Karlsruher Institut für Technologie

Algorithmen für Routenplanung

8. Vorlesung, Sommersemester 2023

Adrian Feilhauer | 17. Mai 2023

Kürzeste Wege in Straßennetzwerken
Beschleunigungstechniken (Fortsetzung)

Thema: Graphpartitionierung

- Exkurs: dünn besetzte Gleichungssysteme
- Wiederholung: Inertial Flow
- PUNCH
- FlowCutter

Exkurs: dünn besetzte Gleichungssysteme

Dünn besetzte Gleichungssysteme

- Lösen großer linearer Gleichungssysteme hat viele Anwendungen
- Schnelles Lösen ist wichtig
- Algorithmus von Gauß in $O(n^3)$, wobei n die Anzahl der Variablen ist
  - Oft zu langsam

- Oft sind Gleichungssysteme dünn besetzt
  - D. h. viele Koeffizienten sind 0
- Idee: Speichere 0-Koeffizienten nicht ab
- Aber: Wie 0-Koeffizienten während des Lösens erhalten?

Dünn besetzte Gleichungssysteme

Ziel: obere Dreiecksmatrix per Gaußelimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$

4 Nullen im oberen Dreieck

Dünn besetzte Gleichungssysteme

Ziel: obere Dreiecksmatrix per Gaußelimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 1 & -2 & -2 & -1 \\ 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

Dünn besetzte Gleichungssysteme

Ziel: obere Dreiecksmatrix per Gaußelimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{1}{2} \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

Dünn besetzte Gleichungssysteme

Ziel: obere Dreiecksmatrix per Gaußelimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & 0 & -1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

Keine Nullen übrig $\rightarrow$:(

Dünn besetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$

Dünn besetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

Dünn besetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Dünn besetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Dünn besetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Alle 4 Nullen erhalten $\rightarrow$:)

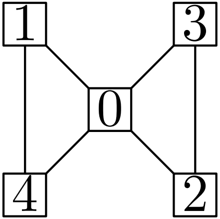
Dünn besetzte Gleichungssysteme

Warum funktioniert das?

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.

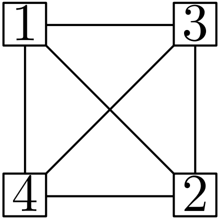
$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$



Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 1 & -2 & -2 & -1 \\ 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$



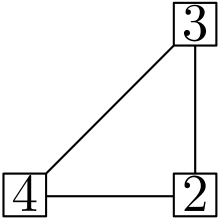
Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{1}{2} \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

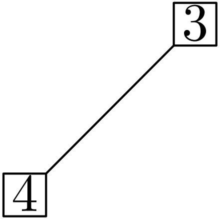


Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.



$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & 0 & -1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & 0 & -1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

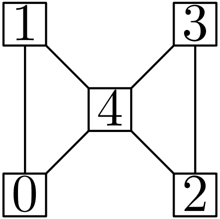
$$\boxed{4}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

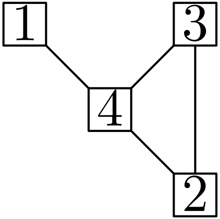


Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.



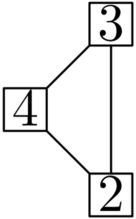
$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.



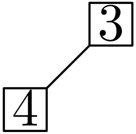
$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.



$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünn besetzte Gleichungssysteme

Jeder Eintrag, der nicht Null ist, entspricht einer Kante.

$\boxed{4}$

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion

Jeder Shortcut zerstört eine Null

Graphpartitionierung

Partitionierung

Anforderungen:

- ausbalanciert
- wenige Randknoten
- zusammenhängend

Partitionierung

Anforderungen:

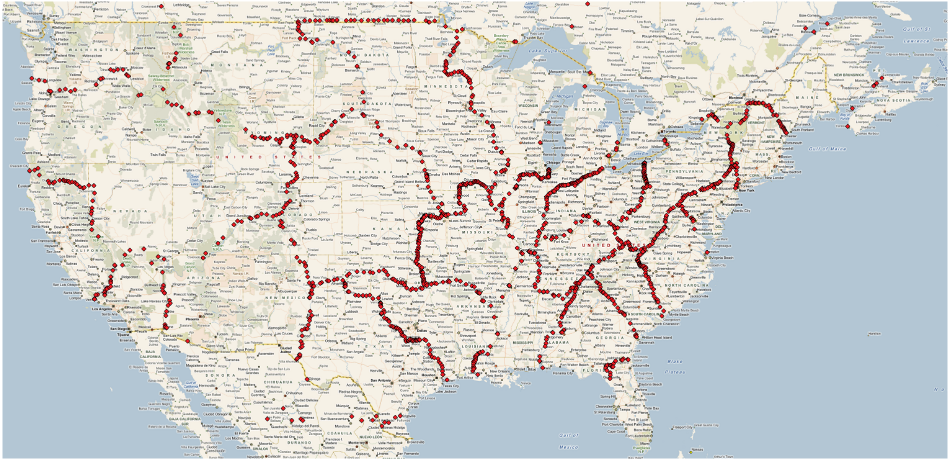
- ausbalanciert
- wenige Randknoten
- zusammenhängend

Black-Box-Partitionierer:

- benutzen keine Einbettung
- oft: teilen rekursiv Graphen in k Teile mit kleinem Schnitt
- lassen sich auf eine Vielzahl Graphklassen anwenden
- heute: spezielle Straßengraph-Partitionierer

Graphpartitionierung

- Informell: teile Graphen in lose verbundene Regionen (Zellen).



Graphpartitionierung

- Formale Definition:
  - Eingabe: ungerichteter Graph $G = (V, E)$
  - Ausgabe: Partition von V in Zellen $V_1, V_2, \ldots, V_k$
  - Ziel: minimale Anzahl Kanten zwischen Zellen
- Standardvariante: $|V_i| \leq U$ für festes U:
  - Anzahl Zellen kann variieren ($\geq \lceil n/U \rceil$).
- Balancierte Variante: k Zellen und Unausgeglichenheit ϵ:
  - genau k Zellen (können nicht-zusammenhängend sein), Größe $\leq (1 + \epsilon)\lceil n/k \rceil$.

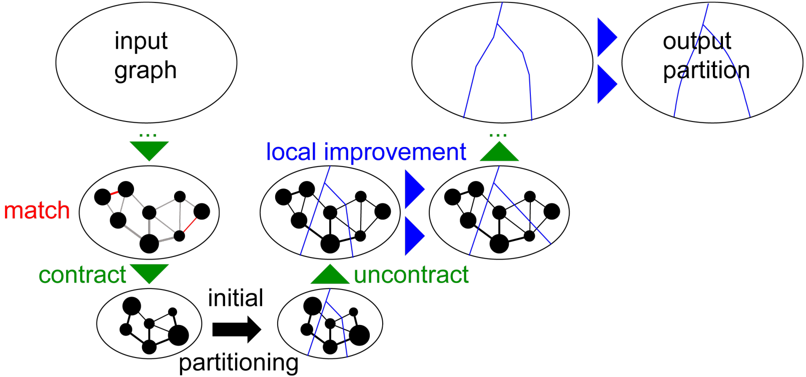
Graphpartitionierung

- Formale Definition:
  - Eingabe: ungerichteter Graph $G = (V, E)$
  - Ausgabe: Partition von V in Zellen $V_1, V_2, \dots, V_k$
  - Ziel: minimale Anzahl Kanten zwischen Zellen
- Standardvariante: $|V_i| \leq U$ für festes U:
  - Anzahl Zellen kann variieren ($\geq \lceil n/U \rceil$).
- Balancierte Variante: k Zellen und Unausgeglichenheit ϵ:
  - genau k Zellen (können nicht-zusammenhängend sein), Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

beides NP-schwer $\Rightarrow$ benutze Heuristiken

Black-Box-Partitionierer

- METIS [KK99]
- SCOTCH [PR96]
- DiBaP [MMS09]
- Kappa [HSS10], KaSPar [OS10], Kaffpa [SS11], KaffpaE [SS12]



Schnitte: Definitionen

Schnitt

- Partition des Graphen in 2 Teile (V_1, V_2)
- Größe: Anzahl Schnittkanten $(|S|)$

Minimaler s-t Schnitt

- entferne minimale Anzahl Kanten, sodass s und t im Graphen nicht mehr verbunden sind
- kann in maximalen Fluss überführt werden
- in Polynomialzeit zu berechnen

Dünnster Schnitt

- Schnitt mit $|S| / \min\{|V_1|, |V_2|\}$ minimal
- NP-schwer

Exact Graph Bisection

- Schnitt mit $|S|$ minimal und $|V_1|, |V_2| < \lceil |V|/2 \rceil$
- NP-schwer

Wiederholung: Inertial Flow

Inertial Flow

Idee:

- Nutze geographische Einbettung
- Basiert auf Max-Flow / Min-Cut
- Berechnet eine Bipartitionierung

Inertial Flow

Idee:

- Nutze geographische Einbettung
- Basiert auf Max-Flow / Min-Cut
- Berechnet eine Bipartitionierung

Algo:

- Für beide Diagonalen, die Horizontale und die Vertikale:
  - Projiziere Knoten auf Gerade
  - Orderne Knoten nach Position
  - Mache die bn ersten/letzten Knoten zur Quelle/Senke
    (ein typischer Wert für b ist 0.25 - 0.45)
  - Berechne einen Min-Cut
- Nimm den besten der 4 berechneten Schnitte

