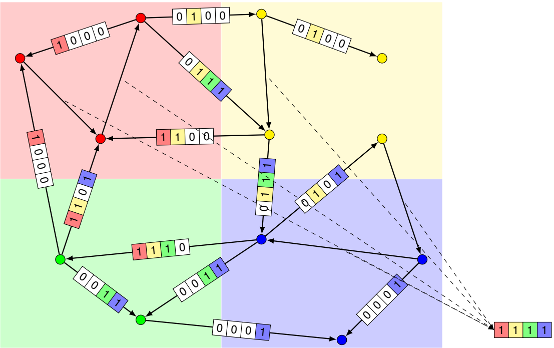
- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



Idee:

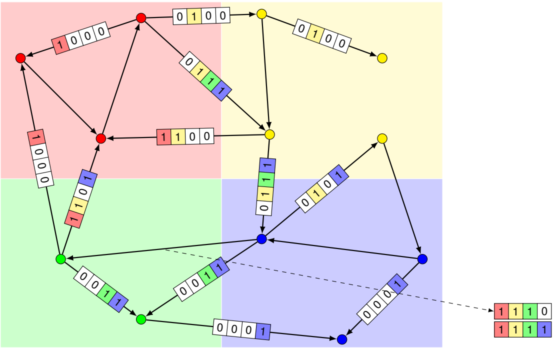
- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt

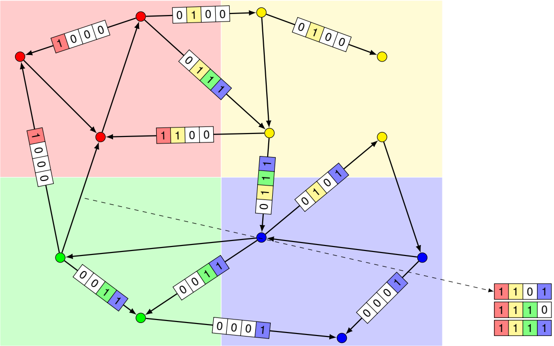


Idee:

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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur



Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt

Idee:

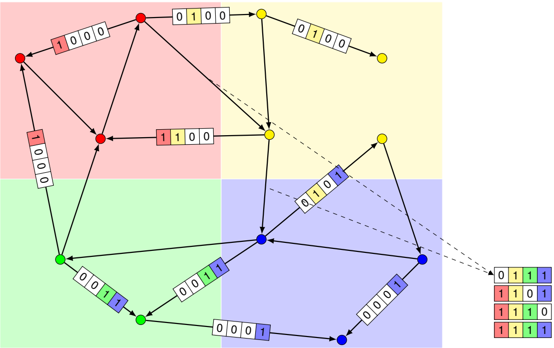
- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



Idee:

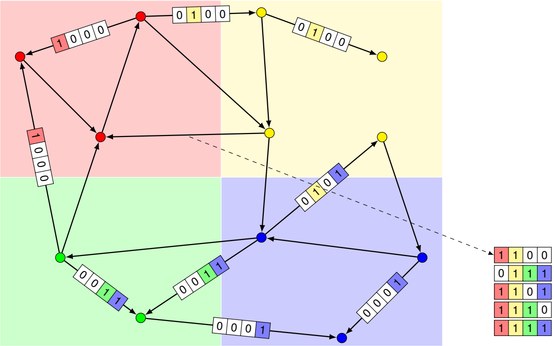
- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



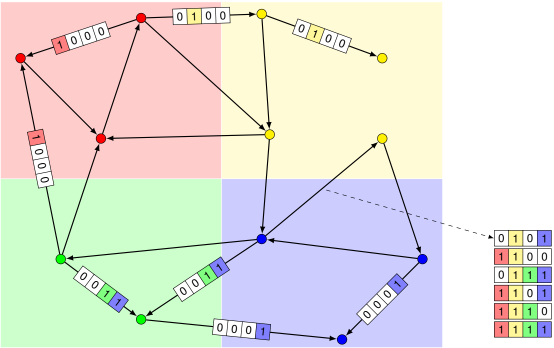
Idee:

- 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:

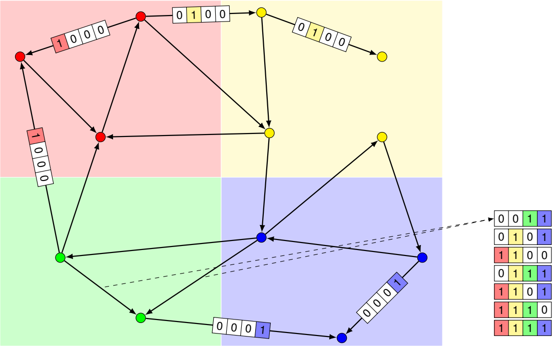
- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Datenstruktur

