Optimales Kanteneinfügen

- Mit und ohne Einbettungsconstraints -

Karsten Klein

Vorlesung

Automatisches Zeichnen von Graphen

WS 07/08 - 08. Januar 2008

TU Dortmund, Fakultät für Informatik,Ls11 Algorithm Engineering

Zusammenfassung von Montag

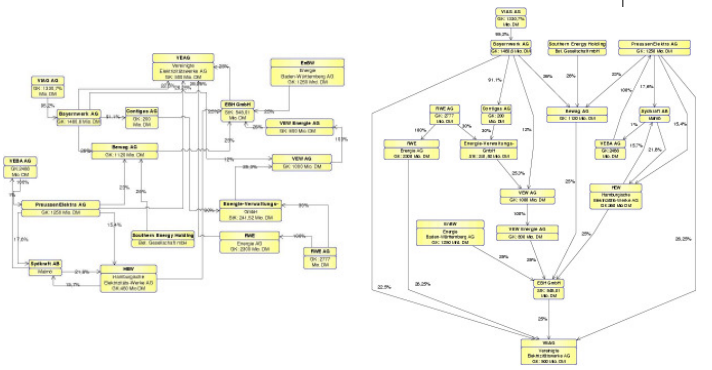
- Einschränkung der Einbettung durch drei grundlegende Constraints:
  Grouping, *Mirror*, *Oriented*
- Einbettungsconstraints an Knoten: Baum von Constraints
- ec-planare Einbettungen und ec-Planarität für (G,C)
- Transformation in ec-Expansion $E(G,C)$
- Planaritätstest mit Nebenbedingungen testet ec-Planarität für (G,C)
- Linearzeit

(Nicht)-Planarität

Wir haben tolle Planaritätstests! Einfach und in Linearzeit!
Wir haben tolle Zeichenverfahren für planare Graphen!

Was ist, wenn ein Graph nicht planar ist?

Wichtiges ästhetisches Kriterium: Kreuzungen
Wir würden den Graph gerne mit minimaler Anzahl Kreuzungen zeichnen!



(Nicht)-Planarität

Wir haben tolle Planaritätstests! Einfach und in Linearzeit!
Wir haben tolle Zeichenverfahren für planare Graphen!

Was ist, wenn ein Graph nicht planar ist?

Wichtiges ästhetisches Kriterium: Kreuzungen
Wir würden den Graph gerne mit minimaler Anzahl Kreuzungen zeichnen!

Kreuzungsminimierung ist NP-schwer
Bestes heuristisches Verfahren derzeit: Planarisierung

Planarisierung

Planarisierungsansatz:

- Lösche kleine Zahl von Kanten, so dass der Teilgraph planar ist (Problem des max. planaren Subgraphs)
- Füge iterativ Kanten unter Minimierung von Kreuzungen hinzu (Dummyknoten erhalten Planarität)

Wir erhalten einen planaren Graph, den wir zeichnen können.

Die Dummyknoten des Graphen werden zu Kreuzungen in der Zeichnung.

Wir betrachten hier nur das Einfügen von Kanten

Planarisierung

Einfache Idee für Einfügen von $e=(v,w)$ in Graph G:

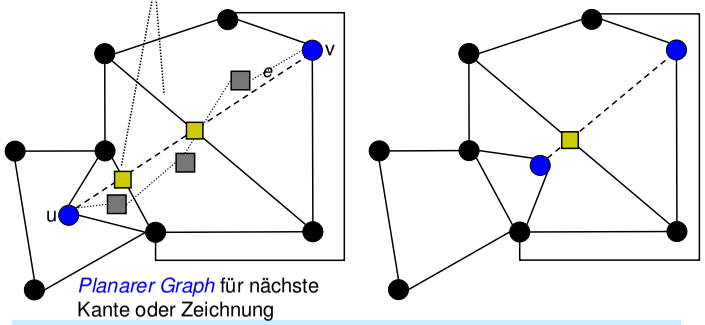
- Fixiere Einbettung von G
- Berechne Dualgraph
- Erweitere Dualgraph um v,w
- Berechne kürzesten Weg von v nach w
- Länge entspricht Anzahl der Kreuzungen

Kanteneinfügen

Durch u,v *augmentierter dualer Graph*

Einfügepfad für neue Kante e

Pfadlänge von Einbettung abhängig!