Inertial Flow

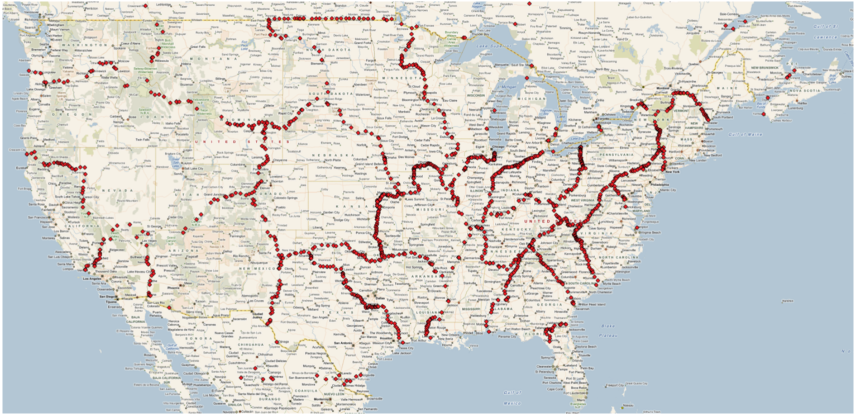
k-Partitionierung

- Inertial Flow teilt den Graph in zwei Teile
- Um k Teile zu erhalten gibt es folgenden einfachen Algorithmus:
  - Solange man weniger als k Teile hat:

  $\rightarrow$ Zerteile das größte Teil in zwei Teile

PUNCH

Natural Cuts



Straßengraphen: dichte Regionen (Gitter) abwechselnd mit natürlichen Schnitten

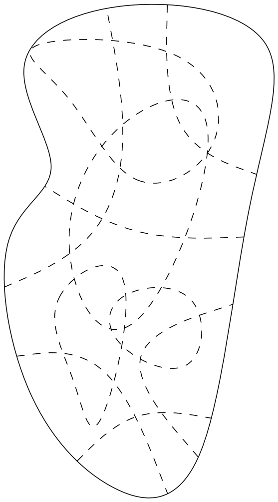
Flüsse, Berge, Wüsten, Wälder, Parks, Grenzen, Autobahnen, ...

PUNCH: Partitioner Using Natural-Cut Heuristics

Übersicht

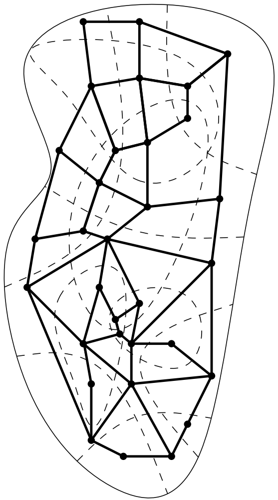
1. Ausdünnung:
   entspricht "match + contract" auf Folie 13
   - finde natürliche Schnitte (auf richtiger Skala)
   - behalte Schnittkanten, kontrahiere alle anderen

2. Zusammensetzen:
   - partitioniere (kleineren) kontrahierten Graphen
     "initial partitioning"
   - greedy + lokale Suche [+ Kombinationen]
     "uncontract + local improvements"



Übersicht

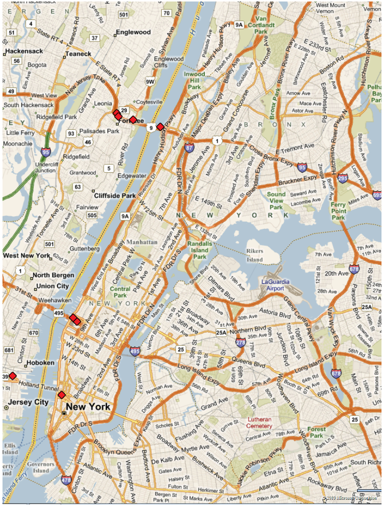
1. Ausdünnung:
   entspricht “match + contract” auf Folie 13
   - finde natürliche Schnitte (auf richtiger Skala)
   - behalte Schnittkanten, kontrahiere alle anderen
2. Zusammensetzen:
   - partitioniere (kleineren) kontrahierten Graphen
     “initial partitioning”
   - greedy + lokale Suche [+ Kombinationen]
     “uncontract + local improvements”



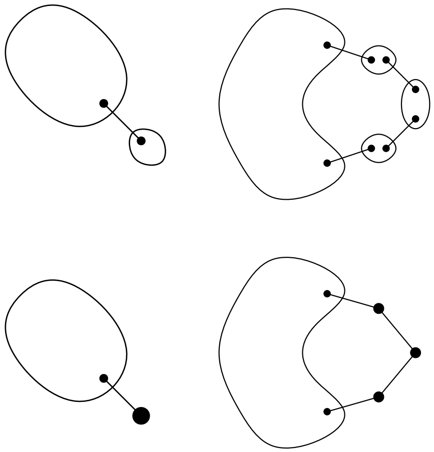
Finde natürliche Schnitte

Karlsruher Institut für Technologie

- Wir brauchen dünne Schnitte zwischen dichten Regionen kleiner gleich U:
- Dünnster Schnitt?
  - Zu aufwendig.
- Berechne minimalen s-t Schnitt (für zufällige s, t)?
  - Meist trivial: Knotengrade sind klein.
- Wir brauchen was anderes:
  - s-t Schnitte zwischen Regionen



Finde natürliche Schnitte



Berechne kleine Schnitte zuerst:

- identifiziere 1- and 2-Schnitte
- reduziert Straßengraph um Faktor 2
- beschleunigt Schnittfindung

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

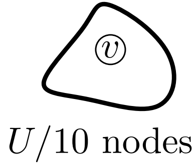
Finde natürliche Schnitte

1. Wähle ein Zentrum v.

v

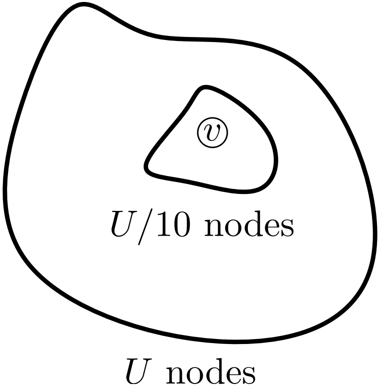
Finde natürliche Schnitte

1. Wähle ein Zentrum v.
2. Breitensuche der Größe U um v:
   - Erste $U/10$ Knoten: Kern
   - Verbliebene Knoten in der Queue: Ring



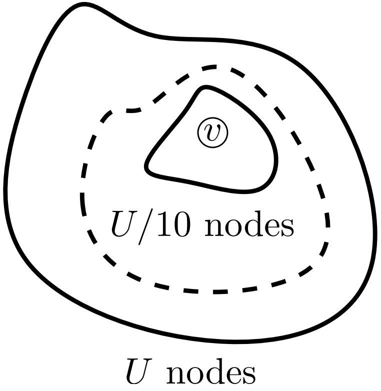
Finde natürliche Schnitte

1. Wähle ein Zentrum v.
2. Breitensuche der Größe U um v:
   - Erste $U/10$ Knoten: Kern
   - Verbliebene Knoten in der Queue: Ring



Finde natürliche Schnitte

1. Wähle ein Zentrum v.
2. Breitensuche der Größe U um v:
   - Erste $U/10$ Knoten: Kern
   - Verbliebene Knoten in der Queue: Ring
3. Finde minimum Kern/Ring Schnitt:
   - standard s–t minimaler Schnitt.
4. Wiederhole für verschiedene "zufällige" v:
   - bis jeder Knoten in ≥ 2 Kernen war



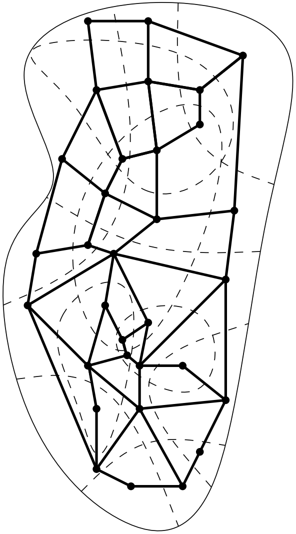
Eigenschaften der Ausdünung

1. viele Kanten werden nie geschnitten
2. Schnittkanten partitionieren den Graphen in Fragmente
3. Fragmentgröße $\leq U$ (meist viel kleiner)

- Generiere Fragmentgraph:
  - Fragment $\rightarrow$ gewichteter Knoten
  - benachbarte Fragmente $\rightarrow$ gewichtete Kante

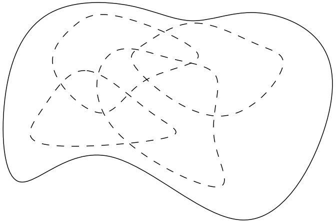
U	fragments	frag size
4 096	605 864	30
65 536	104 410	173
1 048 576	10 045	1 793

(Europe: 18M nodes)



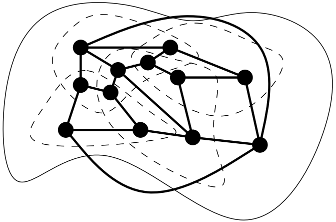
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:



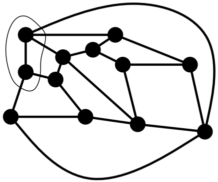
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment



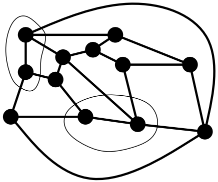
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen



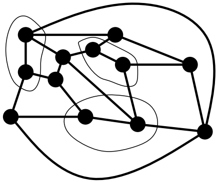
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen



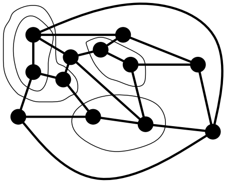
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen



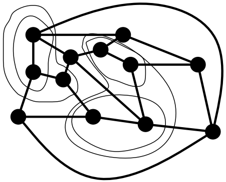
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen



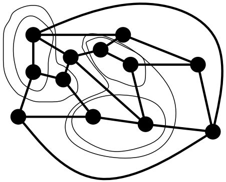
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen
  - stoppe wenn maximal



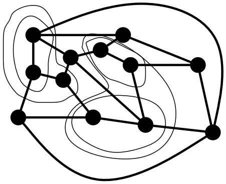
Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen
  - stoppe wenn maximal
- Zufällig greedy:
  - füge Fragmente zusammen, die stärker verbunden...
  - ...im Verhältnis zu ihrer Größe (= # Knoten, die sie repräsentieren).



