Automatisches Zeichnen von Graphen

Kap. 3: Hierarchische Zeichenverfahren

Prof. Dr. Petra Mutzel

Lehrstuhl für
Algorithm Engineering LS11
Universität Dortmund

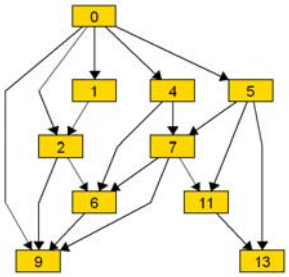
5. VO WS07/08 29. Oktober 2007

Literatur für diese VO

- P. Mutzel: Zeichnen gerichteter Graphen, Skript ursprünglich geschrieben für Schülerinnenveranstaltung
- Einführung in hierarchisches Zeichnen: Emden R. Gansner, Eleftherios Koutsofios, Stephen C. North, Kiem-Phong Vo: A Technique for Drawing Directed Graphs, *Software Engineering* 19 (3), 214-230, 1993
- Originalartikel: K. Sugiyama, S. Tagawa, M. Toda: Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11 (2), 109-125, 1981.

Kapitel 3: Hierarchische Verfahren

- Datenflußdiagramme
- Workflowdiagramme
- Activity Charts
- Organigramme
- UML Sequenzdiagramme
- UML Aktivitätsdiagramme
- Metabolische Pathways
- ...



Sugiyama, Tagawa, Toda 1981

Überblick zu Kapitel 3

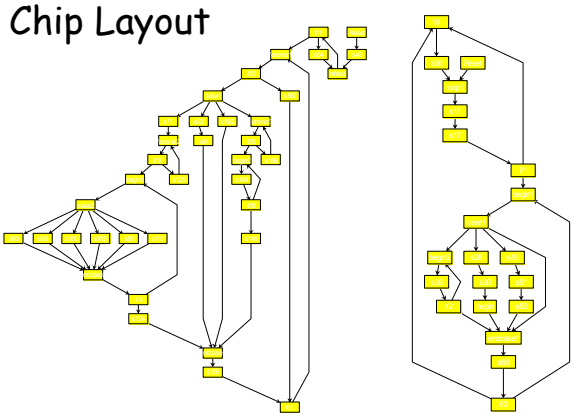
3.1 Einführung und Überblick

- Idee von Sugiyama, Tagawa und Toda 1981
- bis heute populärstes Zeichenverfahren
- wird i.a. für gerichtete Graphen verwendet
- am besten geeignet für gerichtete azyklische Graphen (DAG: „directed acyclic graph“)

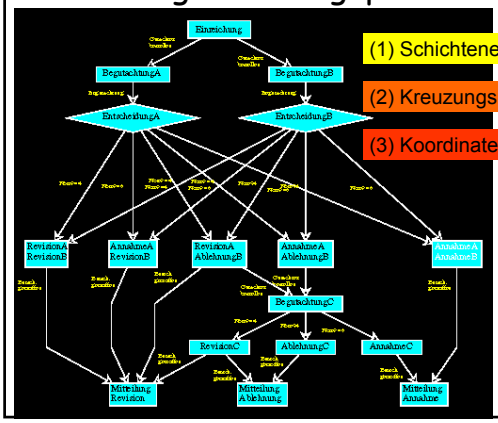
Voraussetzung ab jetzt: G ist ein DAG

- Falls G gerichtete Kreise enthält: Ändere Richtung von möglichst wenigen Kanten, so dass G azyklisch wird (NP-schwierig, mehr später)
- Falls G ungerichtet ist: Richte die Kanten so, dass keine gerichteten Kreise entstehen.

Chip Layout



Begutachtungsprozess



(1) Schichteneinteilung

(2) Kreuzungsminimierung

(3) Koordinatenzuteilung

Algorithmus von Sugiyama et al.

Drei Phasen:

1. Schichtzuweisung

- Jeder Knoten wird einer Schicht zugeordnet, so dass alle Kanten von oben nach unten zeigen

2. Kreuzungsreduktion / (- minimierung)

- Die Knoten werden nun innerhalb der Schichten so geordnet, dass es möglichst wenige Kreuzungen gibt.

