technische universität dortmund

ALGORITHM ENGINEERING

Biconnectivity Augmentation

Vorlesung Graphenalgorithmen
WS 2009/2010

Überblick

- Grundlagen
- Problemdefinition(en)
- Motivation
- Biconnectivity Augmentation
  - Algorithmus von Hsu & Ramachandran
  - Algorithmus von Hsu
- Planar Biconnectivity Augmentation
  - NP-Vollständigkeitsbeweis
  - 2-Approximation

2-Zusammenhang, Schnittknoten

Definition Schnittknoten (*cutvertex*)

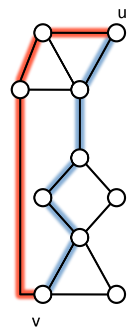
v ∈ V ist Schnittknoten falls G-{v} mehr Zusammenhangskomponenten hat als G

Definition 2-zusammenhängend (*biconnected*)

zshgd. Graph G=(V,E) ist 2-zshgd. gdw. es existiert kein Schnittknoten

alternativ:

∀u,v∈V: ∃2 knotendisjunkte Pfade zwischen u und v



2-Zusammenhang, Brücke

Definition 2-(Knoten-)zusammenhängend (*(vertex-)biconnected*)

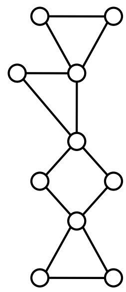
zshgd. G=(V,E) ist 2-(Knoten)-zshgd. gdw. es existiert kein Schnittknoten

Definition 2-Kanten-zusammenhängend (*edge-biconnected*)

zshgd. G=(V,E) ist 2-Kanten-zshgd. gdw. es existiert keine Kante e ∈ E so dass G-{e} zerfällt. So eine Kante nennt man Brücke (*bridge*).

Definition Brücken-zusammenhängend (*bridge-connected*)

zshgd. G=(V,E) ist bridge-connected gdw. es existiert keine Brücke



also: bridge-connected = edge-biconnected

Beobachtung: vertex-biconnected ≠ bridge-connected

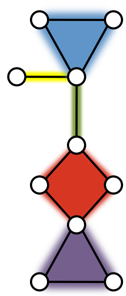
es gilt: G vertex-biconnected ⇒ G bridge-connected

2-Zusammenhangskomponenten

Definition 2-(Knoten-)zusammenhängend (*(vertex-)biconnected*)

zshgd. G=(V,E) ist 2-(Knoten)-zshgd. gdw. es existiert kein Schnittknoten

Definition 2-Zusammenhangskomponenten (*biconnected components*): maximale 2-zsgd. Teilgraphen; auch Blöcke genannt



Biconnectivity Augmentation Problem

Definition Biconnectivity Augmentation Problem

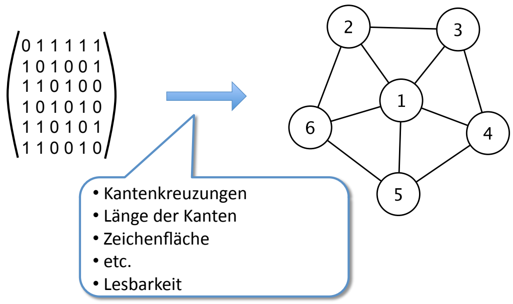
gegeben: G=(V,E) ungerichtet

gesucht: minimale Kantenmenge E'⊆VxV so dass G'=(V, E∪E') 2-zshgd.

- lösbar in O(|V| + |E|)
  → Algorithmus in dieser VL
- einige Varianten:
  - Kanten-Zusammenhang
  - Augmentierung zum k-Zusammenhang
  - Planare Augmentierung
    - planarer Graph gegeben, neue Kanten müssen Planarität erhalten
  - gewichtete Augmentierung
  - gerichtete Augmentierung, d.h. starker Zshg.

Motivation

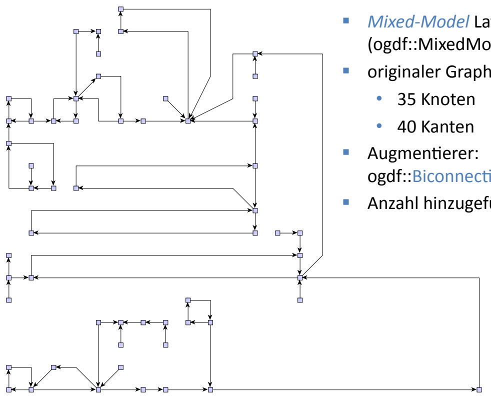
- Graph-Drawing
  - automatische Berechnung eines Layouts



Motivation

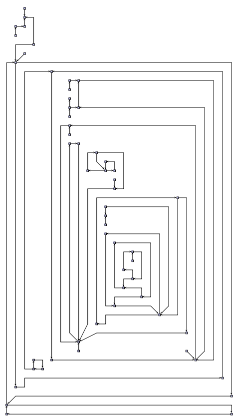
- Graph-Drawing
  - automatische Berechnung eines Layouts
- einige Zeichenalgorithmen benötigen 2-zshgd. Graphen, z.B.
  - *Mixed Model*
  - *Planar Straight-Line*
- Vorgehen, falls Graph nicht 2-zshgd.:
  - finde Augmentierung
  - füge Kanten hinzu
  - berechne Layout
  - entferne Augmentierungs-Kanten, Knotenpositionen unverändert
- **Ziel**: möglichst wenige Kanten hinzufügen, da diese das Layout beeinflussen

Graph Drawing - Beispiel I



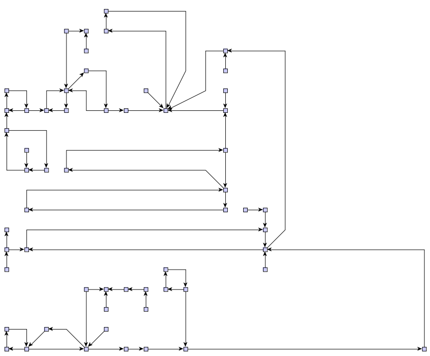
- *Mixed-Model* Layout (ogdf::MixedModelLayout)
- originaler Graph:
  - 35 Knoten
  - 40 Kanten
- Augmentierer: ogdf::BiconnectivityAugmentation
- Anzahl hinzugefügter Kanten: 8