Planarer Graph für nächste Kante oder Zeichnung

Optimizing over all embeddings in linear time: [Gutwenger, Mutzel, Weiskircher `05]

Optimaler Einfügepfad

Definition: Optimaler Einfügepfad

Sei G zshg. planarer Graph und u,v nicht-adjazente Knoten in G. Ein Einfügepfad $p=e_1,\ldots,e_k$ für u und v in Einbettung Π von G heißt *optimaler Einfügepfad* für u und v in G, wenn es keinen kürzeren Einfügepfad für u und v in einer beliebigen Einbettung von G gibt.

Das Problem

Eingabe: Planarer Graph G mit zusätzlicher Kante e

Ausgabe: Kreuzungsminimale Zeichnung, bei der alle Kreuzungen auf e liegen

Alternativ: Finde kombinatorische Einbettung von G, in die e kreuzungsminimal eingefügt werden kann

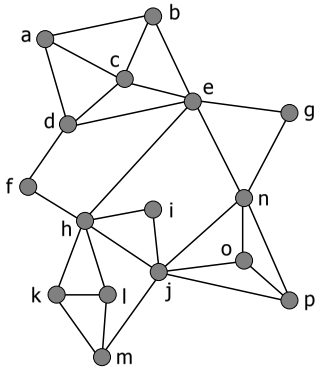
Viele Probleme, die über alle Einbettungen optimieren sind NP-schwer. Dieses nicht!

Optimales Kanteneinfügen

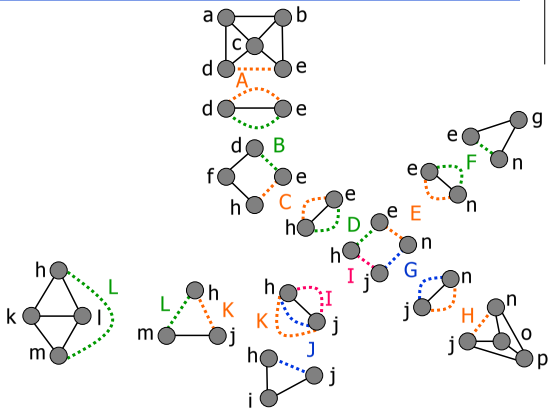
- Warum ist Kreuzungsminimierung dann NP-schwer?
- Reihenfolge bei mehreren Kanten
- Selbst bei Einfügen nur einer Kante muss Optimalität für Kreuzungsminimierung nicht erreicht werden!
- Nicht alle Kreuzungen müssen auf einer Kante liegen, d.h. G muss nicht zwingend planar gezeichnet sein