Konstruktionsphase

- Fragmentgröße deutlich unterhalb von U, Schnitt unnötig groß
- Finde bessere Partition durch Zusammenfassen von Fragmenten
- Algorithmus:
  - starte mit einer Zelle pro Fragment
  - kombiniere adjazente Zellen
  - stoppe wenn maximal



- Zufällig greedy:
  - füge Fragmente zusammen, die stärker verbunden...
  - ...im Verhältnis zu ihrer Größe ($= \#$ Knoten, die sie repräsentieren).

Ergebnis okay, aber es geht besser.

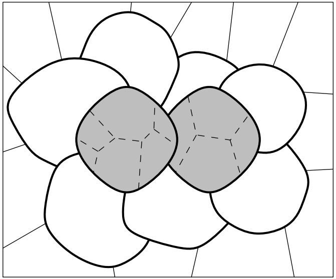
Lokale Suche

(Lokale Reoptimierung auf teilweise entpacktem Graphen)

Lokale Suche

(Lokale Reoptimierung auf teilweise entpacktem Graphen)

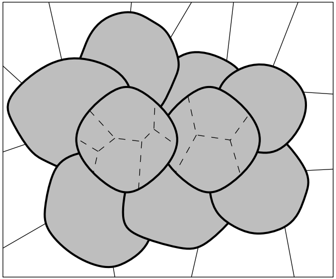
- Für paarweise benachbarte Zellen:
  - Zerteilung in Fragmente;
  - lass konstruktiven, randomisierten Algorithmus auf Subproblem laufen;
  - behalte Lösung wenn besser.



Lokale Suche

(Lokale Reoptimierung auf teilweise entpacktem Graphen)

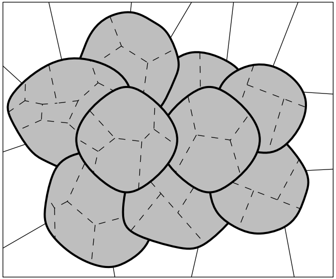
- Für paarweise benachbarte Zellen:
  - Zerteilung in Fragmente;
  - lass konstruktiven, randomisierten Algorithmus auf Subproblem laufen;
  - behalte Lösung wenn besser.
- Variante benutzt auch Nachbarzellen:
  - mehr Flexibilität;
  - beste Ergebnisse (Standard).



Lokale Suche

(Lokale Reoptimierung auf teilweise entpacktem Graphen)

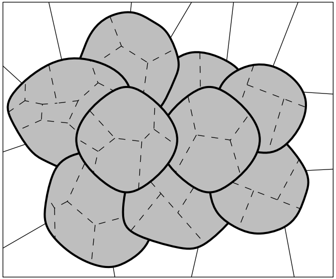
- Für paarweise benachbarte Zellen:
  - Zerteilung in Fragmente;
  - lass konstruktiven, randomisierten Algorithmus auf Subproblem laufen;
  - behalte Lösung wenn besser.
- Variante benutzt auch Nachbarzellen:
  - mehr Flexibilität;
  - beste Ergebnisse (Standard).
- Nachbarzellen können auch in Fragmente zerteilt werden:
  - Subprobleme zu groß;
  - schlechtere Ergebnisse.



Lokale Suche

(Lokale Reoptimierung auf teilweise entpacktem Graphen)

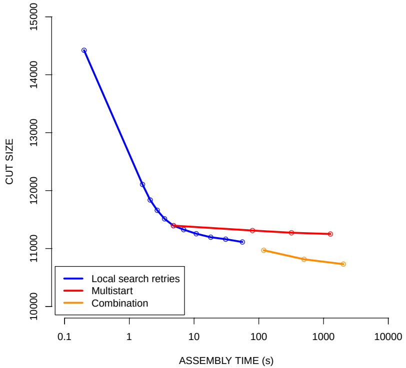
- Für paarweise benachbarte Zellen:
  - Zerteilung in Fragmente;
  - lass konstruktiven, randomisierten Algorithmus auf Subproblem laufen;
  - behalte Lösung wenn besser.
- Variante benutzt auch Nachbarzellen:
  - mehr Flexibilität;
  - beste Ergebnisse (Standard).
- Nachbarzellen können auch in Fragmente zerteilt werden:
  - Subprobleme zu groß;
  - schlechtere Ergebnisse.



Evaluiere jedes Subproblem mehrmals (mit Zufall).

Weitere Verbesserungen

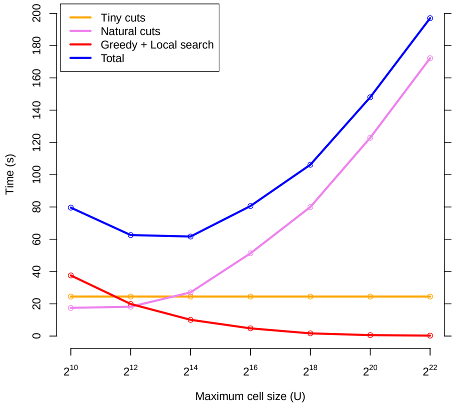
- Mehrfacher Test für jedes Paar
  - lokale Suche hat Zufallskomponenten
- Multistart:
  - konstruktiv + lokale Suche;
  - behalte beste Lösung.
- Kombination:
  - kombiniere manche Lösungen;
  - merge + lokale Suche.



(Europe, $U = 2^{16}$)

längere Laufzeit $\rightarrow$ bessere Lösungen

Laufzeit



Europe (18M vertices), 12 cores

Flaschenhälse: Aufbau für kleine U, Ausdünnung für große U

Qualität

U	A	B	$B/\sqrt{U}$	$B/\sqrt[3]{U}$
1 024	895	16.8	0.52	1.66
4 096	3 602	27.6	0.43	1.73
16 384	14 437	45.6	0.36	1.80
65 536	57 376	72.7	0.28	1.80
262 144	222 626	103.7	0.20	1.62
1 048 576	826 166	134.3	0.13	1.32
4 194 304	3 105 245	127.9	0.06	0.79

(Europe, 16 retries, no multistart/combination)

U: maximal erlaubte Zellengröße

A: durchschn. Zellengröße der Lösungen

B: durchschn. Randknoten pro Zelle

Straßengraphen haben sehr kleine Separatoren

Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH
   mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen,
   verteile den Rest (Multistart)

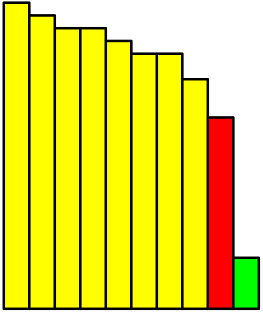
Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH
   mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen,
   verteile den Rest (Multistart)



Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen, verteile den Rest (Multistart)

Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH
   mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen,
   verteile den Rest (Multistart)

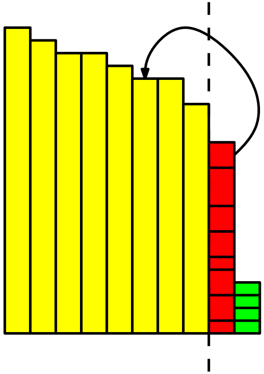
Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen, verteile den Rest (Multistart)



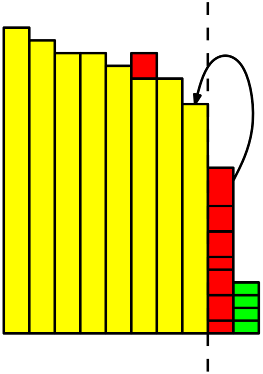
Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen, verteile den Rest (Multistart)



Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

1. benutze Standard-PUNCH
   mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen,
   verteile den Rest (Multistart)

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

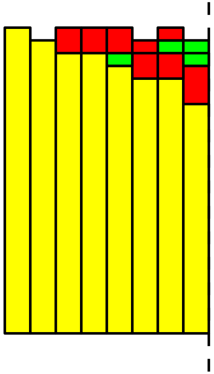
Balancierte Partitionen

Andere Verfahren lösen balancierte Variante:

- finde k Zellen mit Größe $\leq (1+\epsilon)\lceil n/k \rceil$.

Erweiterung von PUNCH:

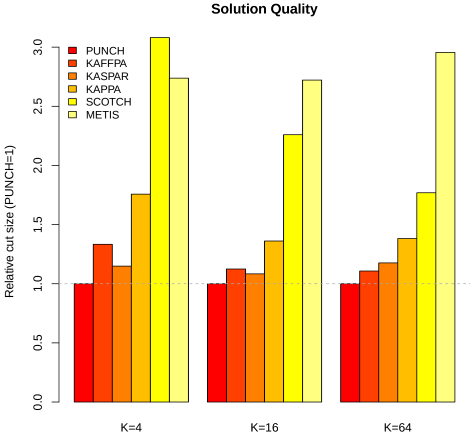
1. benutze Standard-PUNCH
   mit $U = (1+\epsilon)\lceil n/k \rceil$;
2. wähle k Basis Zellen,
   verteile den Rest (Multistart)



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

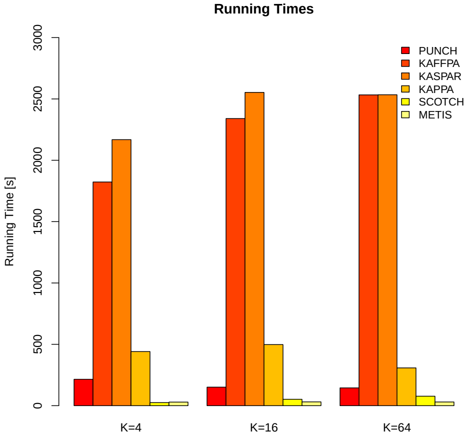
Balancierte Partitionen

PUNCH: bessere Lösungen...