Karlsruher Institut für Technologie

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



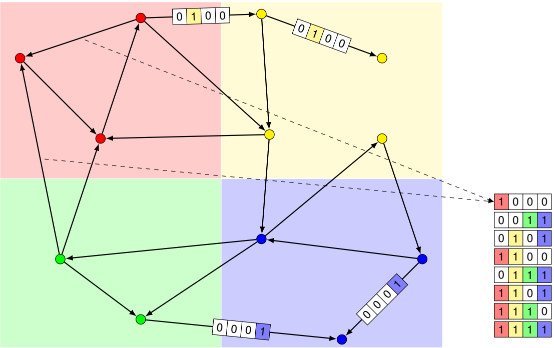
Idee:

- 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



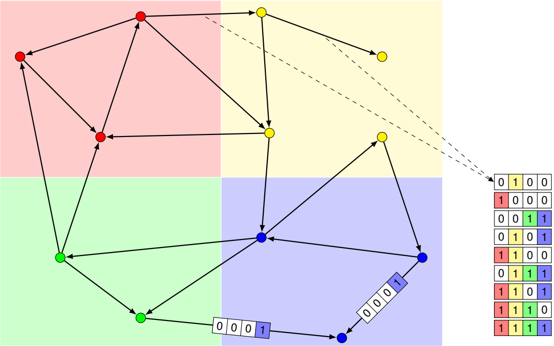
Idee:

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

Datenstruktur

Effizient Flaggen speichern?

- pro Kante eine Flagge
- Beobachtung: Anzahl Kombinationen begrenzt



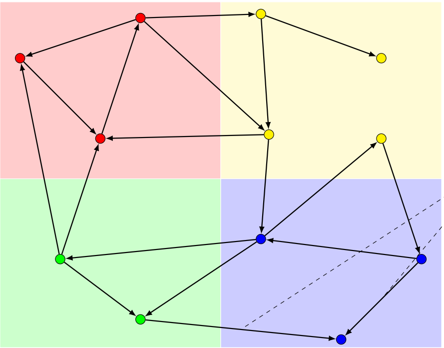
Idee:

- 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



0	0	0	1
0	1	0	0
1	0	0	0
0	0	1	1
0	1	0	1
1	1	0	0
0	1	1	1
1	1	0	1
1	1	1	0
1	1	1	1

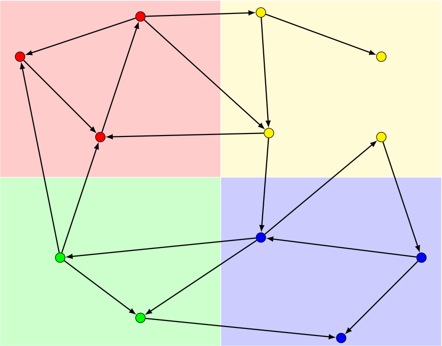
Idee:

- 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



0	0	0	1
0	1	0	0
1	0	0	0
0	0	1	1
0	1	0	1
1	1	0	0
0	1	1	1
1	1	0	1
1	1	1	0
1	1	1	1

Idee:

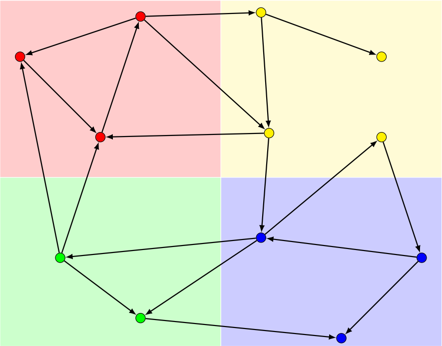
- speichere Flaggen in Matrix
- Zeiger von Kanten auf die Matrix
- verringert Speicherverbrauch um einen Faktor 5

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Flaggenkompression

Beobachtung:

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



0	0	0	1
0	1	0	0
1	0	0	0
0	0	1	1
0	1	0	1
1	1	0	0
0	1	1	1
1	1	0	1
1	1	1	0
1	1	1	1

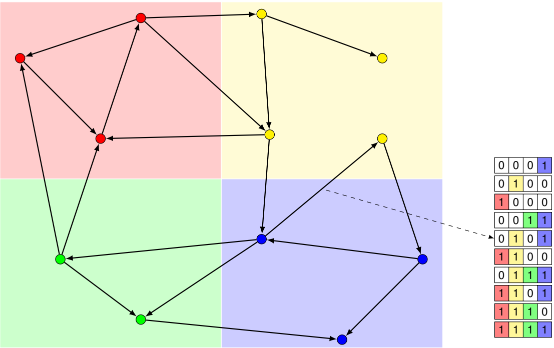
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde "gutes" Mapping

Flaggenkompression

Beobachtung:

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



Idee:

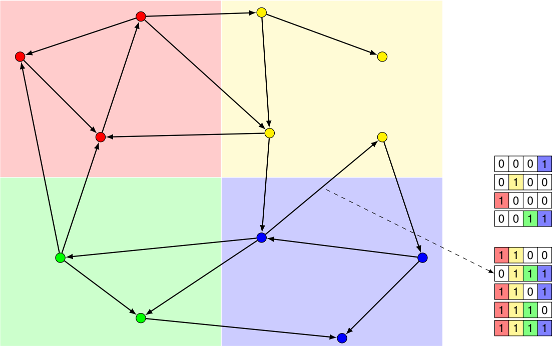
- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Flaggenkompression