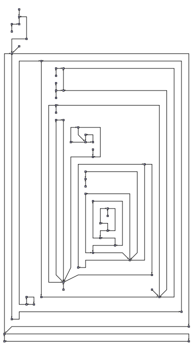
Graph Drawing - Beispiel I



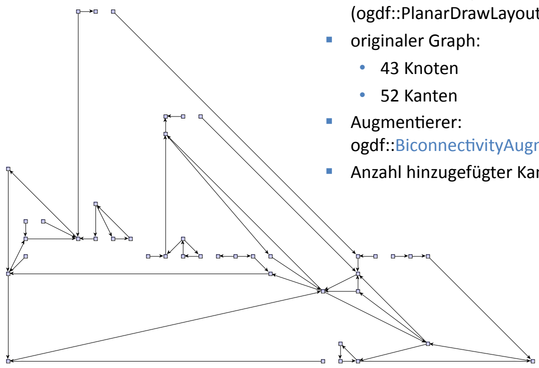
- *Mixed-Model* Layout (ogdf::MixedModelLayout)
- originaler Graph:
  - 35 Knoten
  - 40 Kanten
- Augmentierer: ogdf::DfsMakeBiconnected
- Anzahl hinzugefügter Kanten: 27

Vergleich I



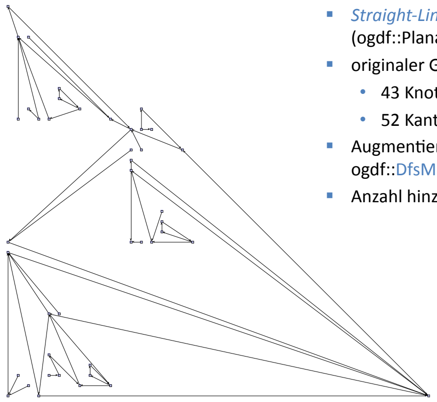


Graph Drawing - Beispiel II



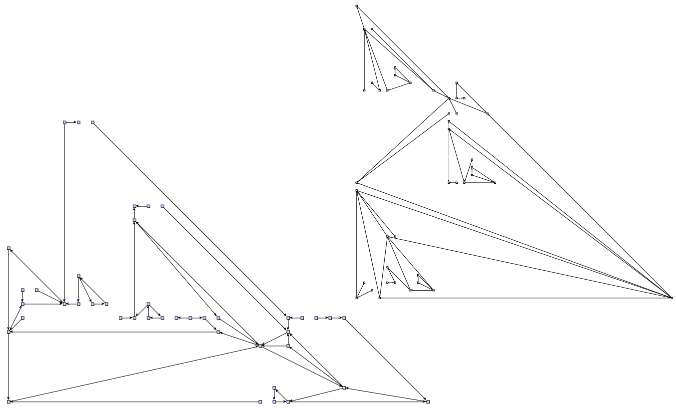
- *Straight-Line* Layout (ogdf::PlanarDrawLayout)
- originaler Graph:
  - 43 Knoten
  - 52 Kanten
- Augmentierer: ogdf::BiconnectivityAugmentation
- Anzahl hinzugefügter Kanten: 8

Graph Drawing - Beispiel II



- *Straight-Line* Layout (ogdf::PlanarDrawLayout)
- originaler Graph:
  - 43 Knoten
  - 52 Kanten
- Augmentierer: ogdf::DfsMakeBiconnected
- Anzahl hinzugefügter Kanten: 30

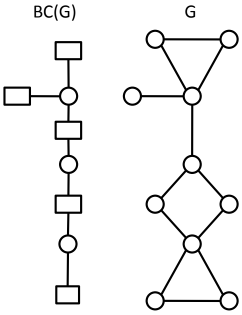
Vergleich II



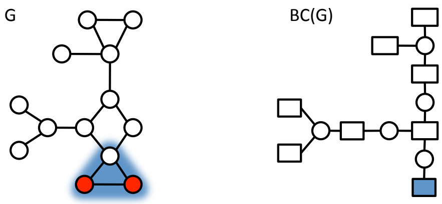
BC-Baum

Definition BC-Baum BC(G) = (V_{bc}=($V_b \cup V_c$), E_{bc}) zu Graph G=(V,E)

- V_b: ein B-Knoten für jeden Block
- V_c: ein C-Knoten für jeden Schnittknoten
- E_{bc}: Kante zwischen c $\in V_c$ und b $\in V_b$ gdw. Schnittknoten zu c liegt im Block zu b
- gewurzelt an einem beliebigen Knoten v $\in V_{bc}$, oft: B-Knoten mit Grad ≥ 2
- Aufbau/Größe O(|V|+|E|)
- ist G nicht zshdg. ⇒ BC(G) ist ein Wald
- **Definition** Knoten mit Grad 1 in BC(G) heißt Blatt
- ℓ := # Blätter in BC(G)

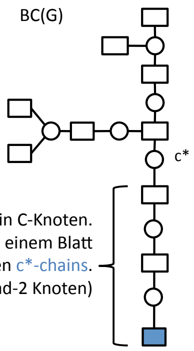


BC-Baum - Definitionen



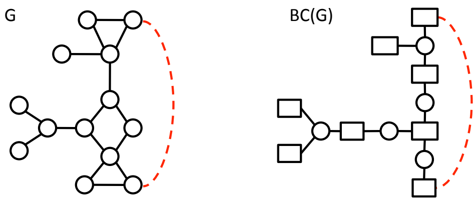
Definition Knoten einer 2-Zshgs.-komponente, welche keine Schnittknoten sind, heißen einfache Knoten

BC-Baum - Definitionen

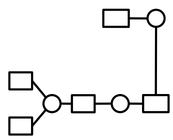


Definition sei c* ein C-Knoten. Teilbäume von c* mit einem Blatt heißen c*-chains. (Pfade mit Grad-2 Knoten)

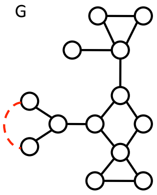
BC-Baum - Beispiel

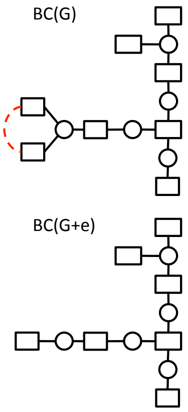


- *profitable* Kante
- Blätter *billig* verbunden



BC-Baum - Beispiel





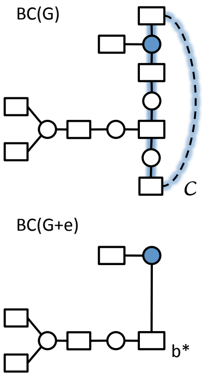
- *unprofitable* Kante
- Blätter *teuer* verbunden

Einfügen einer Kante

Beobachtung sei G=(V,E) zshgd., BC(G) der BC-Baum und $b_1 \neq b_2$ zwei Blätter in BC(G). Sei C der eindeutige Kreis in BC(G) zwischen b_1 und b_2 plus die Kante (b_1,b_2). Sei e eine neue Kante zwischen 2 einfachen Knoten in b_1 und b_2 und G' = G+e.