(Europe, $\epsilon = 0.03$)

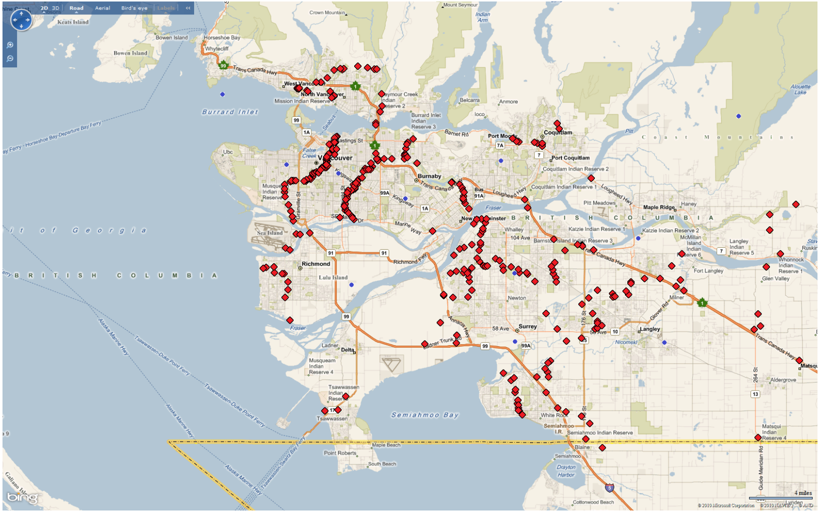
...in vernünftiger Zeit.



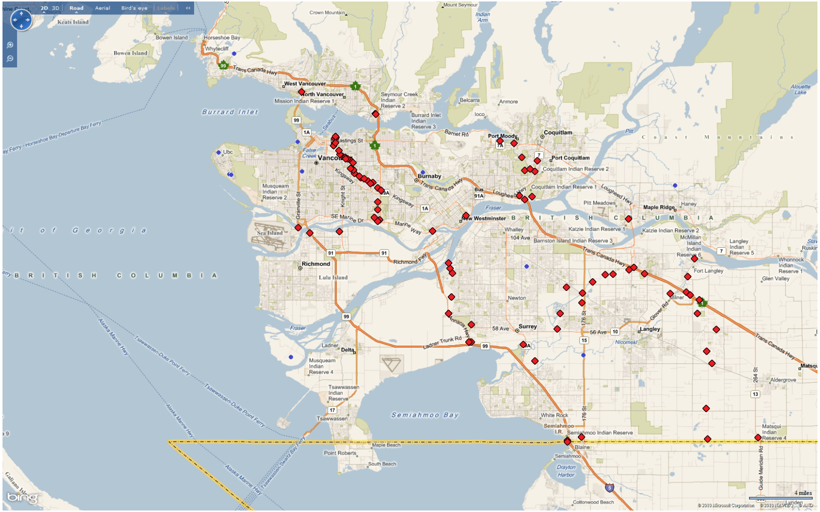
(Europe, $\epsilon = 0.03$)

Vancouver mit METIS

Karlsruher Institut für Technologie

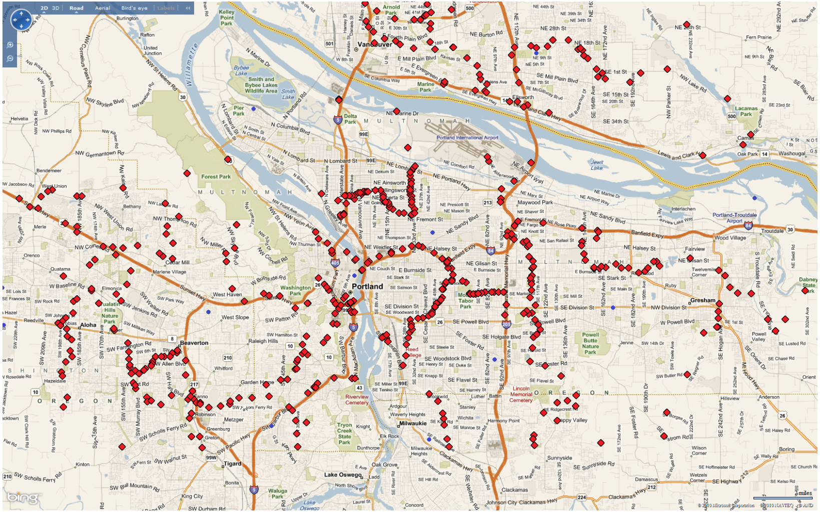


Vancouver mit PUNCH



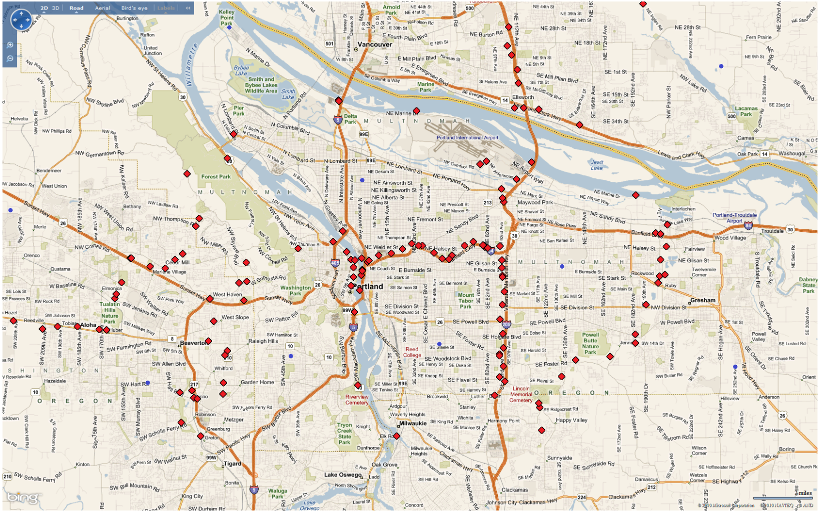
Portland mit METIS

Karlsruher Institut für Technologie



Portland mit PUNCH

Karlsruher Institut für Technologie

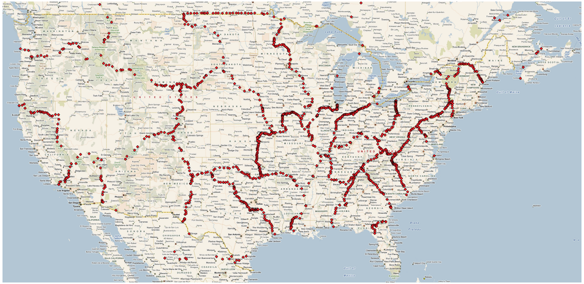


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Bemerkungen

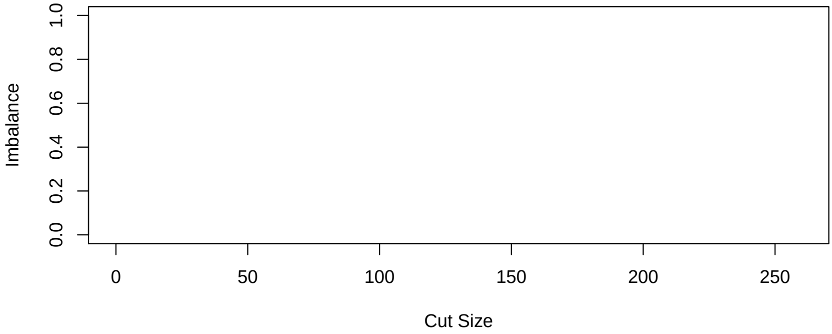
- reduziert Randknoten gegenüber METIS um mehr als Faktor 2
- Beschleunigung von Arc-Flags Vorberechnung um Faktor 2
- auch Multilevel-Partitionen berechnenbar

  top-down liefert beste Ergebnisse
- dabei Vorteile gegenüber METIS sogar größer
- Wie weit vom Optimum entfernt?
- Wird für die Bing Routing Engine benutzt.



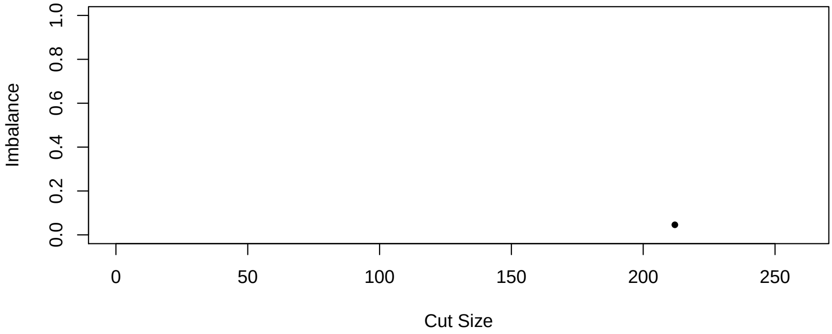
FlowCutter

“Gute” Schnitte



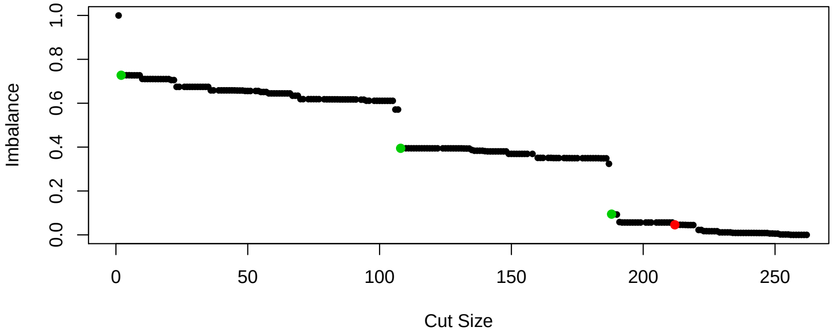
- Imbalance $= 0 \rightarrow$ Schnitt perfekt balanciert
- Imbalance $= 1 \rightarrow$ alle Knoten auf einer Seite

“Gute” Schnitte



- Europa Graph mit $18 \cdot 10^6$
- Ergebnis für maximale imbalance von 5%
- Ist der Schnitt gut?
  (Vielleicht, da $212 \lll 18 \cdot 10^6$?)