Beobachtung:

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



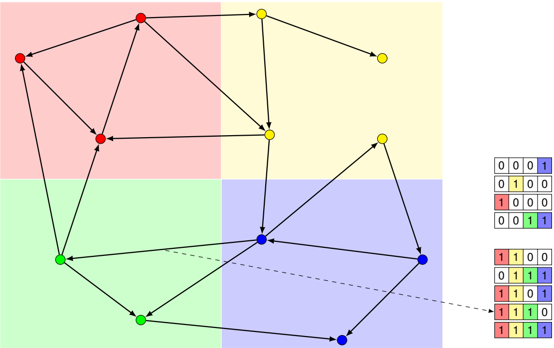
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



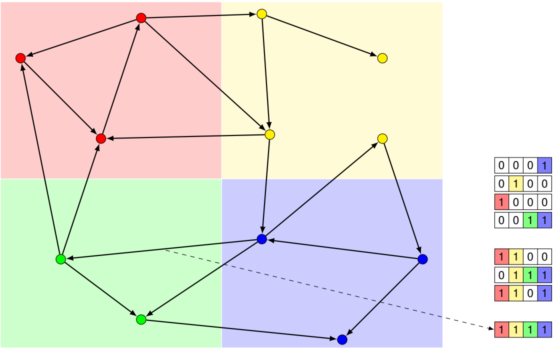
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



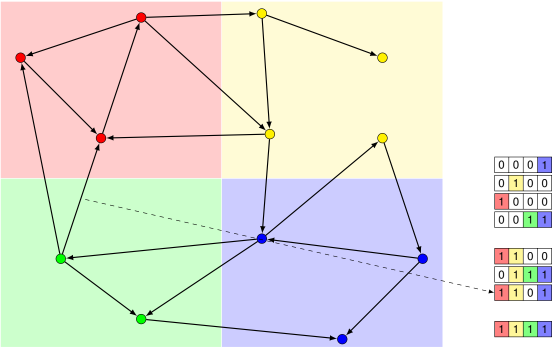
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



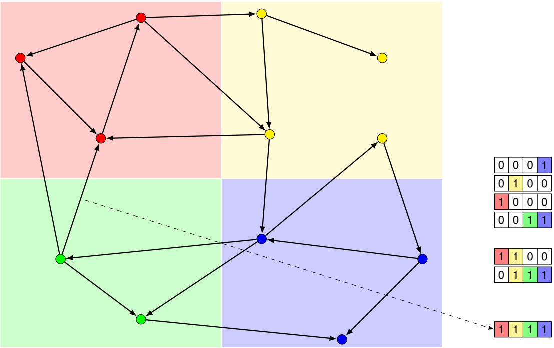
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



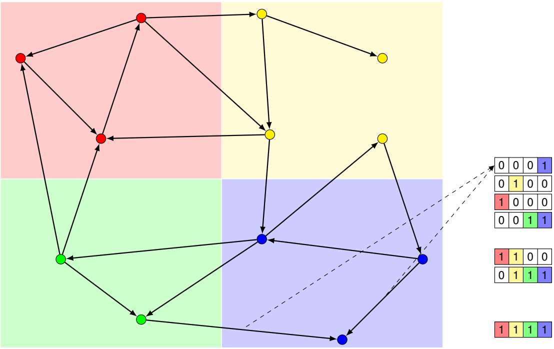
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



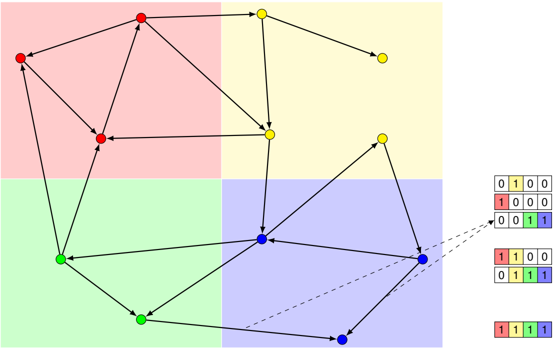
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



Idee:

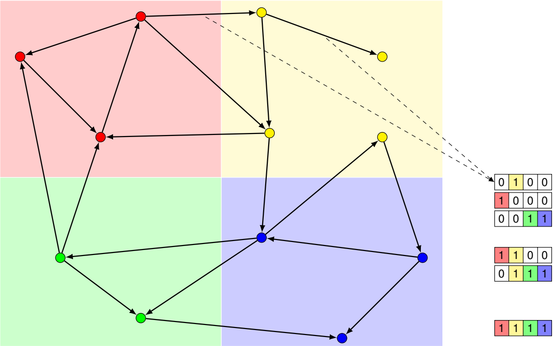
- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Flaggenkompression

Beobachtung:

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



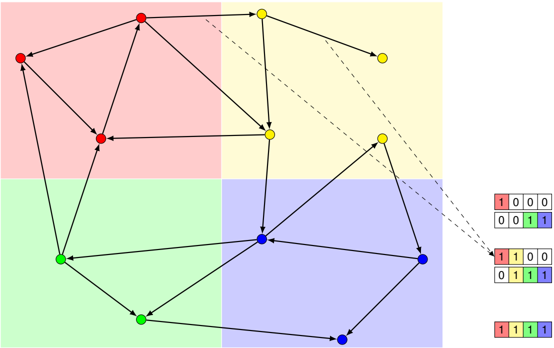
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



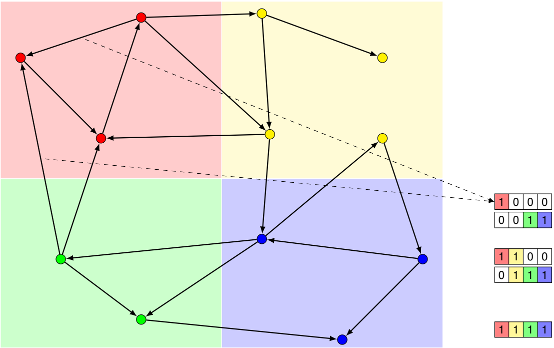
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



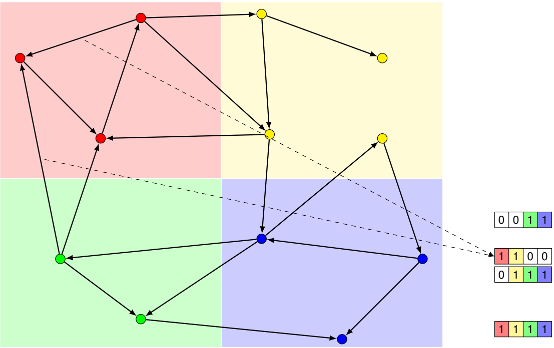
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



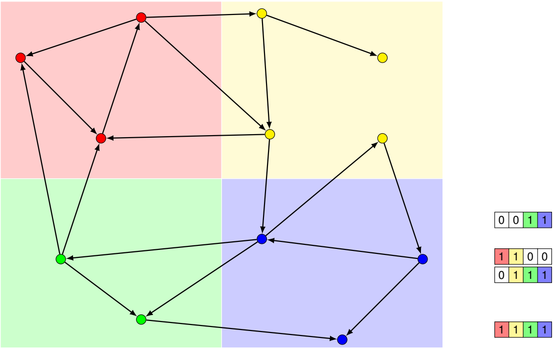
Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Flaggenkompression

Beobachtung:

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



Idee:

- verringere Anzahl eindeutiger Arc-Flags durch kippen
- dadurch Kompression der Matrix
- finde “gutes” Mapping

Experimente Kompression

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

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zufallsanfragen: Anzahl Regionen

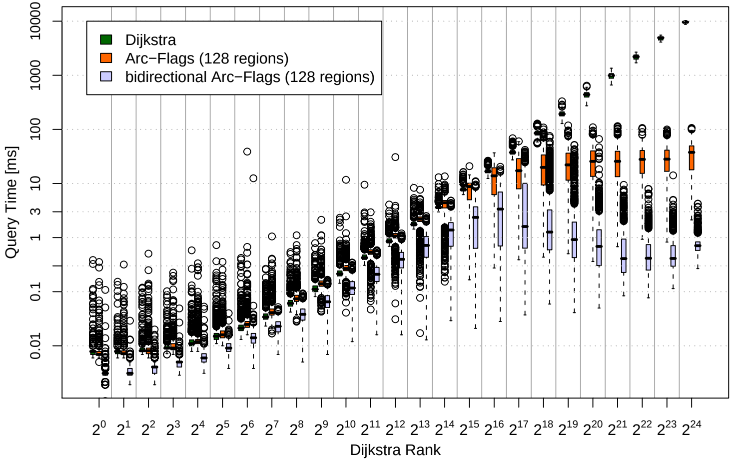
	Prepro		Query		
regions	time [min]	space [B/n]	# settled nodes	time [ms]	spd up
0	0	0	9 114 385	5 591.6	1.0
200	1 028	19	2 369	1.6	3 494.8
400	1 366	20	1 868	1.2	4 659.7
600	1 723	21	1 700	1.1	5 083.3
800	1 892	23	1 642	1.4	3 994.0
1000	2 156	25	1 593	1.1	5 083.3

Beobachtungen:

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

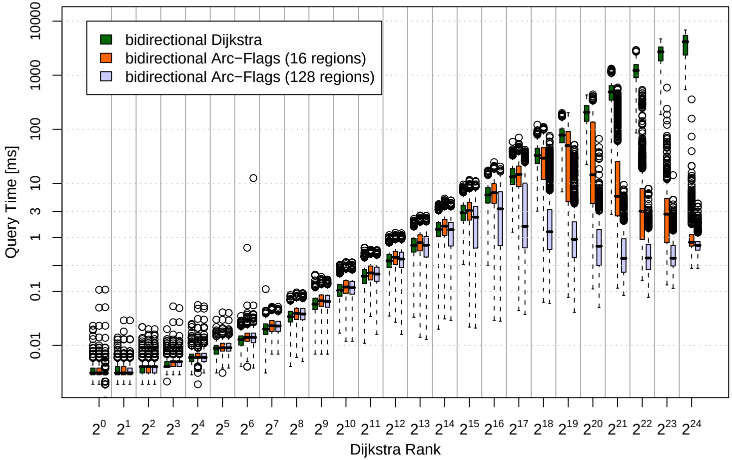
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Dijkstra Rank ArcFlags Uni- vs Bidir.