Wir benutzen SPQR-Bäume, um alle Einbettungen aufzuzählen

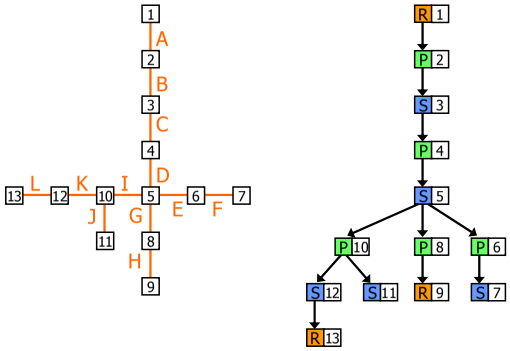
3-Zusammenhangskomponenten



3-Zusammenhangskomponenten

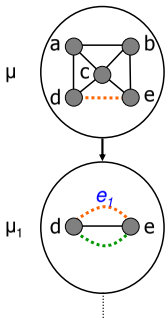


SPQR-Baum



Baum T(G) SPQR-Baum

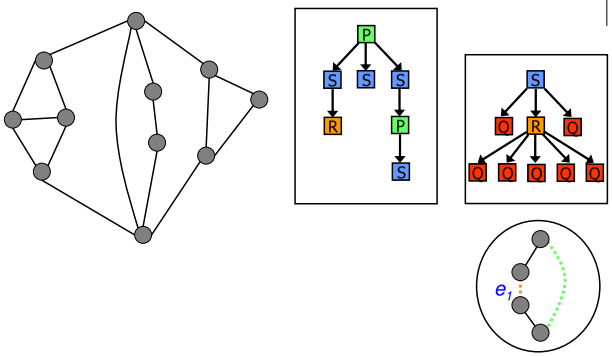
SPQR Baum



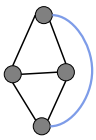
μ ist der *pertinente* Knoten von Kante e_1 in μ_1

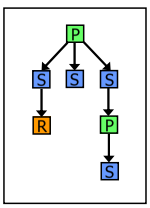
Ein Baumknoten, der den Graphknoten v im Skeleton enthält, heißt *Allokationsknoten* von v.

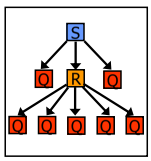
SPQR Baum



SPQR Baum

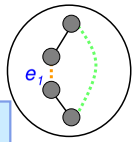






expansion(e_1)

expansion$^+$(e_1) = *expansion*(e_1) ∪ e_1

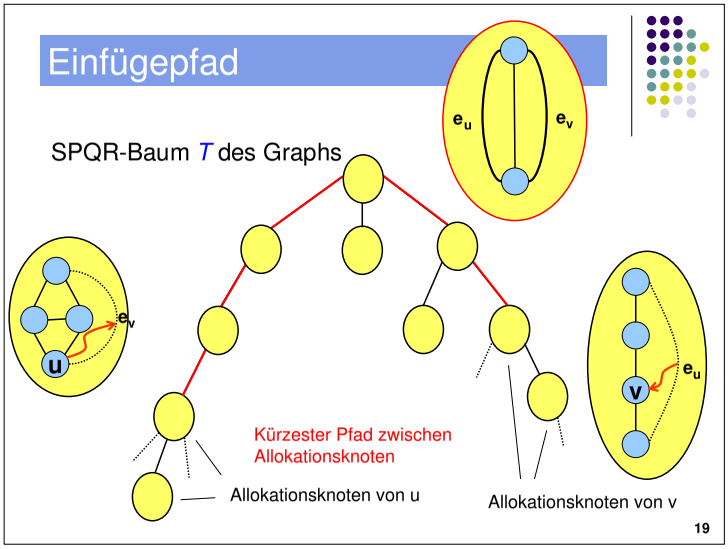


Der Expansionsgraph *expansion(e)* einer Skeletonkante e wird induziert durch die Kanten von G in Q-Knoten des Unterbaums ihres pertinenten Knotens.

Optimales Kanteneinfügen

- Wir wollen u und v verbinden
- Wir wollen dabei über alle Einbettungen optimieren
- Dazu können wir Einbettungen der Skeletons im SPQR-Baum T benutzen falls G 2-zusammenhängend
- u und v liegen im Skeleton von Allokationsknoten μ_1, μ_k

Einfügepfad

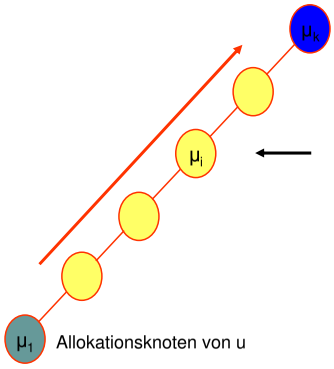


Optimales Kanteneinfügen

- Wir wollen *u* und *v* verbinden
- Wir wollen dabei über alle Einbettungen optimieren
- Dazu können wir Einbettungen der Skeletons im SPQR-Baum *T* benutzen falls *G* 2-zusammenhängend
- *u* und *v* liegen im Skeleton von Allokationsknoten μ_1, μ_k