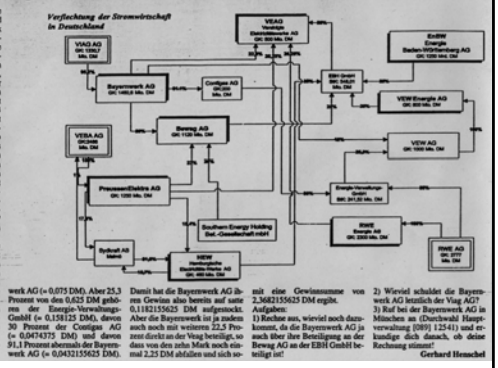
3. Knotenpositionierung

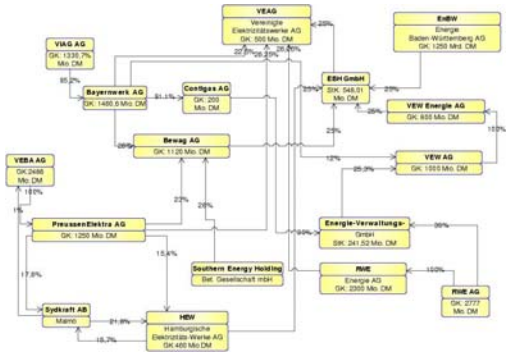
- Die Knoten und Kanten erhalten ihre genauen Koordinaten, so dass zick-zack Kanten verhindert werden

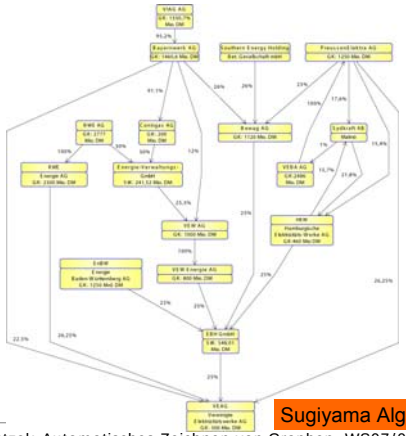
■ Mysterien und Leberwurstbrote oder: Die wirrsten Grafiken der Welt (8)

Die Verflechtung der Stromwirtschaft (nach Michael Stelte)