- gegenüber Dijkstra nur Beschleunigung für weite Anfragen
- birektionale Arc-Flags deutlich schneller als unidirektionale

Dijkstra Rank ArcFlags # Regionen



- gegenüber Dijkstra nur Beschleunigung für weite Anfragen
- 128 Regionen z.T. deutlich besser als 16 (bei 2^{24} fast gleichauf)

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 in Größenordnung von 10^3
- Nahe Anfragen nicht schneller als Dijkstra
- Hohe Vorberechnungszeit

SHARC

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC

Motivation:

- unidirektional
- Vorberechnungszeiten von Arc-Flags reduzieren

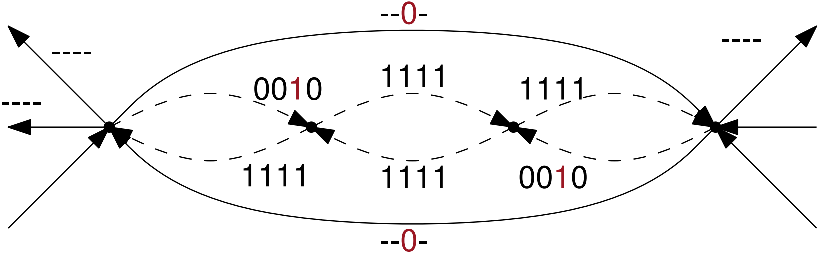
Ideen:

- Multi-Level Arc-Flags
- Integration von Kontraktion
- Verfeinern von Flaggen

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Integration von Kontraktion

KIT Karlsruher Institut für Technologie



Idee:

- sub-optimale Flaggen für entfernte Kanten
- in die Komponente rein: nur own-cell Flagge, sonst voll
- Shortcuts: own-cell false

$\Rightarrow$

- Vorberechnung nur auf kontrahierten Graph
- sub-optimale Flaggen nur am Anfang und Ende der Query
- Überspringen von Kanten automatisch durch Arc-Flags Query

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

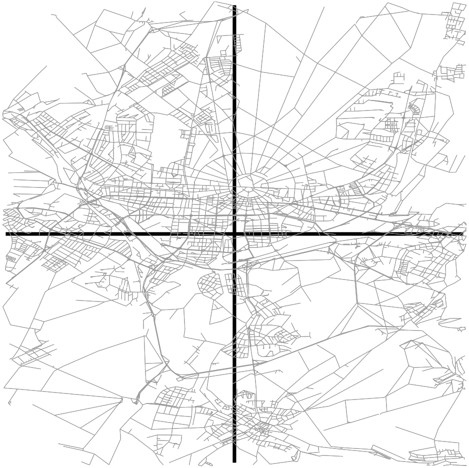
Vorberechnung:

- Multi-Level-Partition

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition

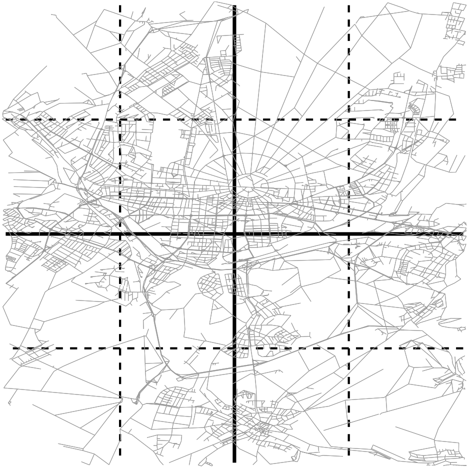


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition

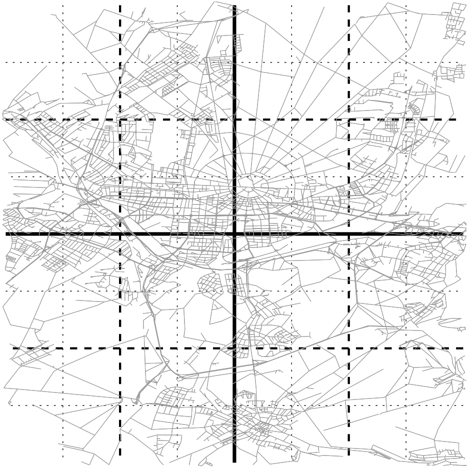


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen

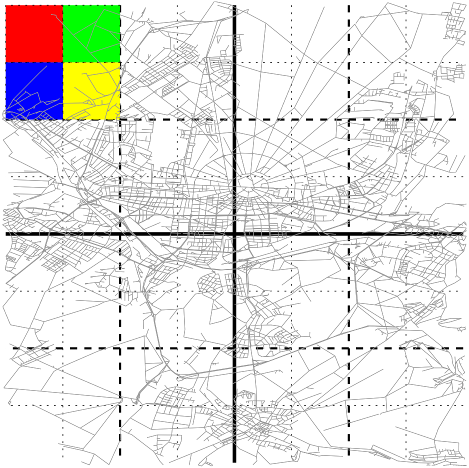


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - berechne Flaggen

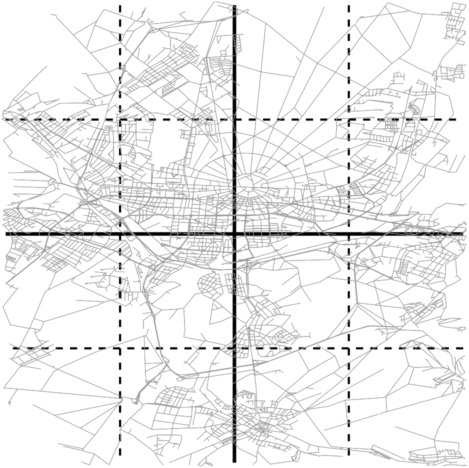


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

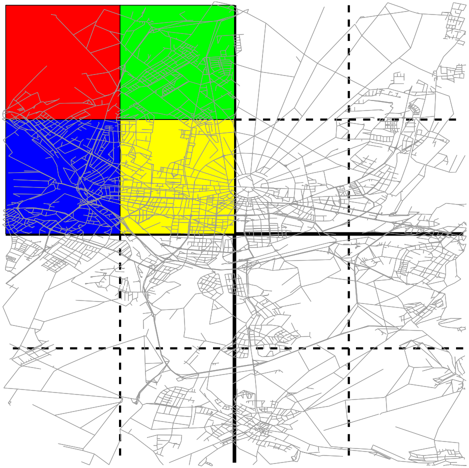
- Multi-Level-Partition
- iterativer Prozess:
    - kontrahiere Subgraphen



SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
    - berechne Flaggen

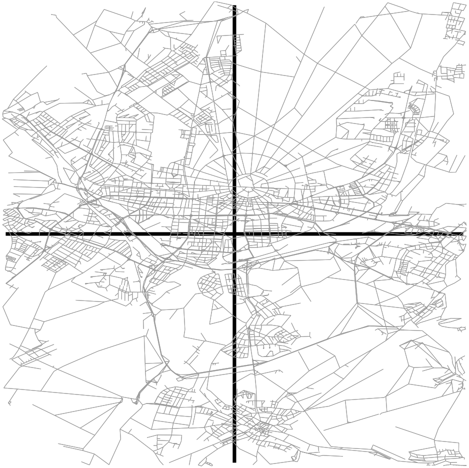


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen

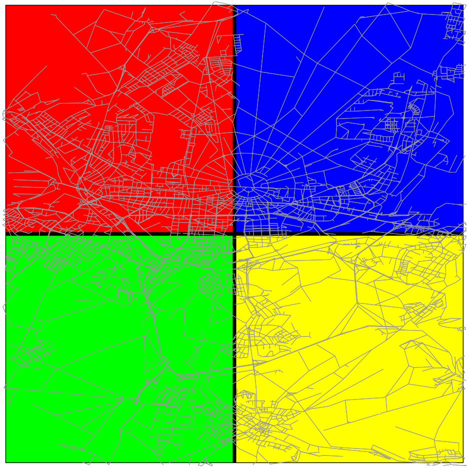


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen



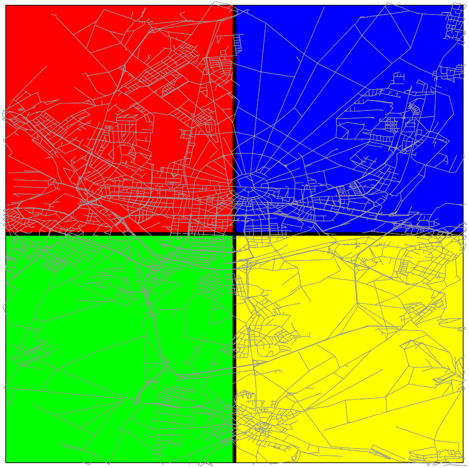
SHARC (Kontraktion, Arc-Flags)

Vorberechnung:

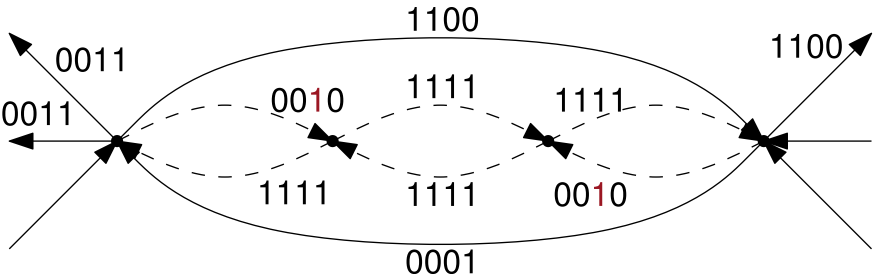
- Multi-Level-Partition
- iterativer Prozess:
  - kontrahiere Subgraphen
  - berechne Flaggen