Dann gilt für BC(G') in Bezug auf BC(G):

1. Knoten und Kanten nicht auf C bleiben unverändert
2. Alle B-Knoten auf C werden kontrahiert zu einem B-Knoten b*
3. C-Knoten auf C mit Grad 2 werden eliminiert
4. C-Knoten c* mit deg(c*) > 2 bleiben bestehen mit Kanten inzident zu den B-Knoten nicht auf C und zu b*.
   Es folgt dass deg'(c*) = deg(c*) -1



profitable Kante

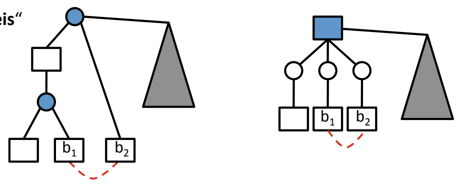
Definition leaf-connection condition (lcc)

2 Blätter b_1, b_2 in BC(G) erfüllen die leaf-connection condition gdw. der Pfad zwischen b_1 und b_2 entweder

1. 2 Knoten mit Grad ≥ 3 oder
2. 1 B-Knoten mit Grad ≥ 4 enthält.

Lemma erfüllen 2 Blätter b_1, b_2 die lcc dann hat $BC(G+(b_1,b_2))$ 2 Blätter weniger als BC(G)

„Beweis"



Untere Schranke (Eswaran, Tarjan[1976])

Lemma Man benötigt mindestens

$\max \{ d+h-2, \lceil \ell/2 \rceil + q \}$

Kanten um den 2-Zshg. herzustellen

mit

- d: max Grad eines C-Knotens
- h: # Zusammenhangskomponenten
- ℓ: # Blätter in BC(G)
- q: # isolierte Blöcke (isolierter Knoten ist ein Block)
- $\ell + q > 1$

Untere Schranke (Eswaran, Tarjan[1976])

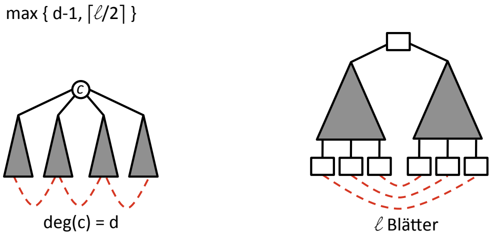
Lemma Man benötigt mindestens

$\max \{ d+h-2, \lceil \ell/2 \rceil + q \}$

Kanten um den 2-Zshg. herzustellen

Beweis

a) Betrachte zshgd. Graphen: h=1, q=0
   $\Rightarrow \max \{ d-1, \lceil \ell/2 \rceil \}$



Untere Schranke (Eswaran, Tarjan[1976])

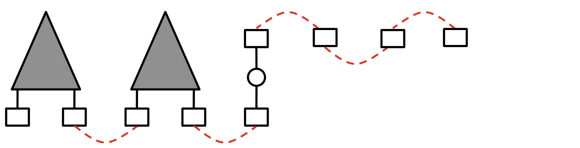
Lemma Man benötigt mindestens

$\max \{ d+h-2, \lceil \ell/2 \rceil + q \}$

Kanten um den 2-Zshg. herzustellen

Beweis

b) Betrachte nicht-zshgd. Graphen: h ≥ 2, q ≥ 0
   1. mache Graph zsghd. durch:
      - nummeriere Zshgs.-Komponenten von BC(G): $Z_1,\dots,Z_h$
      - verbinde Blätter von Z_i und Z_{i+1} (i=1,...,h-1), wobei Kante j und j+1 Block nur gemeinsam haben, wenn Z_{j+1} isolierter B-Knoten



Untere Schranke (Eswaran, Tarjan[1976])

Lemma Man benötigt mindestens

max { d+h-2, ⌈ℓ/2⌉+q }

Kanten um den 2-Zshg. herzustellen

Beweis

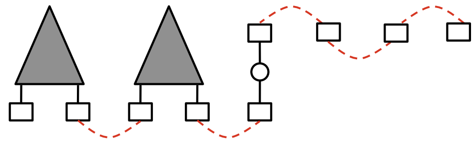
b) Betrachte nicht-zshgd. Graphen: h ≥ 2, q ≥ 0

1. mache Graph zsghd. durch:
   - nummeriere Zshgs.-Komponenten von BC(G): $Z_1,\ldots,Z_h$
   - verbinde Blätter von Z_i und Z_{i+1} (i=1,...,h-1), wobei Kante j und j+1 Block nur gemeinsam haben, wenn Z_{j+1} isolierter B-Knoten
2. verwende Schranke aus a)
   - #eingefügte Kanten in 1. = h-1
   - #Blätter nachher: ℓ' = ℓ + 2q - 2(h-1) (noch offen)
   - Knotengrade innerer Knoten unverändert
   - h-1 + max { d-1, ⌈ℓ'/2⌉ }

     = max { d+h-2, ⌈(ℓ+2q-2h+2)/2⌉ +h-1} = max {d+h-2, ⌈ℓ/2⌉ + q}

Zusammenhang herstellen

noch offen: #Blätter nachher: ℓ' = ℓ + 2q - 2(h-1)



für q=0: ℓ' = ℓ - 2(h-1)

für q>0: ℓ = 0: ℓ' = 2

ℓ = 0, h = q, also ℓ' = 2q - 2(h-1) = 2

ℓ > 0: sei p := h - q

nach den ersten p-1 Kanten ⇒ ℓ'' = ℓ - 2(p-1)

nach den nächsten q Kanten ⇒ ℓ' = ℓ''

insgesamt: ℓ' = ℓ'' = ℓ - 2(p-1) = ℓ - 2(h-q-1) □

Untere/Obere Schranke

Lemma Man benötigt ~~mindestens~~ exakt

max { d+h-2, ⌈ℓ/2⌉+q }

Kanten um den 2-Zshg. herzustellen

Beweis

- durch Angabe eines Algorithmus
- ...allerdings noch ein wenig Vorarbeit notwendig