- Wir laufen entlang des kürzesten Pfades im SPQR-Baum und traversieren den Graph

Kürzester Weg Traversierung



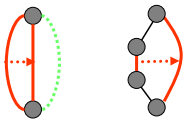
Lokale Einfügepfade:
Finde kürzesten Einfügepfad zwischen Repräsentanten von *u* und *v* in Skeletons der Knoten μ_i

Kürzester Weg Traversierung

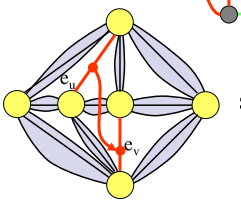
Lokale Einfügepfade:
Finde kürzesten Einfügepfad zwischen Repräsentanten von *u* und *v* in Skeletons der Knoten

S- und P-Knoten: Keine Kreuzungen!

R-Knoten:



Splitte Repräsentanten



Ersetze Kanten durch Expansionsgraph
Berechne Einbettung und kürzesten Weg ??

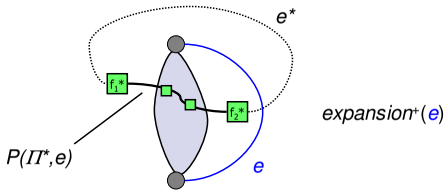
Offene Fragen

- Welche Kosten haben Skeletonkanten?
- Welche Einbettung der Expansionsgraphen?
- Wie berechnen wir kürzeste Wege???

Kosten von Skeletonkanten

Definition: Traversierungskosten

Für Skeletonkante *e* sei Π Einbettung von *expansion*$^+$(*e*) bzw. des Dualgraphs Π^*, f_1 und f_2 durch *e* getrennte Faces und $P(\Pi^*,e)$ der kürzeste Weg zwischen ihnen, der nicht e^* benutzt. Die *Traversierungskosten c(e)* sind definiert als Länge von $P(\Pi^*,e)$.



Kosten von Skeletonkanten

Lemma:

Sei μ ein Knoten in T und e Kante in *skeleton*(μ). Die Länge des Pfades $P(\Pi^*,e)$ ist unabhängig von der Einbettung Π von *expansion*$^+$(e).

Beweis:

Induktion über die Höhe h des Unterbaums T_e am pertinenten Knoten γ von e:

h=1: γ ist Q-Knoten, *expansion*$^+$(e) Kreis zweier Kanten mit einer einzigen Einbettung, *length*($P(\Pi^*,e)$) = 1

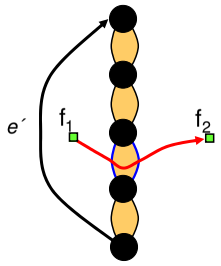
Kosten von Skeletonkanten

Beweis (Fortsetzung):

h>1: Wurzel γ von T_e ist S, P oder R Knoten. Sei e' die virtuelle Kante von μ in *skeleton*(γ).

S-Knoten:

Traversierungskosten



Kosten von Skeletonkanten

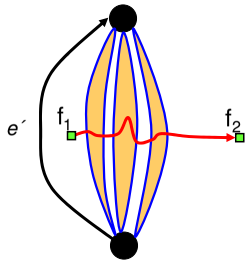
Beweis (Fortsetzung):

h>1: Wurzel γ von T ist S, P oder R Knoten. Sei e' die virtuelle Kante von μ in *skeleton*(γ).

S-Knoten: Skeleton ist ein Kreis und Länge von $P(\Pi^*,e)$ ist Minimum der Länge für Kreiskanten, nach IV unabhängig von Π.