taz

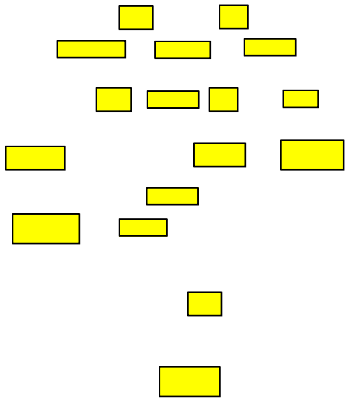




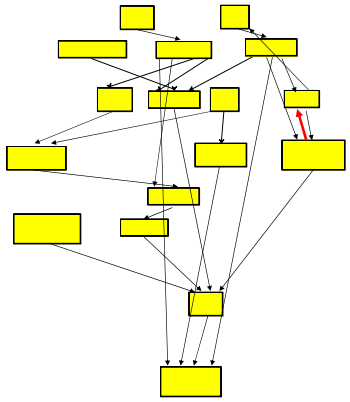


Sugiyama Algorithmus

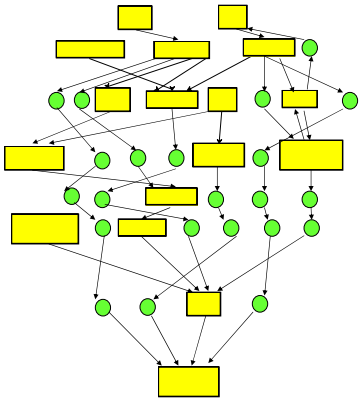
Festlegung der Schichten



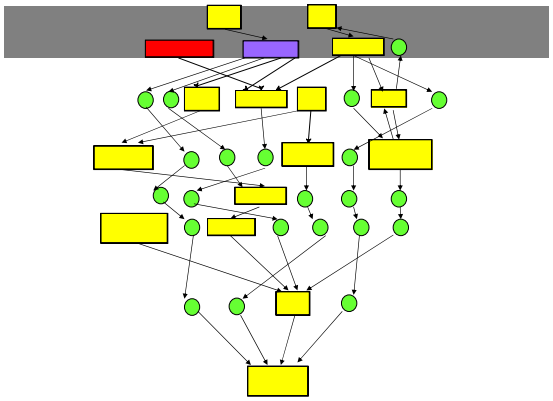
Festlegung der Schichten



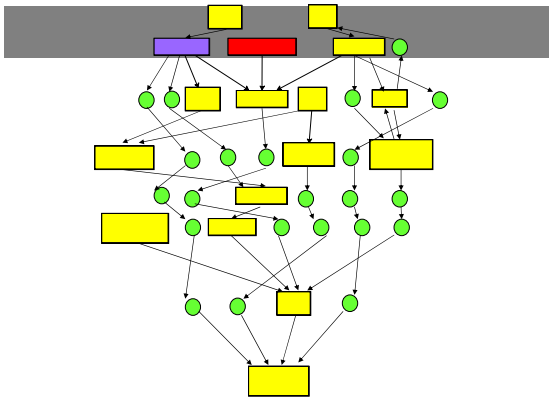
Einfügung künstlicher Knoten



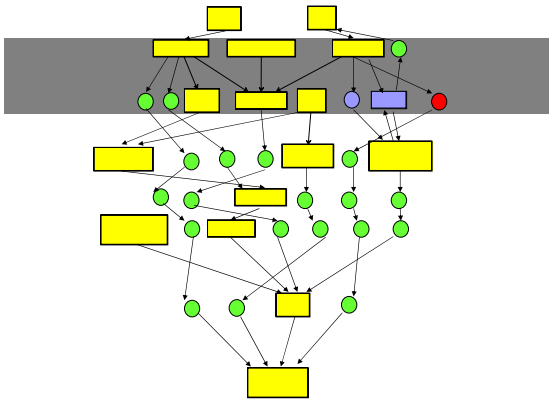
Kreuzungsreduktion



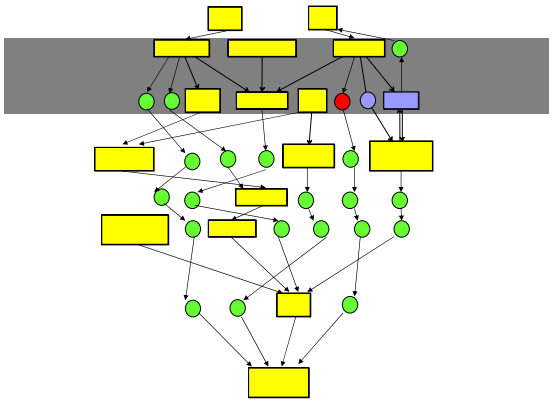
Kreuzungsreduktion



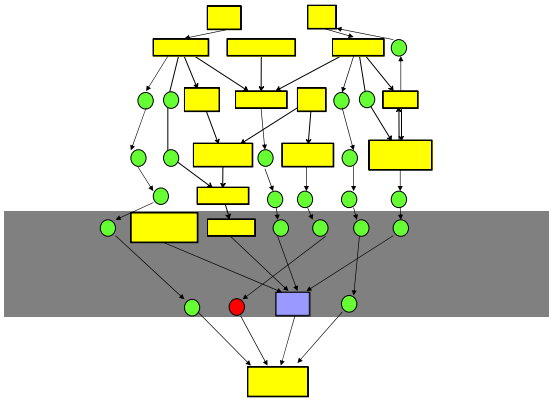
Kreuzungsreduktion



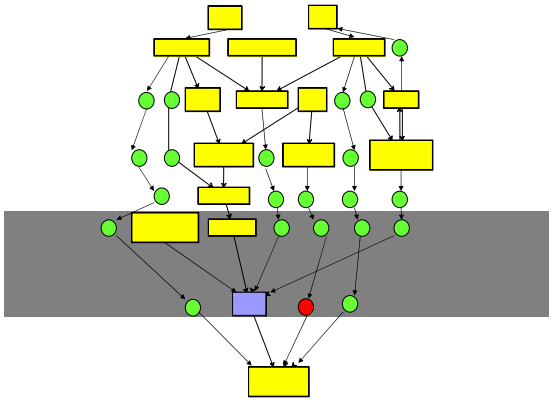
Kreuzungsreduktion



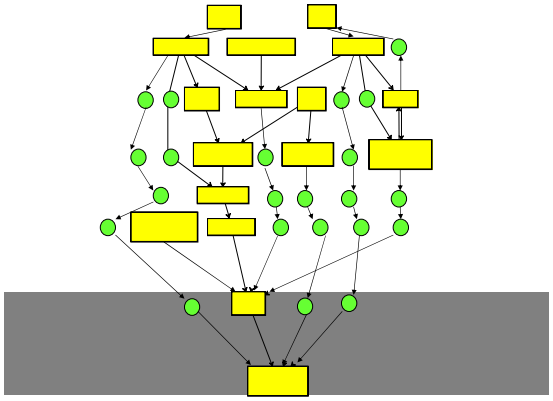
Kreuzungsreduktion



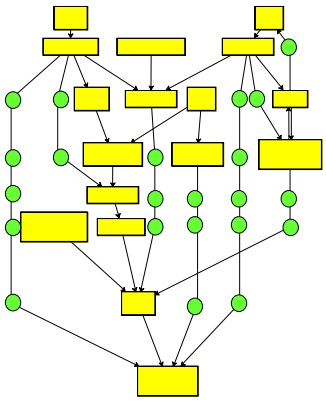
Kreuzungsreduktion



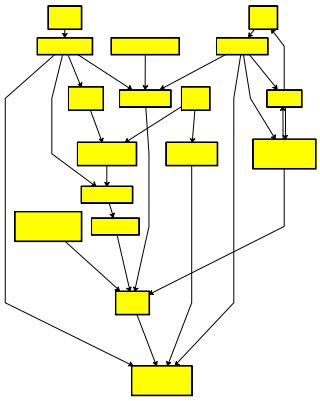
Kreuzungsreduktion



Begradigung der Kanten



Begradigung der Kanten

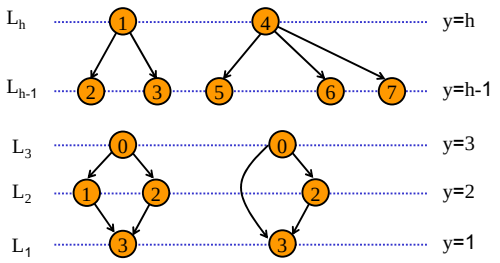


3.2 Schichtzuweisung

Definitionen:

- **Schichtzuweisung:** Partition $V = V_1 \dot\cup V_2 \dot\cup \ldots \dot\cup V_k$, so dass für alle $(u,v) \in E$ gilt: $u \in V_i, v \in V_j \Rightarrow i > j$
- h: Höhe des geschichteten Digraphen
- Breite des geschichteten Digraphen: $\max_{1 \le i \le k} |V_i|$
- einfache Schichtung: für alle $(u,v) \in E, u \in V_i, v \in V_j \Rightarrow i = j+1$

Zeichenkonvention



- Manchmal natürliche Schichtzuweisung, z.B. zeitliche Abläufe
- Sonst: Berechnung

Anforderungen

1. Kompaktheit: kleine Höhe und Breite
2. einfache Schichtung (wegen Kreuzungsminimierung) → erreichbar durch Einführung von künstlichen Knoten
3. wenige künstliche Knoten (bis zu ca. n^2 viele möglich), da:
   - positiver Einfluss auf spätere Phasen (Laufzeit abhängig von Knotenanzahl (echt und künstlich)
   - Knicke später nur an künstlichen Knoten
   - kurze Kanten i.a. besser