Anfragen:

- uni- oder bidirektional
- hierarchisch (Kontraktion)
  LowestCommonSupercell(u,t)
- zielgerichtet (Arc-Flags)



Verfeinerung

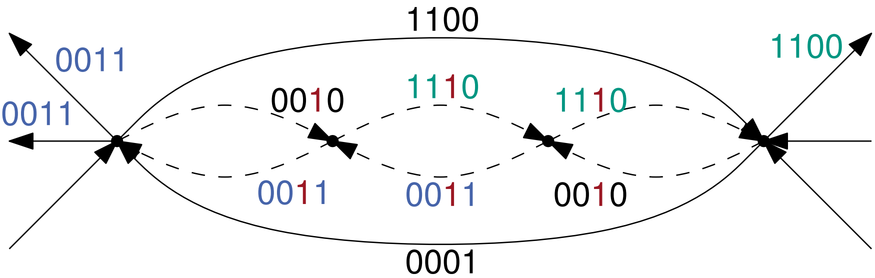


Beobachtung:

- Flaggen schlechter als nötig
- geht es besser?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Verfeinerung



Beobachtung:

- Flaggen schlechter als nötig
- geht es besser?

Idee:

- verfeinere Flaggen
- propagiere Flaggen von wichtigen zu unwichtigen Kanten
- mittels lokaler Suche

$\Rightarrow$ sehr gute Flaggen

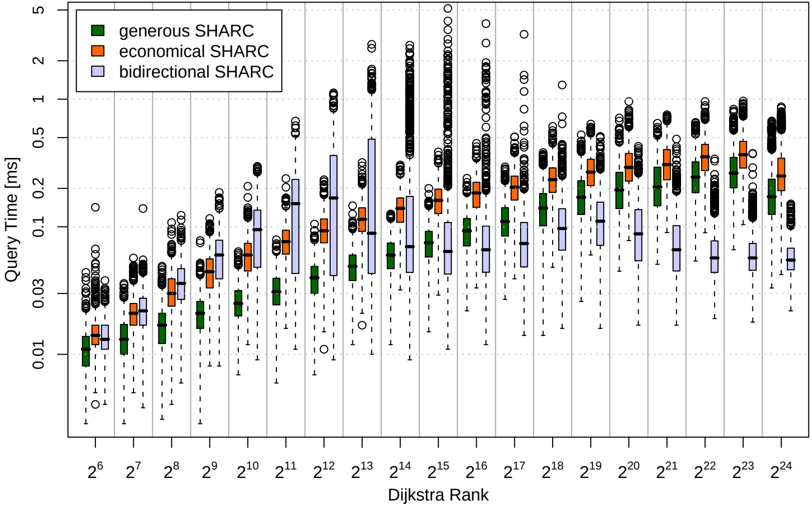
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Zufallsanfragen

	Europe				USA			
	PREPRO		QUERY		PREPRO		QUERY	
	time [h:m]	space [B/n]	#settled nodes	time [μs]	time [h:m]	space [B/n]	#settled nodes	time [μs]
generous SHARC	1:21	14.5	654	290	0:58	18.1	865	376
bidirectional SHARC	2:38	21.0	125	65	2:34	23.1	254	118
economical SHARC	0:34	13.7	784	355	0:38	17.2	1 230	578

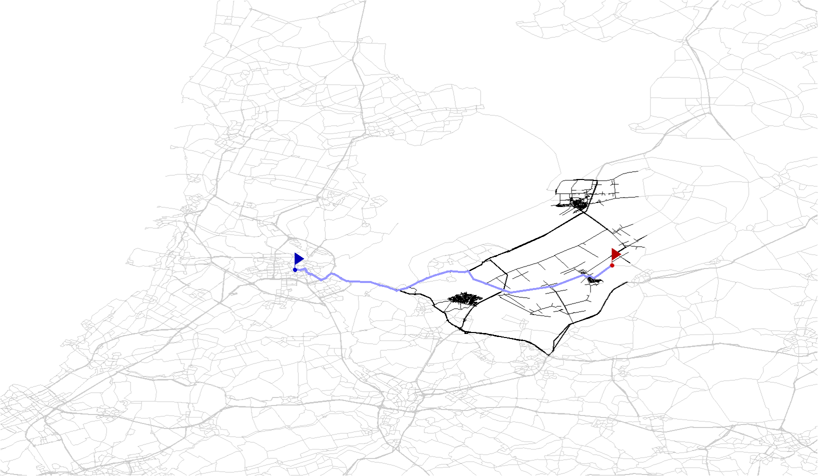
- generous vs economical: unterschiedliche Parametrisierung der Kontraktion und Verfeinerung, Details im Papier
- Um eine Größenordnung schneller als ArcFlags