Algorithmus Hsu & Ramachandran [1993]

3 Phasen:

Phase 1: *Input*: ungerichteter Graph

Output: zusammenhängender Graph

⇒ schon gesehen, Optimalität bleibt erhalten

Phase 2: *Input*: zusammenhängender Graph

Output: zshgd. und balancierter Graph

⇒ unprofitable Kanten

Phase 3: *Input*: zshgd. und balancierter Graph

Output: 2-zshgd. Graph

⇒ ausschließlich profitable Kanten

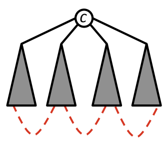
massive C-Knoten

Lemma Man benötigt exakt

max { d-1, ⌈ℓ/2⌉ }

Kanten um den 2-Zshg. herzustellen

(bei zshgd. Graph)



deg(c)-1

Definition ein C-Knoten c* ist massiv

gdw. deg(c*) ≥ ⌈ℓ/2⌉ +2

sei c* massiv ⇒ deg(c*) - 1 ≥ ⌈ℓ/2⌉ +1 Kanten benötigt

Definition Wenn BC(G) keinen massiven Knoten enthält so ist er balanciert. Dann nennen wir auch G balanciert.

Beobachtung BC(G) hat max. 1 massiven C-Knoten

Beweis Übung

Algorithmus Hsu & Ramachandran [1993]

3 Phasen:

Phase 1: *Input*: ungerichteter Graph

Output: zusammenhängender Graph

⇒ schon gesehen, Optimalität bleibt erhalten

Phase 2: *Input*: zusammenhängender Graph

Output: zshgd. und balancierter Graph

⇒ unprofitable Kanten

Phase 3: *Input*: zshgd. und balancierter Graph

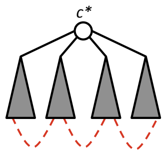
Output: 2-zshgd. Graph

⇒ ausschließlich profitable Kanten

Algorithmus - Phase 2

sei c* massiv ⇒ deg(c*) ≥ ⌈ℓ/2⌉ +2
sei δ := deg(c*) - 1 - ⌈ℓ/2⌉

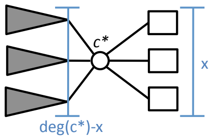
Lemma für einen massiven c-Knoten c* existieren
2 δ + 1 c*-chains



deg(c*) ≥ ⌈ℓ/2⌉ +2

Beweis

- sei x Anzahl an c*-chains
- # Blätter im BC-Baum:
  2(deg(c*) - x) + x ≤ ℓ
  x ≥ 2deg(c*) - ℓ
  x ≥ 2(δ + 1 + ⌈ℓ/2⌉) - ℓ
  x ≥ 2δ + 2 ⌈ℓ/2⌉ - ℓ + 2
  x ≥ 2δ + 2
  x ≥ 2δ + 1 □



Algorithmus - Phase 2

sei c* massiv ⇒ deg(c*) ≥ ⌈ℓ/2⌉ +2
sei δ := deg(c*) - 1 - ⌈ℓ/2⌉

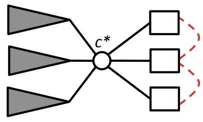
Lemma für einen massiven c-Knoten c* existieren
2 δ + 1 c*-chains

balanciere BC-Baum:

- kontrahiere c*-chains durch Einfügen von 2δ Kanten

⇒ c* ist anschließend nicht mehr massiv:

$$\begin{array}{ccc} \deg'(c^*) - 1 & = & \lceil \ell'/2 \rceil \\ = & & = \\ \deg(c^*) - 2\delta - 1 & = \lceil \ell/2 \rceil - \delta = & \lceil (\ell - 2\delta)/2 \rceil \end{array}$$

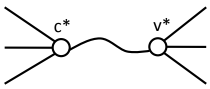


- falls balancierter Graph mit ⌈ℓ/2⌉ Kanten augmentiert werden kann ⇒ optimal,
denn: #eingefügte Kanten: 2δ + ⌈ℓ'/2⌉ = 2δ + ⌈(ℓ-2δ)/2⌉ = δ + ⌈ℓ/2⌉ = deg(c*) - 1
(aber: es darf kein anderer massiver C-Knoten entstanden sein!)

Algorithmus - Phase 2

z.Z.: nach Phase 2 existiert kein massiver C-Knoten:

- bereits gesehen: c* ist nicht mehr massiv
- Annahme: C-Knoten v* ≠ c* ist massiv
  ⇒ deg'(v*) ≥ ⌈ℓ'/2⌉ + 2
  ⇒ ℓ' ≥ deg'(v*) + deg'(c*) - 2
  ≥ ⌈ℓ'/2⌉ + 2 + ⌈ℓ'/2⌉ + 1 - 2
  > ℓ'



Algorithmus Hsu & Ramachandran [1993]

3 Phasen:

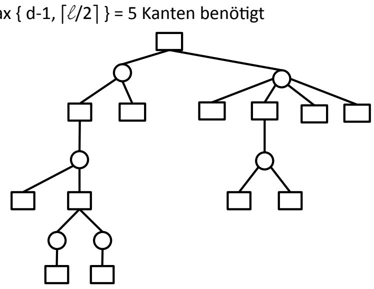
Phase 1: *Input*: ungerichteter Graph
Output: zusammenhängender Graph
⇒ schon gesehen, Optimalität bleibt erhalten

Phase 2: *Input*: zusammenhängender Graph
Output: zshgd. und balancierter Graph
⇒ unprofitable Kanten, Optimalität bleibt erhalten

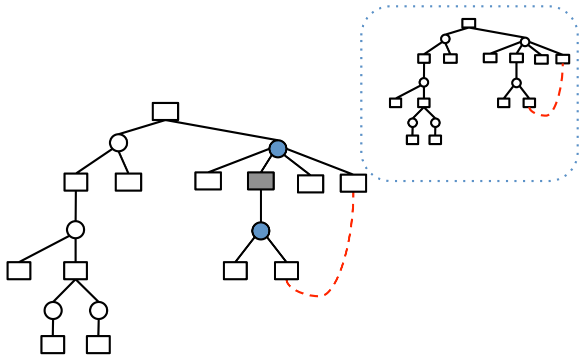
Phase 3: *Input*: zshgd. und balancierter Graph
Output: 2-zshgd. Graph
⇒ ausschließlich profitable Kanten