Layer Assignment

- Longest Path Layering (Längster Pfad Schichtung)
  – Adaption von Topologischer Sortierung
- Coffman/Graham [1972]
  – Multiprocessor Scheduling für garantierte Breite
- Optimales Layering bzgl. Kantenlänge
  – (Integer) Linear Programming
- Healy/Nikolov [2001]
  – Integer Linear Programming für garantierte Breite und Höhe inkl. künstlicher Knoten

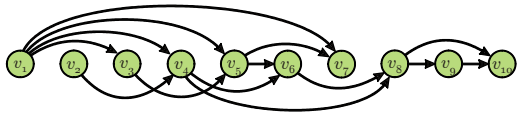
Longest Path Layering

- Idee: basiert auf Topological Sort

→ Rückblick auf DAP2

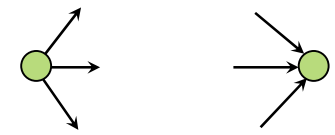
Topologisches Sortieren

Topologisches Sortieren	
Gegeben:	DAG $G = (V,A)$
Gesucht:	eine Sortierung $v_1,\ldots,v_n$ der Knoten von G mit $i<j$ für alle $(v_i,v_j)\in A$



Quellen & Senken

- Eine *Quelle* ist ein Knoten ohne eingehende Kanten
- Eine *Senke* ist ein Knoten ohne ausgehende Kanten



Beobachtung:
Jeder (nicht-leere) DAG G =(V,A) hat mind. eine Quelle und eine Senke.

Beweis: Annahme: G hat *keine* Senke
- Verfolge von u_1 an immer ausgehende Kanten
  $p_i := u_1,u_2,\ldots,u_i$
- Falls $i > |V| \Rightarrow$ ein Knoten zweimal auf p_i
- $\Rightarrow$ Kreis! Widerspruch!
- Analog für Quelle

Algorithmus

Idee:
Wähle immer Quelle und entferne sie dann

while G ist nicht leer **do**
  Wähle eine Quelle s in G und gib sie aus
  $G := G–s$
end while

Verbesserungen

Wie finden wir effizient eine Quelle?
- Wird v gelöscht, dann können nur Zielknoten w von Kanten (v,w) zu Quellen werden.
- Genügt ausgehende Nachbarmenge $N^-(v)$ zu betrachten.

Müssen wir Knoten wirklich löschen?
- Nein!
  Verwalte Eingangsgrad in Knotenfeld *indeg*.

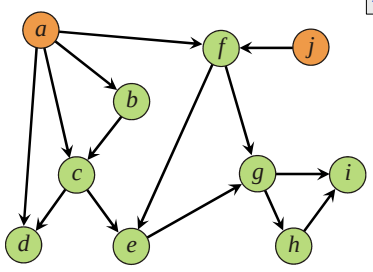
Algorithmus TopSort

```
(1)  var Queue Q
(2)  var int indeg[V]
(3)  ▷ Initialisierung
(4)  for all v ∈ V do {
(5)     indeg[v] := d+(v)
(6)     if indeg[v] = 0 then Q.PUT(v)
(7)  }
```


Algorithmus TopSort (2)

```
(8)  ▷ Hauptschleife
(9)  while not Q.ISEMPTY() do {
(10)    v := Q.GET()
(11)    Gib v aus
(12)    for all (v,u) ∈ A-(v) do {
(13)       indeg[u] := indeg[u] - 1
(14)       if indeg[u] = 0 then Q.PUT(u)
(15)    }
(16) }
```


Beispiel: TopSort



Ausgabe: a j b f c e d g h i

Q	a	j	b	f	c	e	d	g	h	i

Analyse der Laufzeit

- Initialisierung (Zeilen 1–7): $\Theta(|V|)$
  (da $d^+(v)$ in konstanter Zeit abrufbar)
- Jeder Knoten kommt genau einmal in Q
  $\Rightarrow$ **while**-Schleife wird $|V|$-mal durchlaufen
- Die **for all**-Schleife (Zeile 12) wird insgesamt für jede Kante einmal durchlaufen: $\Theta(|A|)$
- Gesamtaufwand: $\Theta(|V|+|A|)$

Analyse TopSort

Theorem:
Der Algorithmus TopSort berechnet eine topologische Sortierung der Knoten eines DAGs G=(V,A) in Zeit $\Theta(|V|+|A|)$.