Dijkstra Rank SHARC



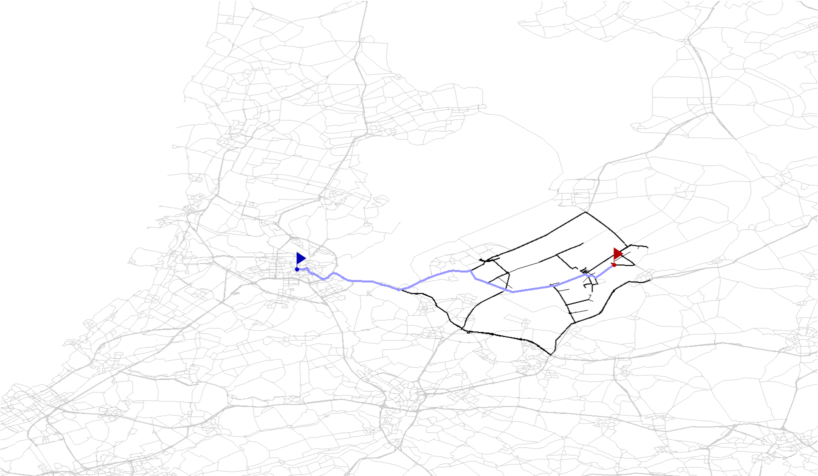
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Suchraum – Arc-Flags

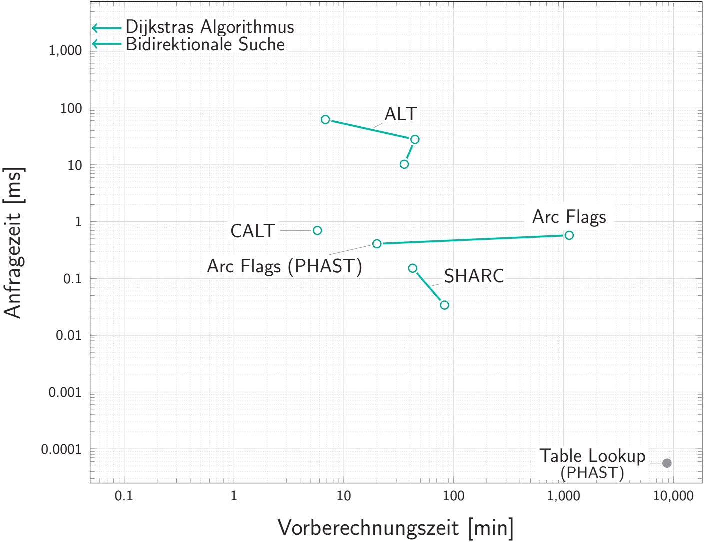


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Suchraum – SHARC

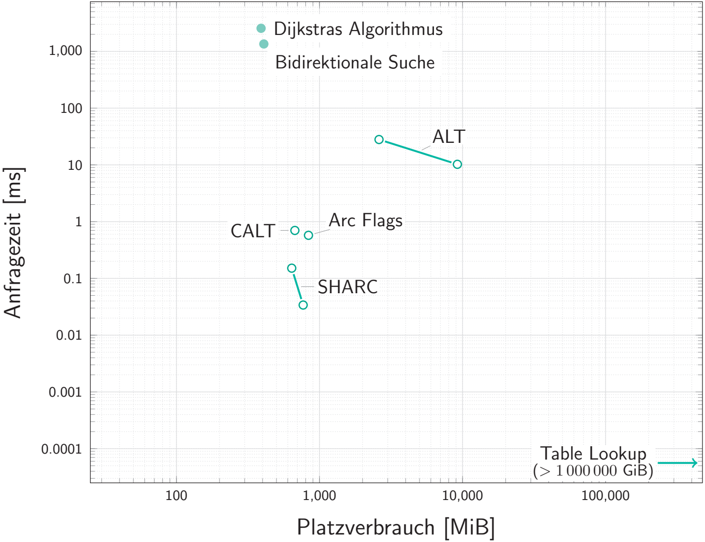


Übersicht bisherige Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Übersicht bisherige Techniken



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nächste Termine

Montag, 13.5.2019

(Valentin Buchhold)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Literatur I

Karlsruher Institut für Technologie

- Reinhard Bauer, Moritz Baum, Ignaz Rutter, and Dorothea Wagner.
  On the Complexity of Partitioning Graphs for Arc-Flags.
  Journal of Graph Algorithms and Applications, 17(3):265–299, 2013.

- Reinhard Bauer and Daniel Delling.
  SHARC: Fast and Robust Unidirectional Routing.
  ACM Journal of Experimental Algorithmics, 14(2.4):1–29, August 2009.
  Special Section on Selected Papers from ALENEX 2008.

- Moritz Hilger, Ekkehard Köhler, Rolf H. Möhring, and Heiko Schilling.
  Fast Point-to-Point Shortest Path Computations with Arc-Flags.
  In Camil Demetrescu, Andrew V. Goldberg, and David S. Johnson, editors, *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*, volume 74 of *DIMACS Book*, pages 41–72. American Mathematical Society, 2009.