Phase 3 - Beispiel

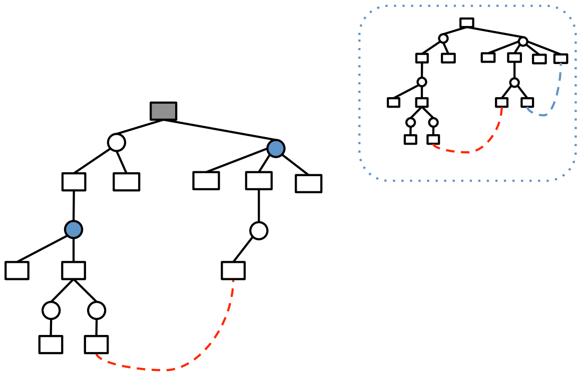
ℓ = 9, d = 5
⇒ balancierter BC-Baum/Graph
⇒ max { d-1, ⌈ℓ/2⌉ } = 5 Kanten benötigt



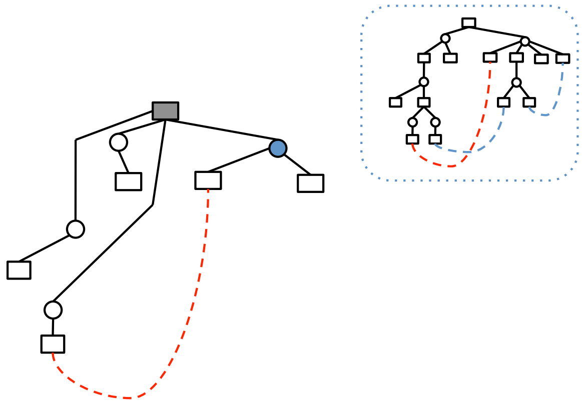
Phase 3 - Beispiel



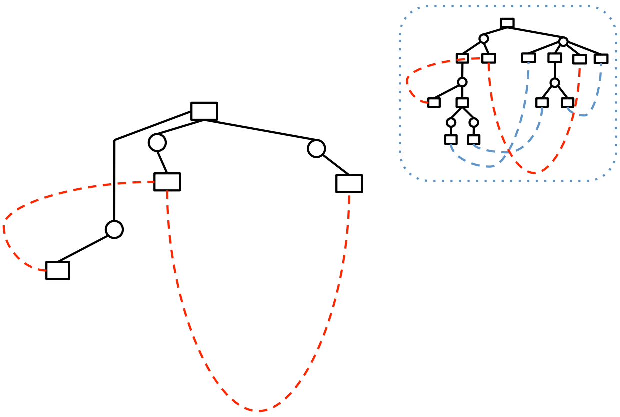
Phase 3 - Beispiel



Phase 3 - Beispiel



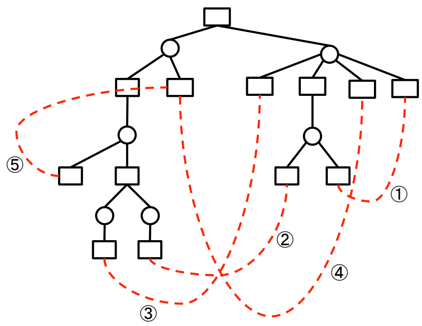
Phase 3 - Beispiel



Phase 3 - Beispiel - Endergebnis

$\ell = 9$, $d = 5$

$\Rightarrow \max\{\, d-1, \lceil \ell/2 \rceil \,\} = 5$



Algorithmus - Phase 3

Input: zshgd. und balancierter Graph G, BC-Baum BC(G), (aktuelle) Anzahl Blätter in BC(G): ℓ
Output: 2-zshgd. Graph G
while ($\ell > 3$) do

    if (es existiert C-Knoten mit Grad > 2)
        v := wähle C-Knoten mit max. Grad
    else
        v := wähle B-Knoten mit max. Grad

    if (es existiert C-Knoten ≠ v mit Grad > 2)
        w := wähle C-Knoten ≠ v mit max. Grad
    else
        if (es existiert B-Knoten ≠ v mit Grad > 2)
            w := wähle B-Knoten ≠ v mit max. Grad
        else w := v

    finde 2 Blätter b_1 und b_2 wobei der Pfad durch v und w verläuft
    G := G+(b_1,b_2)
    update BC(G)
end while
verbinde die restlichen ℓ Blätter mit ℓ-1 Kanten
return G

Analyse - Optimalität

Lemma für zshgd. Graph G mit balanciertem BC-Baum BC(G) werden $\lceil \ell/2 \rceil$ Kanten eingefügt. Anschließend ist G 2-zshgd.

Beweis

- 2-Zusammenhang: klar
- $\lceil \ell/2 \rceil$ Kanten:
  - eingefügte Kante erfüllt leaf-connection condition
    $\Rightarrow$ #Blätter sinkt um 2
  - falls BC(G) balanciert bleibt
    $\Rightarrow$ Induktion über ℓ $\Rightarrow$ optimal
  - Induktionsanfang: ℓ=2, ℓ=3 ✓
  - z.Z.: es entsteht kein massiver C-Knoten
    - Annahme: es existiert c* mit deg'(c*) ≥ $\lceil \ell'/2 \rceil + 2$
    - Fall 1: c* lag auf dem Einfügepfad
    - Fall 2: c* lag nicht auf dem Einfügepfad

Analyse - Optimalität

Lemma für zshgd. Graph G mit balanciertem BC-Baum BC(G) werden $\lceil \ell/2 \rceil$ Kanten eingefügt. Anschließend ist G 2-zshgd.