P-Knoten:

Traversierungskosten



Kosten von Skeletonkanten

Beweis (Fortsetzung):

h>1: Wurzel γ von T ist S, P oder R Knoten. Sei e' die virtuelle Kante von μ in *skeleton*(γ).

S-Knoten: Skeleton ist ein Kreis und Länge von $P(\Pi^*,e)$ ist Minimum der Länge für Kreiskanten, nach IV unabhängig von Π.

P-Knoten: Skeleton aus parallelen Kanten $e', e_1, \ldots, e_l$, und Länge von $P(\Pi^*,e)$ ist die Summe der Längen $P(\Pi^*_{e_i}, e_i)$, unabhängig von Π.

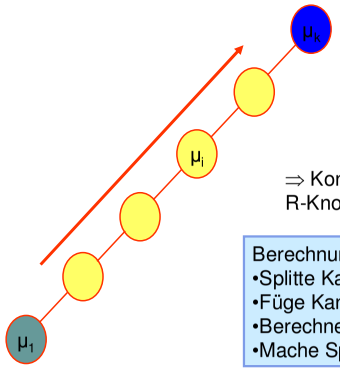
R-Knoten: Skeleton ist 3-zus. planarer Graph mit zwei Spiegeleinbettungen, Länge von $P(\Pi^*,e)$ deshalb unabhängig von Π.

Traversierungskosten

Traversierungskosten sind unabhängig von Einbettung

⇒ Beliebige Einbettung berechnen und kürzesten Pfad mit BFS berechnen. Das geht in Linearzeit.

Kürzester Weg Traversierung



⇒ Konkateniere lokale Einfügepfade der R-Knoten zu optimalem Einfügepfad *p*

Berechnung einer Einbettung aus *p*:
- Splitte Kanten e_i in p durch Knoten w_i
- Füge Kanten zwischen $u,w_1,...,w_l,v$ ein
- Berechne beliebige Einbettung (Graph planar)
- Mache Splitten rückgängig

Erweiterung für zshgd. Graphen

Wir benutzen den Block-Vertex Baum

Definition: Block-Vertex Baum

Seien $G=(V,E)$ und B Menge der Blöcke von G. Dann ist der Graph

$$(V \cup B, \{ (v,b) \mid v \in b \})$$

der *Block-Vertex Graph* $\mathcal{B}$ von G.

Der *Repräsentant* eines Knoten v in Block b ist entweder v wenn $v \in b$ oder der erste Knoten auf dem eindeutigen Pfad von b nach v in $\mathcal{B}$.

Kanteneinfügen

Graph G, Knoten u,v

Konstruiere Block-vertex Baum $\mathcal{B}$ von G

Bestimme Pfad $u,B_1,c_1,...,B_{k-1},c_{k-1},B_k,v$ von u nach v in $\mathcal{B}$

for i:=1,...,k do

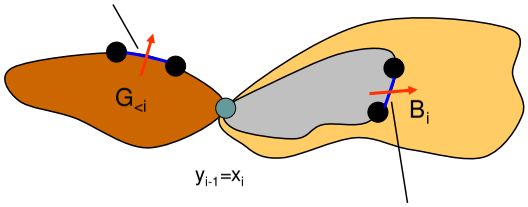
  Seien x_i und y_i Repräsentanten von u und v in B_i

  p_i:= OPTIMALBLOCKEINFÜGEN(B_i,x_i,y_i)

return p1+...+pk

Kombination von Teilpfaden

Letzte Kante in Einfügepfad zwischen u und y_{i-1}



Erste Kante in Einfügepfad zwischen x_i und v

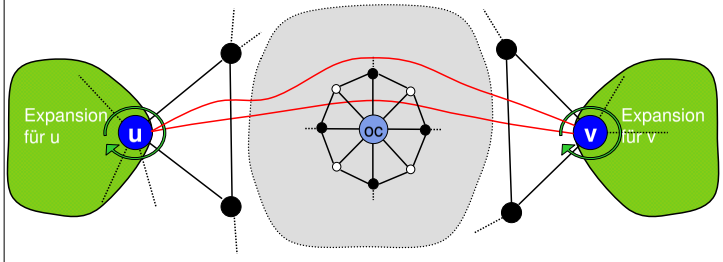
Kanteneinfügen mit Einbettungsconstraints