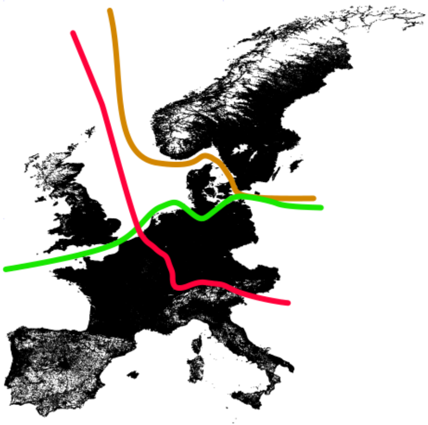
“Gute” Schnitte



- Grün markierte Schnitte bieten besseren Trade-Off

→ Pareto-Front benötigt um gute Schnitte zu wählen

“Gute” Schnitte Visualisiert



FlowCutter

FlowCutter

- Heuristischer Algorithmus zum Finden balancierter *st*-Schnitte
- Liefert zusammenhängende Partitionen
- Findet Pareto-optimale Lösungen für Imbalance und Schnittgröße

FlowCutter

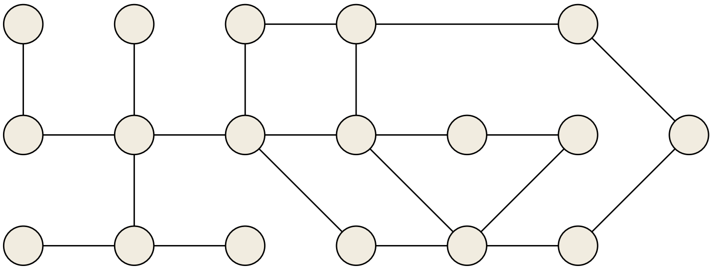
FlowCutter

- Heuristischer Algorithmus zum Finden balancierter *st*-Schnitte
- Liefert zusammenhängende Partitionen
- Findet Pareto-optimale Lösungen für Imbalance und Schnittgröße

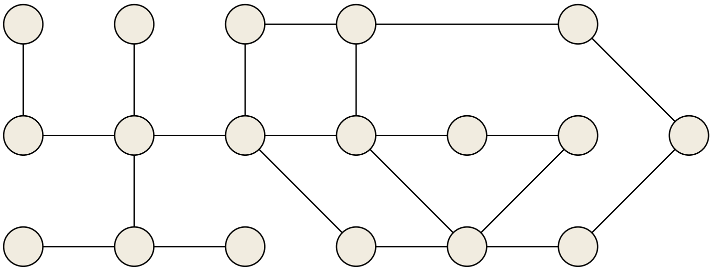
Idee

- Füge schrittweise Start- und Zielknoten hinzu um Schnitt zu ändern
  - Erweitere Start-/Zielregion bis zum Schnitt (kleinere Seite)
  - Füge einen Knoten jenseits des Schnitts hinzu
- Berechne neuen Schnitt

Beispiel FlowCutter

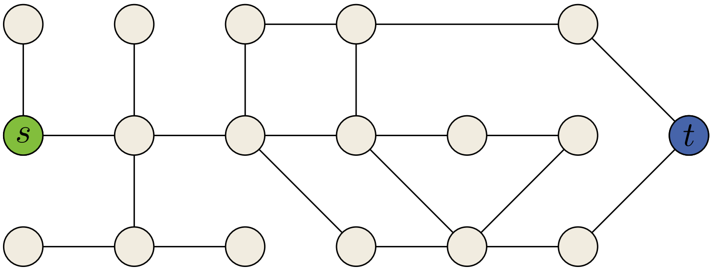


Beispiel FlowCutter



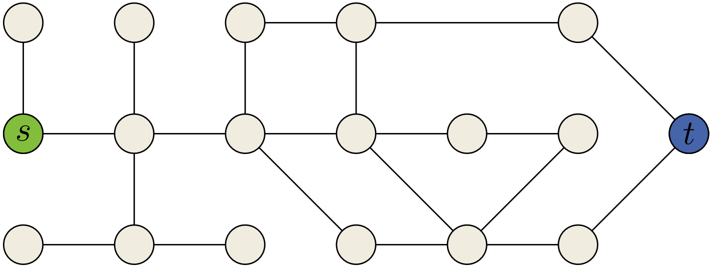
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



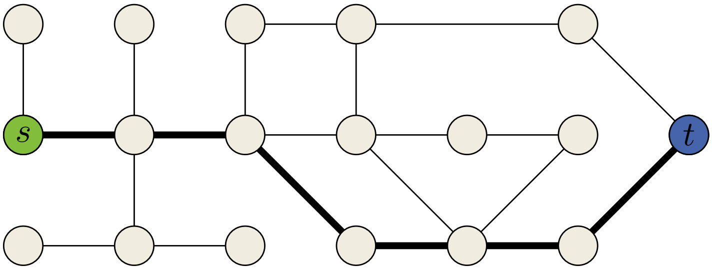
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



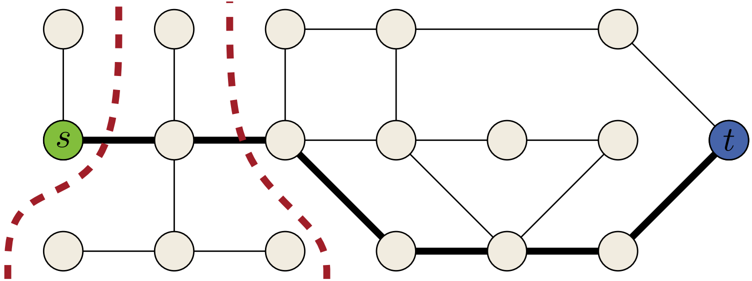
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



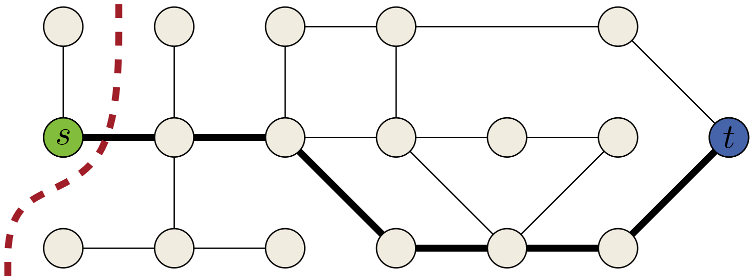
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



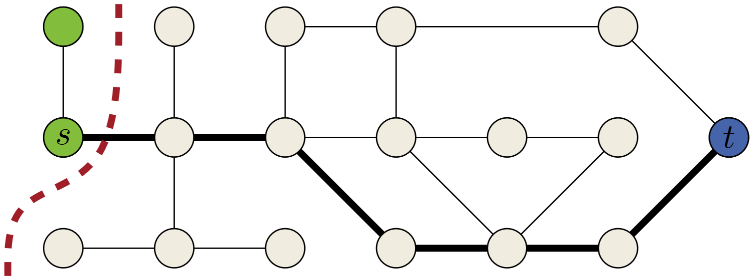
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



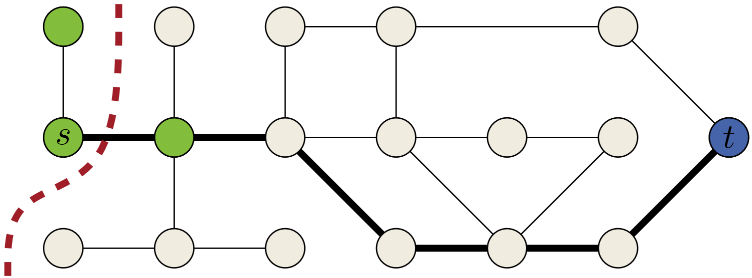
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



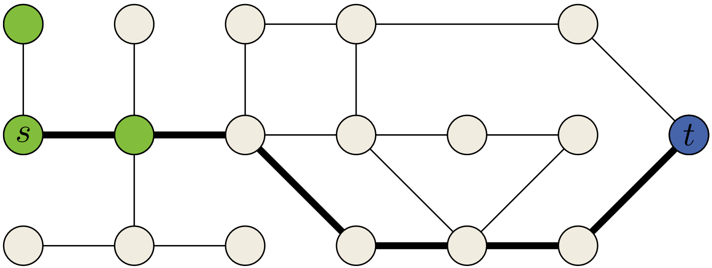
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



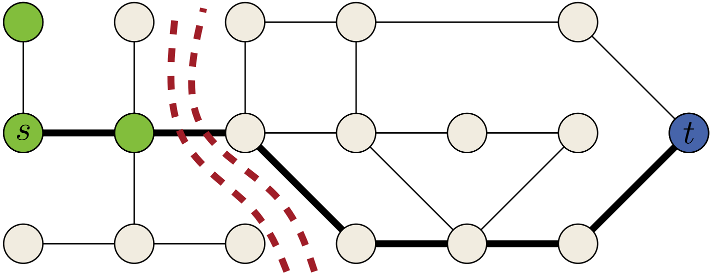
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



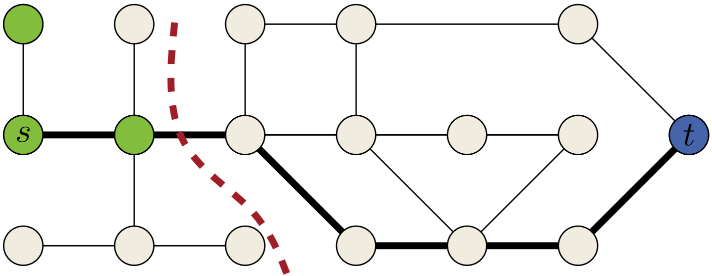
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