Beweis z.Z.: es entsteht kein massiver C-Knoten

- Annahme: es existiert c* mit deg'(c*) $\geq \lceil \ell'/2 \rceil + 2$
- Fall 1: c* lag auf dem Einfügepfad

  $\Rightarrow$ deg'(c*) = deg(c*) - 1

  $\Rightarrow$ deg(c*) = deg'(c*) + 1 $\geq \lceil \ell'/2 \rceil + 3 = \lceil (\ell-2)/2 \rceil + 3 = \lceil \ell/2 \rceil + 2$

  $\Rightarrow$ c* war vorher schon massiv

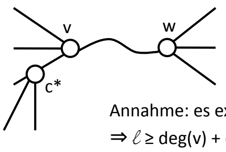
Analyse - Optimalität

Lemma für zshgd. Graph G mit balanciertem BC-Baum BC(G) werden $\lceil \ell/2 \rceil$ Kanten eingefügt. Anschließend ist G 2-zshgd.

Beweis z.Z.: es entsteht kein massiver C-Knoten

- Annahme: es existiert c* mit deg'(c*) $\geq \lceil \ell'/2 \rceil + 2$
- Fall 2: c* lag nicht auf dem Einfügepfad

  $\Rightarrow$ deg'(c*) = deg(c*)
- da c* nicht auf dem Einfügepfad lag, muss es 2 Knoten v, w geben mit deg(v) $\geq$ deg(w) $\geq$ deg(c*) $\geq \lceil \ell'/2 \rceil + 2 = \lceil \ell/2 \rceil + 1$ (vom Algorithmus gewählt)



$\ell \geq$ deg(v) + deg(w) -2

$\geq \lceil \ell/2 \rceil + 1 + \lceil \ell/2 \rceil + 1 - 2$

$= 2 \lceil \ell/2 \rceil$

$\Rightarrow \ell$ gerade und deg(v) = deg(w) = $\ell/2 + 1$

Annahme: es existiert C-Knoten c* mit deg(c*) ≥ 3

$\Rightarrow \ell \geq$ deg(v) + deg(w) - 2 + deg(c*) - 2 $\geq 2(\ell/2) + 1 > \ell$ $\square$

Analyse - Laufzeit

Lemma Der Algorithmus von Hsu&Ramachandran hat Laufzeit O(|V| + |E| + |V|α(|V|))

Beweis

- BC-Baum: Aufbau O(|V| + |E|)
- C-Knoten sortieren: Bucket-Sort, O($|V_{bc}|$)
  - Sortierung updaten: insgesamt O($|V_{bc}|$)
- Blätter finden: O(|Pfad|)
  - anschließend: B-Knoten auf Pfad kontrahiert

$\Rightarrow$ jeder Pfad wird nur einmal betrachtet

$\Rightarrow$ ein C-Knoten c* wird nur deg(c*) mal betrachtet

- BC-Baum updaten:
  - mit Hilfe von UNION-FIND Datenstruktur auf B-Knoten

    $\Rightarrow$ insgesamt O(|V| + |E| + |V|α(|V|)) $\square$

α: inverse Ackermann-Funktion

optimal?

Moment!

- Biconnectivity Augmentation Problem lösbar in O(|V| + |E|)

  → Algorithmus in dieser VL

  → O(|V| + |E| +|V|α(|V|)) ≠ O(|V| + |E|)

- Algorithmus von Hsu [2002]:
  - Laufzeit O(|V| + |E|)
  - keine dynamischen Updates des BC-Baums notwendig
  - Idee einfacher...
  - ... aber Beweis wesentlich komplizierter

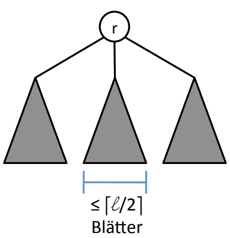
  → Grundidee des Algorithmus

Algorithmus von Hsu [2002]

Idee

- BC-Baum umwurzeln, so dass Einfügen von Kanten keine Updates erfordert

Definition BC-Baum mit Wurzelknoten r ist normalisiert gdw. es existiert kein Teilbaum (Zweig) von r mit > $\lceil \ell/2 \rceil$ Blättern



Lemma

(a) In jedem (BC-)Baum existiert ein Knoten r so dass der an r gewurzelte (BC-)Baum normalisiert ist.

(b) So eine Wurzel lässt sich in linearer Zeit finden.

Beweis Übung

Algorithmus von Hsu [2002]

3 Phasen:

Phase 1: wie bei Hsu&Ramachandran[1993]: Zusammenhang herstellen

Phase 2: wie bei Hsu&Ramachandran[1993]: Massive C-Knoten entfernen

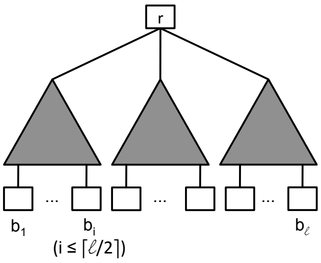
Phase 3:

- normalisiere BC-Baum, sei r die Wurzel
- Fall 1: r ist ein B-Knoten
  - füge Kantenmenge E_1 hinzu
- Fall 2: r ist ein C-Knoten
  - füge Kantenmenge E_2 hinzu

Fall 1 - B-Knoten

BC(G) mit Wurzel r, r ∈ V_b, normalisiert

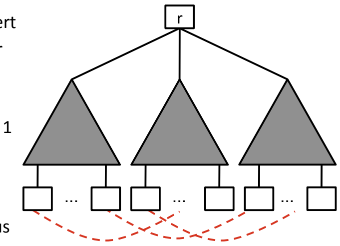
- nummeriere Blätter nach Preorder $b_1,\dots,b_\ell$
- $E_1 := \{(b_i, b_{i+\lfloor \ell/2 \rfloor}) \mid i = 1,\dots, \lceil \ell/2 \rceil\}$



Fall 1 - B-Knoten

BC(G) mit Wurzel r, r ∈ V_b, normalisiert

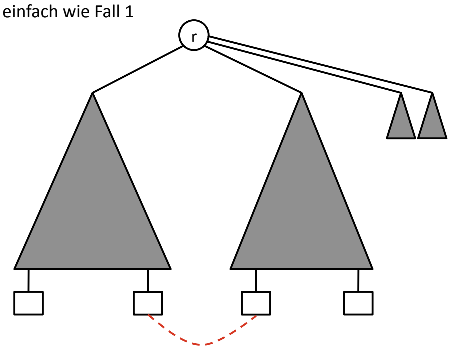
- nummeriere Blätter nach Preorder $b_1,\dots,b_\ell$
- $E_1 := \{(b_i, b_{i+\lfloor \ell/2 \rfloor}) \mid i = 1,\dots, \lceil \ell/2 \rceil\}$
- jedes Blatt wird durch mindestens 1 Kante verbunden
  - falls ℓ ungerade: $b_{\lceil \ell/2 \rceil}$ wird 2 mal verbunden
- jedes Blatt wird mit einem Blatt aus einem anderen Teilbaum verbunden
- offensichtlich 2-zshgd.