Kanteneinfügen mit Einbettungsconstraints

Nutze ec-Expansion für Kantenreihenfolge

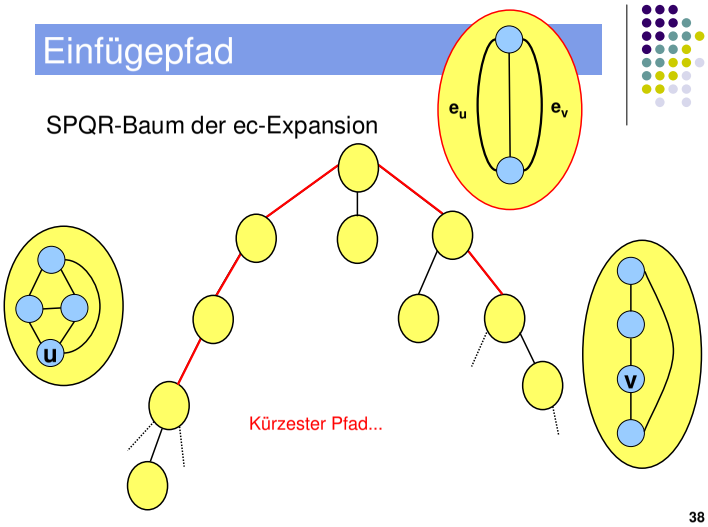
Expansionskanten nicht kreuzen

ec-Einfügepfad

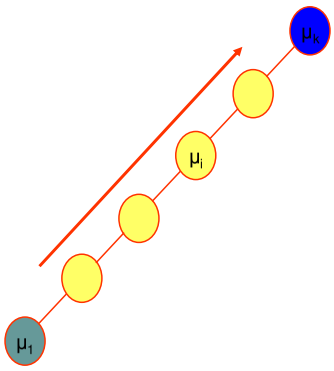


Einfügepfad

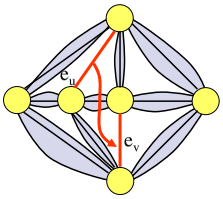
SPQR-Baum der ec-Expansion



Kürzester Weg Traversierung



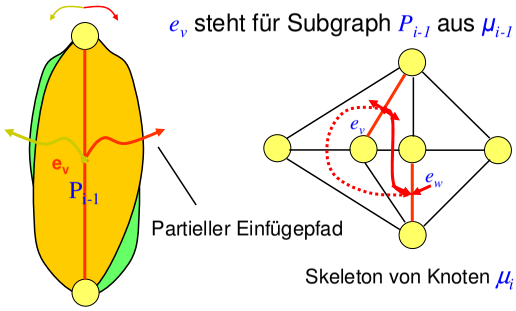
Lokaler Einfügepfad??



ec-planares Kanteneinfügen

Spiegeln

e_v steht für Subgraph P_{i-1} aus μ_{i-1}



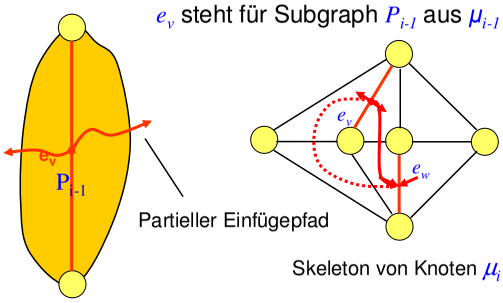
Partieller Einfügepfad

Skeleton von Knoten μ_i

Bisher: Einfügepfad in P_{i-1} bekannt, bei Bedarf (Verlassen nach rechts/links) spiegeln.

ec-planares Kanteneinfügen

e_v steht für Subgraph P_{i-1} aus μ_{i-1}



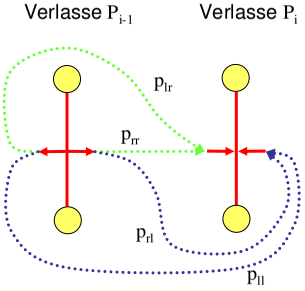
Partieller Einfügepfad

Skeleton von Knoten μ_i

Jetzt mit ec:Spiegeln nicht erlaubt!