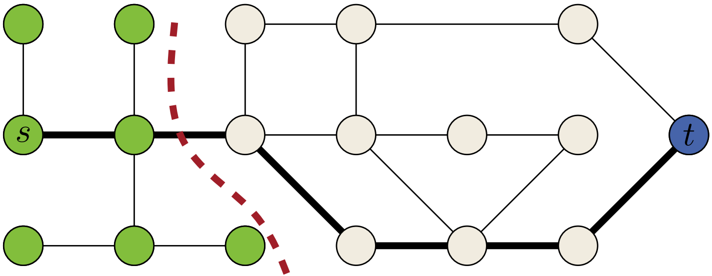
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



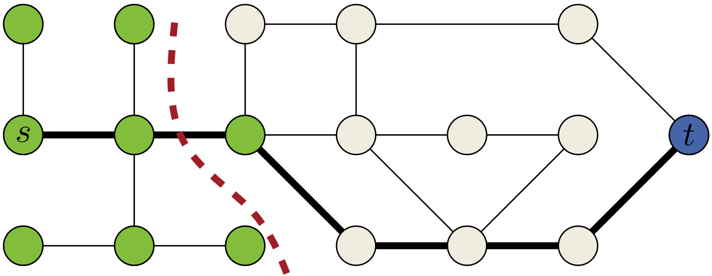
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



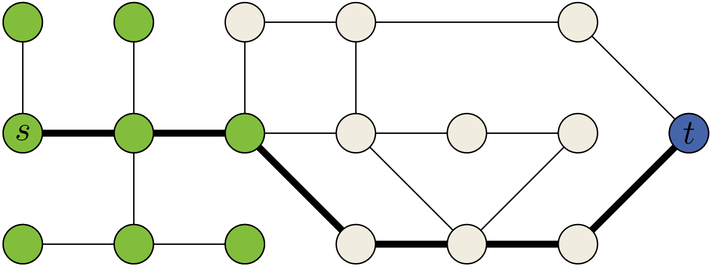
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



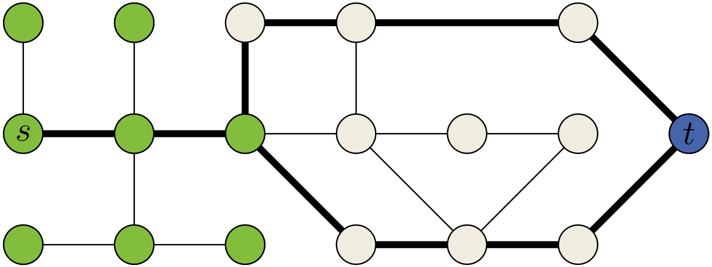
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



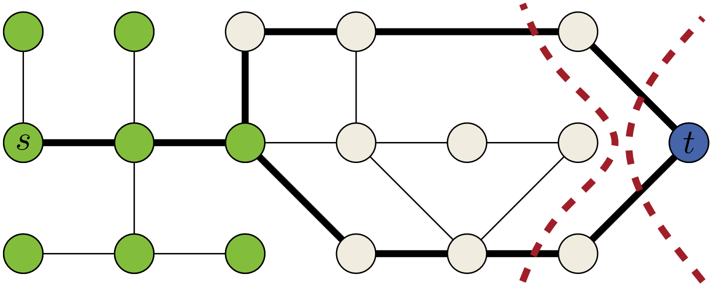
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



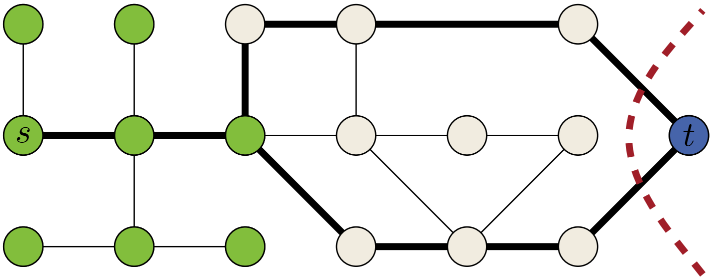
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



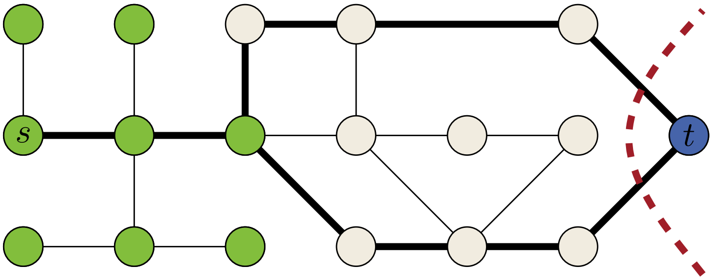
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



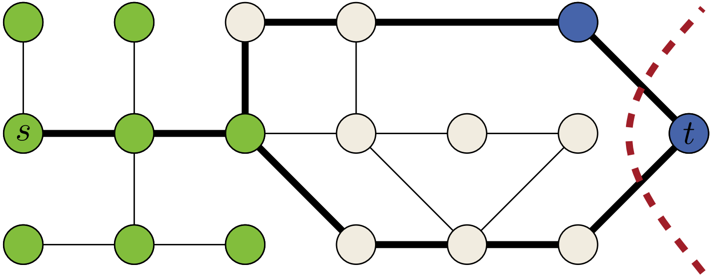
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



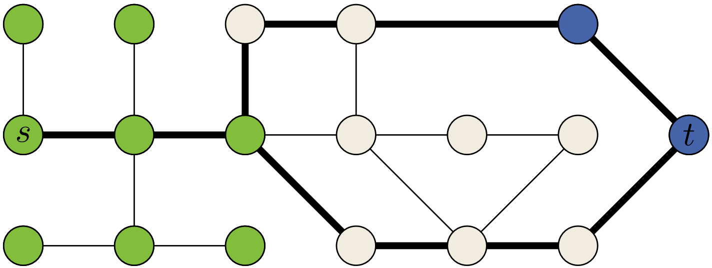
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



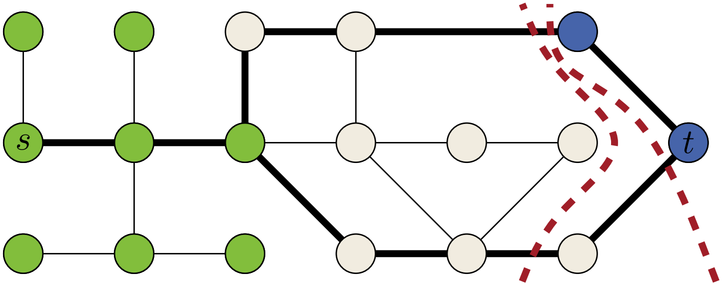
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



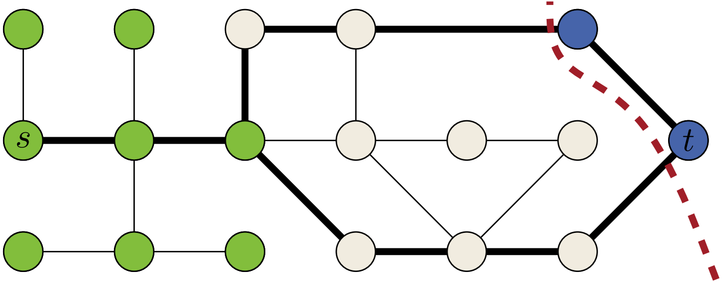
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



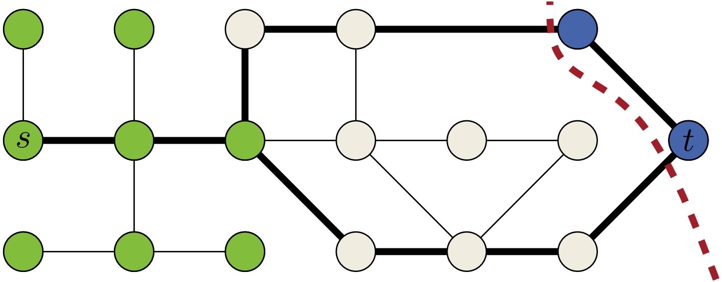
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



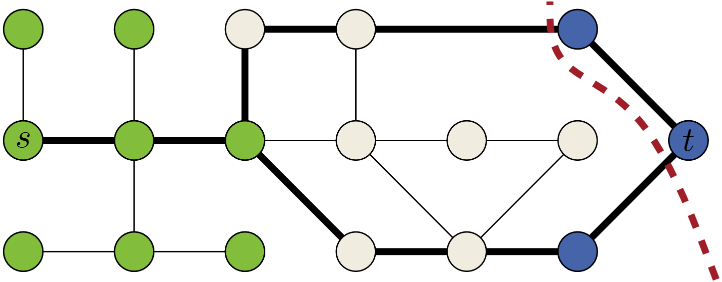
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



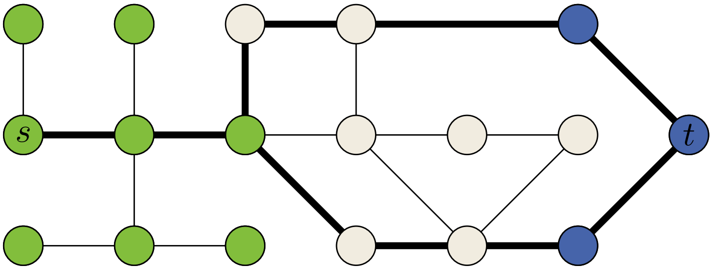
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