ENDE Rückblick DAP2

Algorithmus TopSort

```
(1)  var Queue Q
(2)  var int indeg[V]
(3)  ▷ Initialisierung
(4)  for all v ∈ V do {
(5)     indeg[v] := d+(v)
(6)     if indeg[v] = 0 then Q.PUT(v)   #Q++
(7)  }
     count=1; new#Q=0
```


Algorithmus Longest Path Layering

```
(8)   ▷ Hauptschleife
(9)   while  #Q ≠ 0  do {
(10)    Für alle i=1,…,#Q {
(11)      v := Q.GET()
(12)    level(v)=count
(13)    for all (v,u) ∈ A-(v) do {
(14)       indeg[u] := indeg[u] - 1
(15)       if indeg[u] = 0 then Q.PUT(u); new#Q++
(16)    }
(17)  } #Q=new#Q; new#Q=0; count++
(18)  }
```


Longest Path: Eigenschaften

- sehr effizient: Laufzeit $O(|V|+|E|)$
- Höhe ist minimal (= Länge eines längsten gerichteten Pfades in G)
- keine Kontrolle über die Breite und gleichzeitig minimale Höhe

Schichtung mit Breitenminimierung

Minimierung der Breite (*width*) W für gegebene minimale Höhe H ist NP-schwierig.

Dies folgt direkt aus der NP-Schwierigkeit von

MULTIPROCESSOR SCHEDULING
Gegeben: W Prozessoren, Laufzeit H, n Aufgaben mit Laufzeit 1, DAG G=(V,E), $|V|$=n, mit $(u,v) \in E \Leftrightarrow u$ muss vor v erledigt werden.
Frage: Kann alles mit W Prozessoren in Zeit H erledigt werden?

Positiver Aspekt: Heuristiken für Multiprocessor Scheduling sind direkt übertragbar

Coffman/Graham Schichtung

Lexikographische Ordnung auf endlichen Teilmengen von N:
$S,T \subset \{1,2,...\}$

$S < T$ falls $S = \phi, T \neq \phi$

oder $S \neq \phi, T \neq \phi, \max(S) < \max(T)$

oder $S \neq \phi, T \neq \phi, \max(S) = \max(T)$,

$S \setminus \{\max(S)\} < T \setminus \{\max(T)\}$

Beispiel:

$\phi < \{1\}$

$\{1,2,3\} < \{1,4\}$

$\{2,4,7\} < \{3,4,7\}$

CG-Schichtung Algorithmus

- **Eingabe:** DAG G=(V,E), W∈N
- **Ausgabe:** Schichtung mit Breite ≤ W
- Vorverarbeitungsphase: Erzeuge die transitive Reduktion von G → Eingabe G

Bemerkungen:

- Transitive Reduktion: es gibt keine transitiven Kanten
- Eine gerichtete Kante (u,v) in G heißt **transitiv**, wenn ein (u,v)-Weg in G\(u,v) existiert.
- Transitive Reduktion kann in linearer Zeit (basierend auf DFS) ermittelt werden.
- Direkt mit DFS → quadratische Zeit bzgl. E

CG-Schichtung Algorithmus

Idee: 2 Phasen

Phase 1: Berechnung einer Ordnung π[] auf den Knoten:

1. Das Label 1 erhält ein Quellknoten
2. Seien Labels 1,2,…i-1 bereits vergeben, dann erhält derjenige Knoten v das Label i mit
   i. v besitzt noch kein Label
   ii. Alle Knoten u mit (u,v)∈E besitzen bereits ein Label
   iii. Unter all denjenigen v, die i. und ii. erfüllen, ist die Menge der direkten Vorgänger lexikographisch minimal.

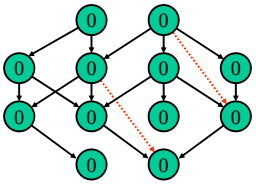
Phase 2: Starte mit dem untersten Layer und wähle den Knoten mit größtem Label, der platziert werden darf.