Fall 2 - C-Knoten

BC(G) mit Wurzel r, r ∈ V_c, normalisiert

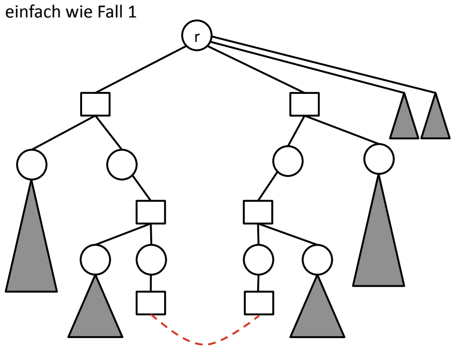
- bei weitem nicht so einfach wie Fall 1



Fall 2 - C-Knoten

BC(G) mit Wurzel r, r ∈ V_c, normalisiert

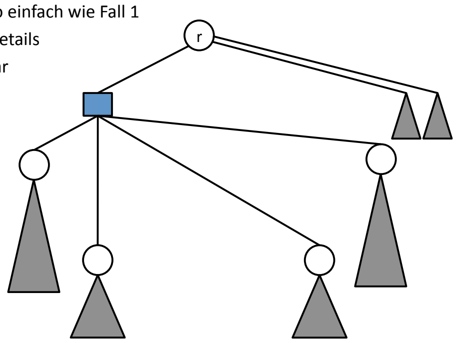
- bei weitem nicht so einfach wie Fall 1



Fall 2 - C-Knoten

BC(G) mit Wurzel r, r ∈ V_c, normalisiert

- bei weitem nicht so einfach wie Fall 1
- daher: hier keine Details
- aber: es ist machbar



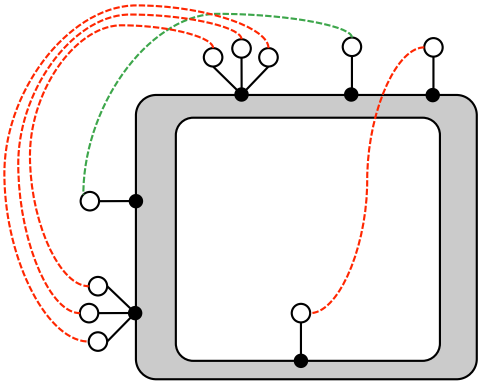
- bei Interesse: T.-S. Hsu: *Simpler and faster biconnectivity augmentation*, Journal of Algorithms 45 (2002), pp. 55-71

Zusammenfassung

- Biconnectivity Augmentation Problem optimal lösbar in O(|V| + |E|)
- Details zu Algorithmus mit Laufzeit O(|V| + |E| + |V|α(|V|))
- Ideen zu Algorithmus mit Laufzeit O(|V| + |E|)
- Erweiterungen:
  - Weighted Biconnectivity Augmentation
    - zusätzlich Kantengewichte
    - ⇒ NP-schwer
    - ⇒ Übung (?)
  - Planar Biconnectivity Augmentation (PBA)
    - *gegeben*: planarer Graph G=(V,E)
    - *gesucht*: min. Kantenmenge E' so dass G'=(V, E ∪ E') 2-zshgd.
    - *Bedingung*: G' ist planar
    - ⇒ NP-schwer

    hier: NP-Vollständigkeitsbeweis, 2-Approximation

PBA schwieriger!



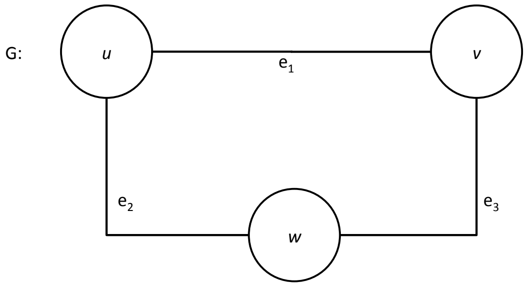
PBA ist NP-schwierig

- ursprünglich: Reduktion auf 3-Partition
  - 3-Partition ist stark NP-vollständig
  - graphentheoretisches Problem?
- hier: Reduktion auf planares Vertex Cover (PVC)
- VC: Knotenmenge, so dass alle Kanten abgedeckt werden
- planares VC: Eingabegraph ist planar
  - Garey, Johnson [1976]: PVC ist NP-schwer
  - Reduktion von PVC auf VC
- also: PVC durch PBA lösen:
  - PVC $\leq_p$ PBA (Entschidungsproblem zu PBA)

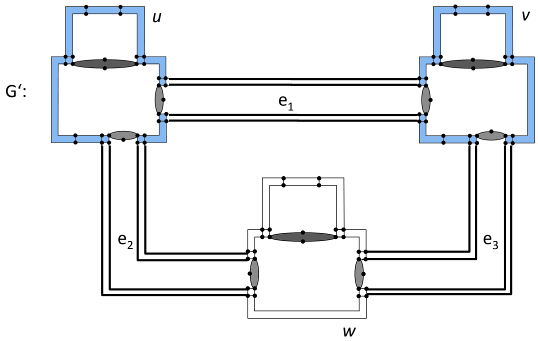
PVC $\leq_p$ PBA

PVC: Graph G=(V,E) gegeben

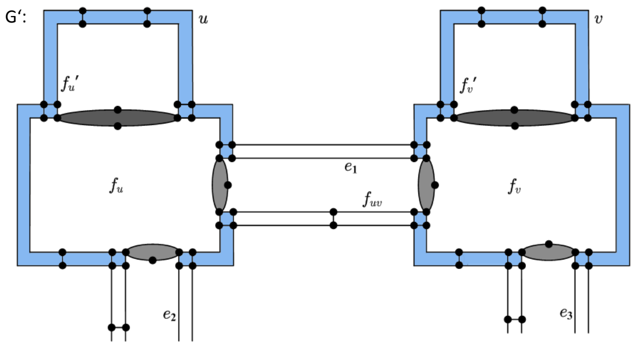
für Reduktion: fixiere Einbettung, konstruiere Graph G‘=(V‘,E‘)