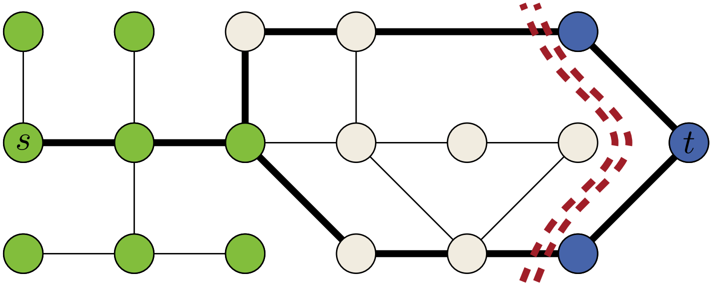
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



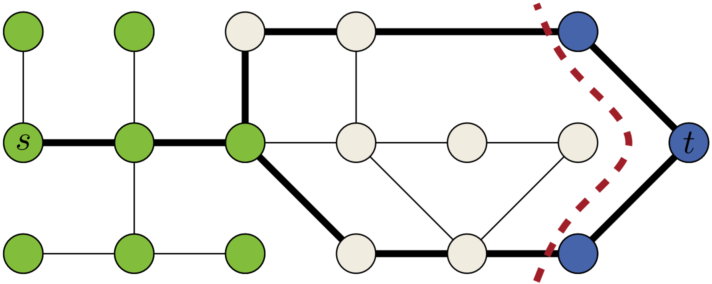
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



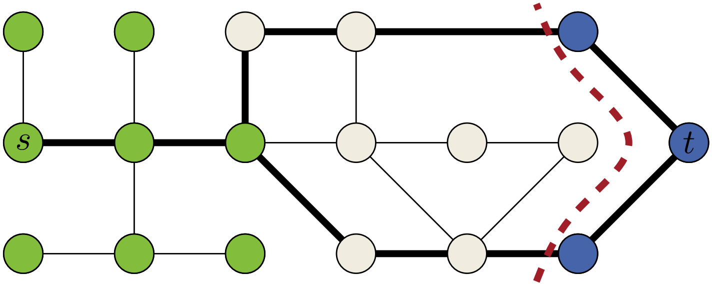
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



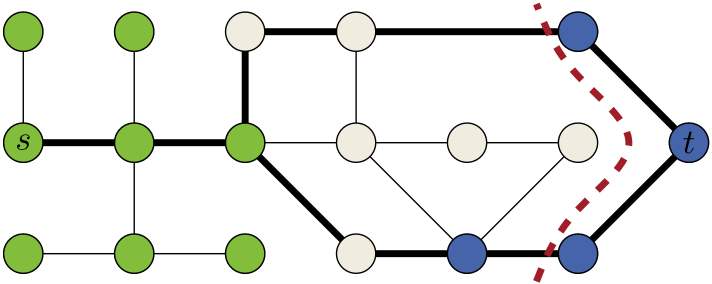
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



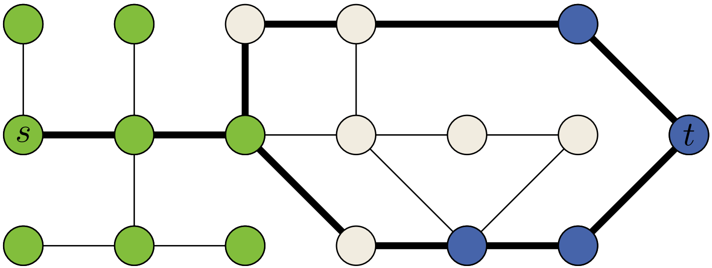
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



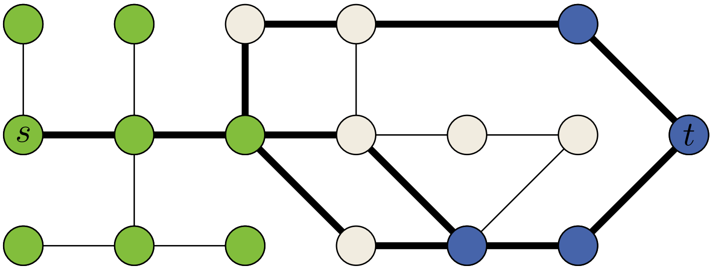
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



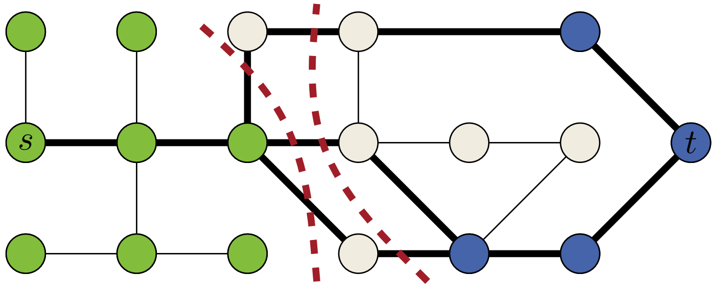
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



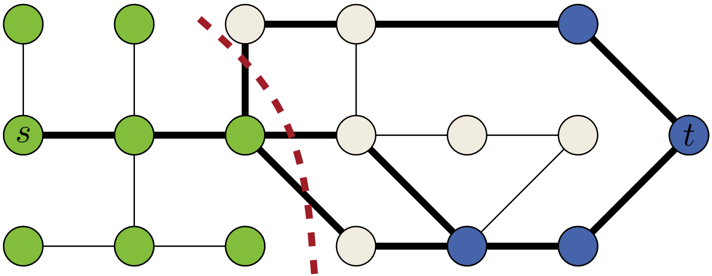
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



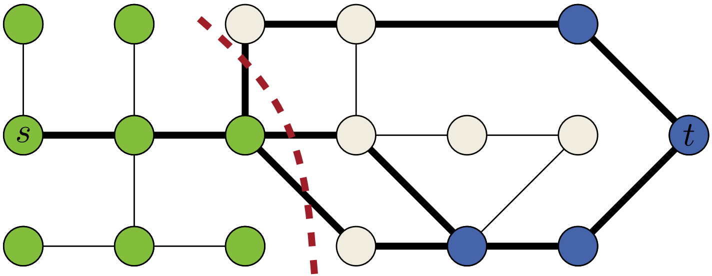
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



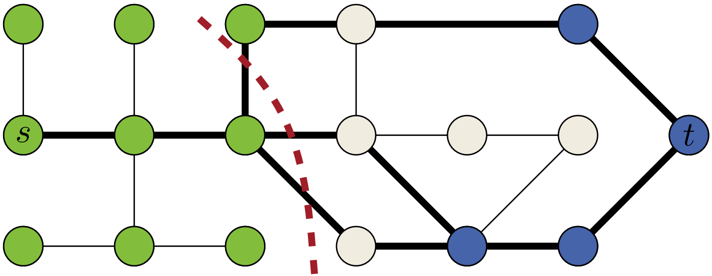
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



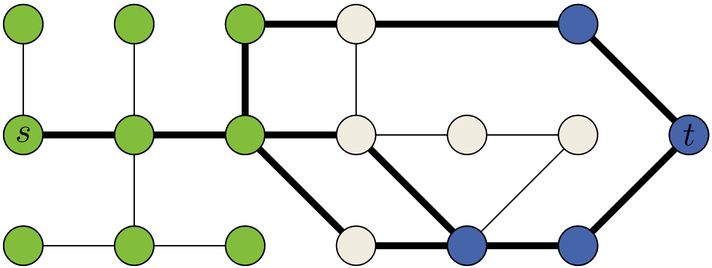
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



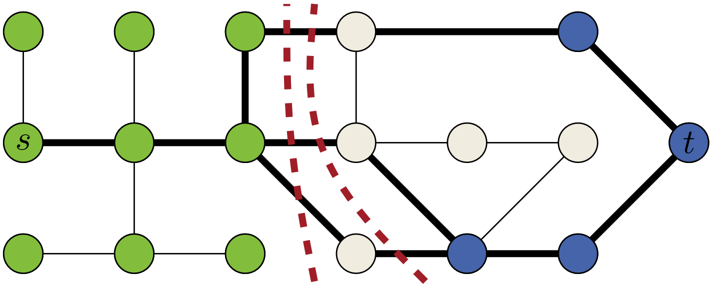
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



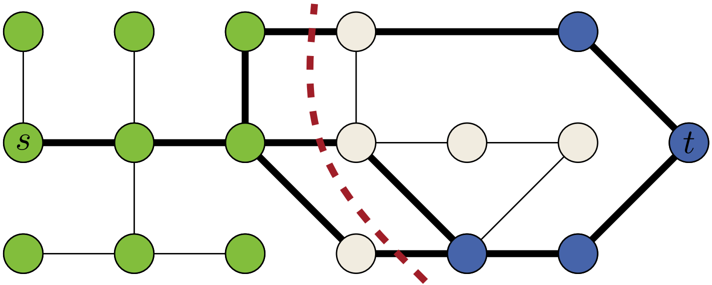
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



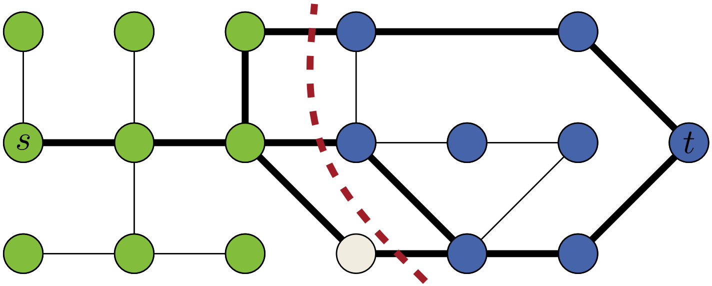
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



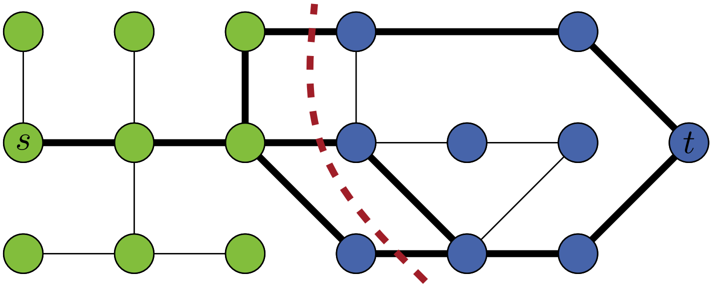
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