CG-Schichtung Algorithmus

1. **Für alle** i=1,…,|V|: π[i]=0
2. **Für alle** i=1,…,|V| {
3. Wähle v∈V mit π[v]== 0 und {π[u] | (u,v)∈E} minimum bzgl. „<„
4. π[v]=i }
5. K=1; L_1=∅; U=∅
6. **Solange** (U ≠V) **{**
7. **Falls** $|L_K|$<W und ein u∈V\U existiert mit
8. ∀ (u,w)∈E: w∈L_1 ∪ L_2 ∪…∪ L_{K-1} {
9. Wähle ein solches u mit π[u] maximum
10. L_K=L_K ∪ {u}; U=U ∪ {u} **}**
11. **Sonst** {
12. K++; L_K= ∅}
13. }

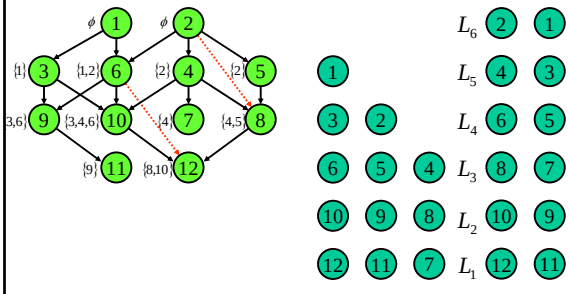
Coffman/Graham-Schichtung

Beispiel



Coffman/Graham-Schichtung

Beispiel



Coffman/Graham-Schichtung

Beide Lösungen des Beispiels sind optimal, aber i.a. gilt nur:

Theorem (Lam & Sethi [1977])
Sei h_{min} die minimale Höhe einer Schichtung mit Breite W. Dann gilt für die Höhe h der CG-Schichtung.

$$h \leq \left(2 - \frac{2}{W}\right) h_{\min}$$

Achtung: Das macht nur Sinn, wenn die Größe künstlicher Knoten sehr viel kleiner als die echter Knoten ist.

Healy & Nikolov [2001]: ILP zur Berechnung einer Schichtung mit vorgegebener Breite und Höhe, falls eine solche existiert (zählt echte Knoten und künstliche Knoten).

Minimierung der Anzahl künstlicher Knoten

- $y_v \in N$: vertikale Koordinaten von v
- **ILP-Formulierung:** Minimiere $\sum_{(u,v)\in E} y_u - y_v$

so dass $y_u - y_v \geq 1 \quad \forall (u,v) \in E$

$y_v \in \mathrm{N}$

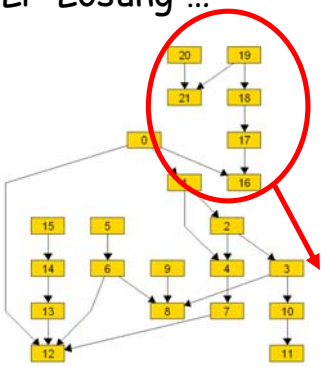
- Das LP ohne die Ganzzahligkeitsbedingungen hat eine ganzzahlige Lösung, denn die Matrix der Nebenbedingungen ist total unimodular, d.h. die Determinante jeder quadratischen Untermatrix ist 0,1, oder -1.

Minimierung der Anzahl künstlicher Knoten

- In diesem Fall findet der Simplex-Algorithmus eine ganzzahlige Optimallösung.
- Es gibt schnelle spezialisierte Techniken zur Lösung dieses Problems.
- Eine praktisch effiziente Implementierung mit Hilfe einer Netzwerk-Simplex-Technik wurde von Gansner, Koutsofios, North, Vo (1993) beschrieben.

Beispiel: LP-Lösung ...

- Das LP minimiert nicht die Höhe der Zeichnung
- LP kann aber so abgeändert werden
- geändertes Layout zeigt auch TopSort-Variante
- CG-Zeichnung: s. Übung



... und mit Coffman-Graham Layering (AGD)