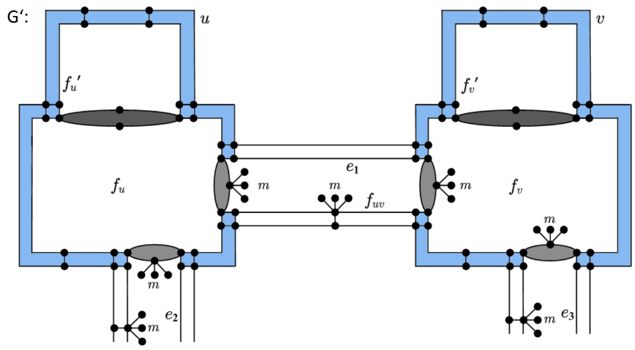
PVC $\leq_p$ PBA



PVC $\leq_p$ PBA



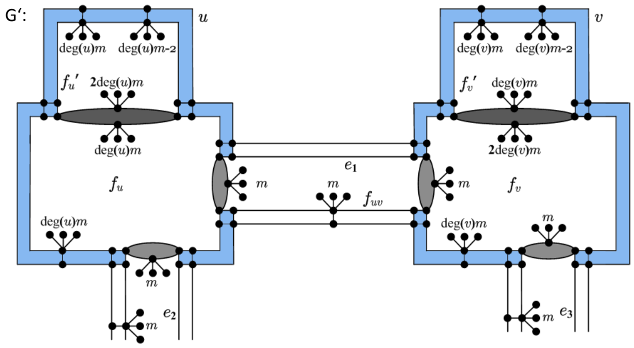
PVC $\leq_p$ PBA



- m := 4|V|

PVC $\leq_p$ PBA

G':



- m := 4|V|
- u ∈ VC, v ∉ VC

PVC $\leq_p$ PBA

Theorem es existiert VC in G mit k Knoten gdw. es existiert zulässige planare Augmentierung für G' mit 7,5m|E|-|V|+k Kanten

Beweis

„⇒“

- *Annahme*: k Knoten sind im VC
- BC(G') hat 15m|E| - 2|V| Blätter
  - m pro Kante
  - 7 deg(v)m -2 pro Knoten v
- fixiere Einbettung wie im Beispiel
- 2k Blätter können nicht billig verbunden werden
- 1/2 (15m|E| - 2|V| - 2k) + 2k Kanten zur Augmentierung = 7,5m|E|-|V|+k

$$= \sum_{v\in V}(7\deg(v)m-2)+\sum_{e\in E} m$$
$$= 7m\sum_{v}\deg(v)+m|E|-2|V|$$
$$= 14m|E|+m|E|-2|V|$$
$$= 15m|E|-2|V|$$

PVC $\leq_p$ PBA

Theorem es existiert VC in G mit k Knoten gdw. es existiert zulässige planare Augmentierung für G' mit 7,5m|E|-|V|+k Kanten

Beweis

„⇐“

- *Annahme*: Augmentierung für G' mit 7,5m|E|-|V|+k Kanten
  ⇒ 2k Blätter müssen teuer verbunden werden
- von den Blatt-„Bündeln“ der Größe m sind entweder alle Blätter billig oder alle Blätter teuer verbunden
- da m = 4|V| > 2k
  ⇒ diese Blätter liegen in den f_v' faces
- VC := Menge der Knoten mit 2 teuer verbundenen Blättern
  ⇒ zulässiges Vertex Cover der Größe k

PVC $\leq_p$ PBA

Theorem es existiert VC in G mit k Knoten gdw. es existiert zulässige planare Augmentierung für G' mit 7,5m|E|-|V|+k Kanten

Beweis

- G' ist planar
- G' hat Größe O(|V||E|)
- G' kann in polynomieller Zeit konstruiert werden □

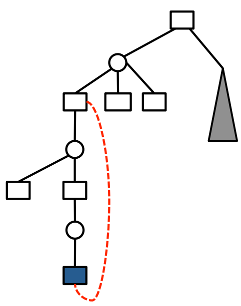
Ergebnisse zu PBA:

- PBA kann mit Güte 2 approximiert werden
- Die best-bekannte Approximationsgüte ist 2
- Laufzeit: O(|V| log |V|)
- ... und das mit einem sehr einfachen Algorithmus

2-Approximation zu PBA

Input: planarer, zshgd. Graph G=(V,E)
Output: planarer, 2-zshgd. Graph G

- solange BC-Baum mehr als 1 B-Knoten enthält:
  - falls C-Knoten existiert, wobei alle Kinder Blätter sind
    - verbinde diese B-Knoten zu einem B-Knoten
    - wähle dieses Blatt b
  - sonst wähle ein Blatt b zufällig
  - verbinde b mit dem am höchsten liegenden B-Knoten wie es die Planarität zulässt



2-Approximation zu PBA

Laufzeit:

- inkrementelle Planaritäts-Tests
- pro „Teilpfad“ in BC-Baum 1 Test
  - anschließend: Traversierung fortsetzen
  - oder: Kante einfügen ⇒ Pfad kontrahiert
- O(log |V|) amortisiert pro Planaritätstest
- Laufzeit: O(|V| log |V|)

Güte 2:

- jedes Blatt bekommt eine Kante: ℓ Kanten, optimale Lösung ≥ $\lceil \ell/2 \rceil$
- außerdem werden Kanten aufgrund von Planarität eingefügt. Diese Kanten müssen in einer optimalen Lösung aber auch vorhanden sein
- Details: siehe Kant, Bodlaender: *Planar Graph Augmentation Problems*, Algorithms and Data Structures, 2nd Workshop WADS '91, pp. 286-298

Ausblick/Organisatorisches

- Güte 2 das best-mögliche für PBA?
  - es gibt Paper mit 3/2- bzw. 5/3-Algorithmen; Gütebeweise fehlerhaft
  - Idee ⇒ Diplomarbeit
    - auch für Spezialfälle
- 5/3-Approximation für Spezialfall bei dem alle Schnittknoten zu einer 2-Zshgs.-komponente gehören und der SPQR-Baum Durchmesser 2 hat

Organisatorisches:

- nächste Vorlesung:
  - Mittwoch, 13.01.: Carsten Gutwenger:
    3-Zusammenhang, *SPQR-Bäume*, etc.
- nächste Übung:
  - Montag, 11.01.: *Isomorphie, TSP*