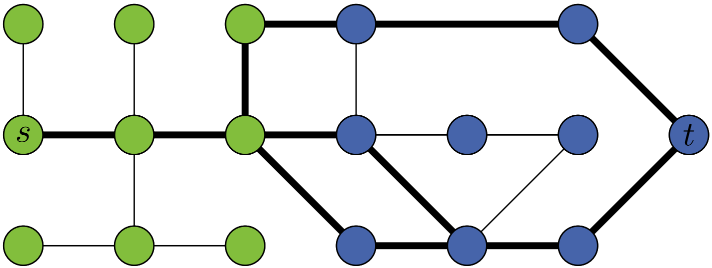
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



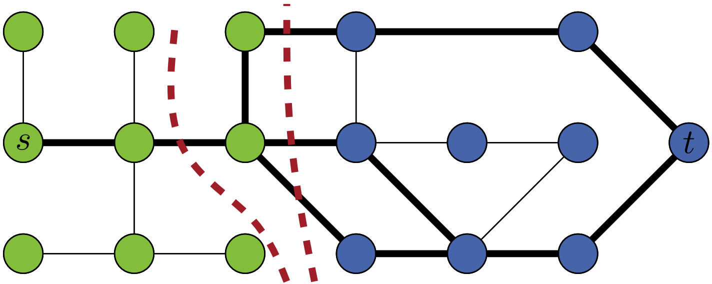
1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

Beispiel FlowCutter



1. Erhöhe Fluss
2. Finde s- und t-Schnitt
3. Wähle kleinere Seite
4. Assimiliere Seite
5. Pierce Schnitt

FlowCutter - Eigenschaften

- Jeder gefundene Schnitt ist eine gültige Partitionierung
- Schnitt wird nie kleiner
- Balanciertheit wird (für gleiche Schnittgröße) nie schlechter

FlowCutter - Eigenschaften

- Jeder gefundene Schnitt ist eine gültige Partitionierung
- Schnitt wird nie kleiner
- Balanciertheit wird (für gleiche Schnittgröße) nie schlechter

$\rightarrow$ Schritte liefern (Pareto) Verbesserungen

$\rightarrow$ Überschreibe letztes Ergebnis, wenn Schnitt gleich groß ist

FlowCutter - Eigenschaften

- Jeder gefundene Schnitt ist eine gültige Partitionierung
- Schnitt wird nie kleiner
- Balanciertheit wird (für gleiche Schnittgröße) nie schlechter

$\rightarrow$ Schritte liefern (Pareto) Verbesserungen

$\rightarrow$ Überschreibe letztes Ergebnis, wenn Schnitt gleich groß ist

Findet Pareto Optimale Menge an Schnitten (heuristisch)

Piercing Knoten Auswahl

Auswahl des Piercing Knotens:

- Optimale Wahl vermutlich NP-hart

$\rightarrow$ Heuristisch

Orakel zur Piercing Knoten Wahl:

- **Wichtigste Regel**:
  Wenn möglich, wähle Knoten, der Schnitt nicht vergrößert
- **Nachgelagerte Regel**:
  - Für Start-Seite: Maximiere $\mathrm{dist}(x, t) - \mathrm{dist}(s, x)$
  - Für Ziel-Seite: Minimiere $\mathrm{dist}(x, t) - \mathrm{dist}(s, x)$

Laufzeit Analyse

Konstanten:

- c: Größe des letzten berechneten Schnitts
- m: Anzahl an Kanten

Laufzeit Optimierung:

- Fluss wird c Mal augmentiert
- mit der richtigen Implementierung: $O(m)$ pro Augmentierung

$\rightarrow$ $O(cm)$ Laufzeit

Andere Schnitt-Szenarien

Knoten Separatoren

- Wähle inzidente Knoten zu Schnittkanten (größere Seite)

Andere Schnitt-Szenarien

Knoten Separatoren

- Wähle inzidente Knoten zu Schnittkanten (größere Seite)

Schnitt (ohne spezifische s, t)

- Wähle s und t zufällig (gleichverteilt)
- Lasse FlowCutter mehrfach laufen
- Findet (probabilistisch) Schnitte mit guter Balanciertheit

CCH Experimente

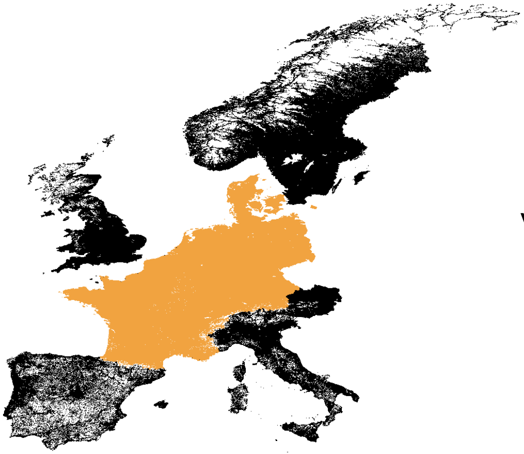
	Search Space		#Arcs		Running times		
	# Nodes		in CCH	#Tri.	Order	Cust.	Query
	Avg.	Max.	[$\cdot 10^6$]	[$\cdot 10^6$]	[min]	[ms]	[μs]
M	1 223	1 983	69.9	1 390	**2**	2 242	1 162
K	639	1 224	73.9	578	$\leq$3 552	975	304
I	733	1 569	67.4	590	17	932	385
F3	734	1 159	60.3	519	42	853	366
F20	**616**	**1 102**	**58.8**	**460**	281	**780**	**271**

Europe Graph

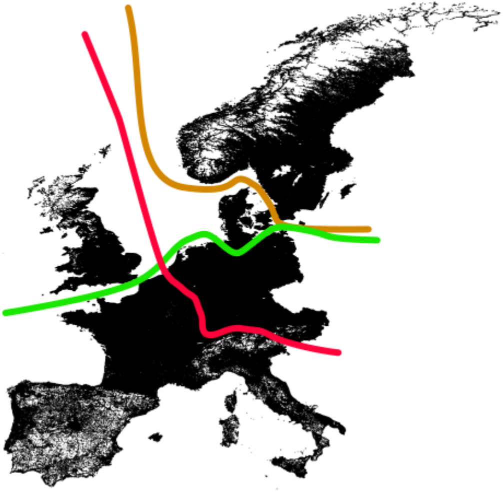
M = Metis, K = KaHip, I = InertialFlow

Fx = FlowCutter with x *st*-pairs

Top Level Schnitt Europa



vs



Experimente - Pareto Balanciertheit

max ϵ	Achieved ϵ [%]			
[%]	F20	K	M	I
0	0.000	0.000	0.000	0.000
1	0.132	0.998	0.000	0.089
3	0.132	0.457	0.000	0.008
5	4.894	0.464	0.000	0.857
10	9.330	0.043	0.000	0.375
20	10.542	3.139	0.000	0.132
30	10.542	3.139	0.017	7.384
50	44.386	3.139	33.336	10.542
70	66.655	3.139	41.178	44.386
90	84.199	3.139	83.087	84.257

- Graph von Zentraleuropa (keine Insel Spezialfälle)
- FlowCutter kommt gewünschter Balanciertheit am nähsten

Experimente - Pareto Schnitt Größe

max ϵ	Cut Size			
[%]	F20	K	M	I
0	**240**	716	369	1 180
1	**220**	245	360	391
3	**220**	227	372	319
5	**213**	227	369	276
10	**180**	228	375	241
20	**162**	250	375	220
30	**162**	250	369	203
50	**155**	250	9 881	**162**
70	**86**	250	14 375	**155**
90	**13**	250	28	17

- Graph von Zentraleuropa (keine Insel Spezialfälle)
- FlowCutter liefert kleinste Schnitte

Andere Graphen (keine Straße)

- Beliebter Walshaw Datensatz:
- FlowCutter liefert beste bekannte Lösungen für Imbalance $= 5\%$
- FlowCutter liefert zusammenhängende Partitionen

$\rightarrow$ Nur solche Graphen wurden betrachtet

- 24 Graphen

Andere Graphen (keine Straße)

- Beliebter Walshaw Datensatz:
- FlowCutter liefert beste bekannte Lösungen für Imbalance $= 5\%$
- FlowCutter liefert zusammenhängende Partitionen

$\rightarrow$ Nur solche Graphen wurden betrachtet

- 24 Graphen

- Bei 18 Graphen perfekte Übereinstimmung mit Referenzlösung
- Bei 3 Graphen Fehler kleiner als 5 Kanten
- Laufzeit:
  - Schnell für kleine c
  - Wenn c in Größenordnung von m liegt wird Performance schlechter

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Literatur I

Daniel Delling, Andrew V. Goldberg, Ilya Razenshteyn, and Renato F. Werneck.
Graph Partitioning with Natural Cuts.
In *25th International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 1135–1146. IEEE Computer Society, 2011.

Michael Hamann and Ben Strasser.
Graph Bisection with Pareto-Optimization.
In *Proceedings of the 18th Meeting on Algorithm Engineering and Experiments (ALENEX'16)*. SIAM, 2016.

George Karypis and Gautam Kumar.
A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs.
SIAM Journal on Scientific Computing, 20(1):359–392, 1999.

Peter Sanders and Christian Schulz.
Distributed Evolutionary Graph Partitioning.
In *Proceedings of the 14th Meeting on Algorithm Engineering and Experiments (ALENEX'12)*, pages 16–29. SIAM, 2012.

Literatur II

Aaron Schild and Christian Sommer.
On Balanced Separators in Road Networks.
In *Proceedings of the 14th International Symposium on Experimental Algorithms (SEA'15)*, Lecture Notes in Computer Science. Springer, 2015.