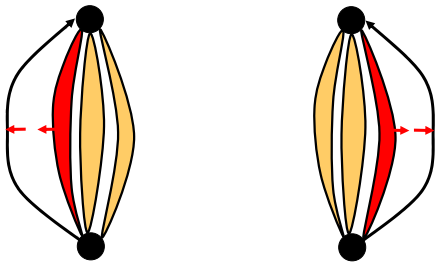
Beide Pfade können Teil des optimalen Einfügepfades sein!
Keine lokale Entscheidung mehr möglich!!

Lokale Einfügepfade



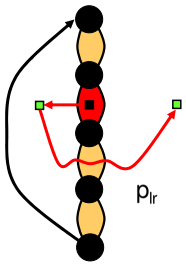
Zwei Wege um P_{i-1} zu verlassen, zwei Wege für P_i um in e_v zu laufen
Also vier mögliche Einfügepfade für μ_i: p_{ll},p_{lr},p_{rl},p_{rr}

Lokale Einfügepfade: P-Knoten

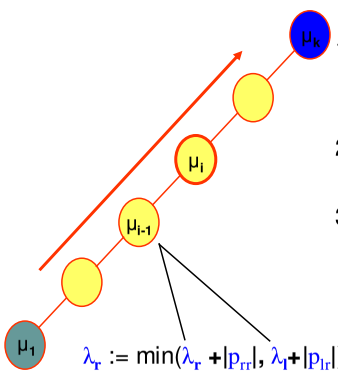


Weiterhin keine Kreuzung nötig, aber wir wissen die Richtung nicht

Lokale Einfügepfade: S-Knoten



Kürzester Weg Traversierung



An Knoten μ_i (S,P,R-Knoten):

1. Berechne Länge der Einfügepfade die nach rechts/links laufen: λ_r,λ_l
2. Speichere in welche Richtung μ_{i-1} verlassen wird: s^i_l,$s^i_r \in \{r, l\}$
3. Speichere lokalen Einfügepfad der beim Verlassen nach rechts/links benutzt wird p^i_l, p^i_r

$\lambda_r := \min(\lambda_r + |p_{rr}|, \lambda_l + |p_{lr}|)$

Berechnung des optimalen Einfügepfades

An Knoten μ_k berechne Pfad rückwärts aus gespeicherten Werten (Hier gilt $\lambda_r=\lambda_l$, da alle Faces um *v* erlaubt).

$s_k := l$
for i := k **downto** 1 **do**
  $p_i := p^i_{s_i}$
  $s_{i-1} := s^i_{s_i}$
end for
return $p_1+...+p_k$

ec-planares Kanteneinfügen

- Berechnet einen optimalen ec-Einfügepfad für Knoten *u* und *v* in Block *B* in Zeit O(|E|):
  SPQR-Baum
  BFS auf Teilgraphen von *G*+virtuelle Kanten
- Kann leicht für zshgd. Graphen erweitert werden wie im nicht-Constraint Fall auch

Zusammenfassung

- Kreuzungsoptimales Einfügen einer einzelnen Kante in Linearzeit
- Nutzt SPQR-Bäume
- Berechnung lokaler Einfügepfade und Konkatenation
- Auch für ec in Linearzeit möglich, aber umständlicher

WANTED:

Diplomand für Entwicklungen rund um die Embedding Constraints

- Finden von ec-planaren Untergraphen
- Umsetzung eines Planarisierungsverfahrens mit ec
- Implementierung in OGDF
- Erweiterung...