Algorithmen im Chip-Entwurf 5

Reale Algorithmen zur Partitionierung, Timing-Analyse und Platzierung

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Übersicht

- Timing-Analyse
  - Mehrere kritische Pfade
- Platzierung
  - Annealing Mechanismus
  - Kostenfunktion
- Optional: Kernighan-Lin
  - Partitionierung via MinCut
- Zusammenfassung

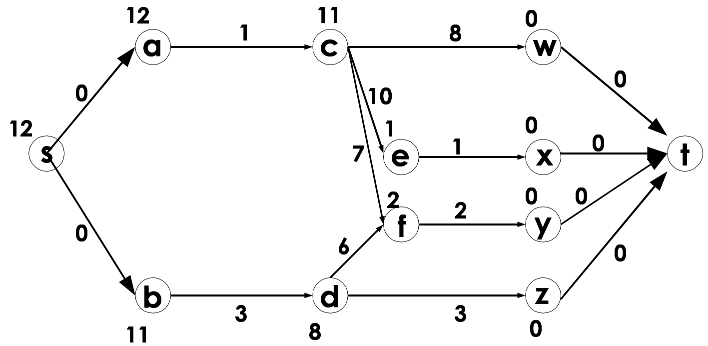
Timing-Analyse

- Kritischer Pfad
  - Einfach (slack=0)
  - Nächstkritischer Pfad?
- Vorgehensweisen
  - Alle Pfade berechnen
    - Rechenzeit- und Speicherbedarf
  - *k* längste Pfade en Block berechnen
    - Wenig flexibel: *k* bei Start der Berechnung fest
  - Pfade inkrementell berechnen
    - Flexibel: Rechen- und Speicheraufwand reduziert
- Idee
  - Timing-Graph annotieren
  - Pfade aufzählen (enumerate)

Verfahren nach Ju und Saleh

- Design Automation Conference 1991
  - Paper auf Web-Seite
  - Details in Abschnitt 3
- Graphannotation
  - Längste Verzögerung bestimmen
  - Aber auch an jeder Abzweigung merken
    - Wieviel schneller würde die Alternative sein?
- Pfadaufzählung
  - Beginne mit längstem Pfad
  - Wähle minimal schnellere Abzweigung
  - Erzeuge von dort ausgehend längsten Pfad
- Vorteil
  - Erzeugung beliebig vieler/weniger Pfade
    - Exakt an Anforderungen anpassbar

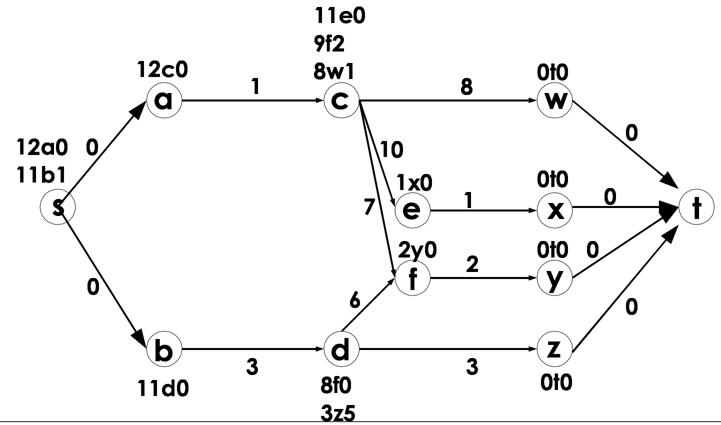
Annotation des Timing-Graphen



$$T_{sink}(u) = \underset{(u,v)\in E}{Max}\left(T_{sink}(v) + w(u,v)\right)$$

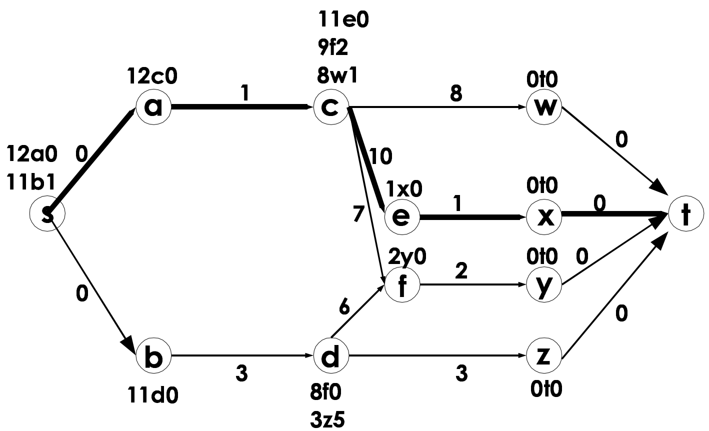
Erweiterung

Aber zusätzlich je *u*
- *T*sink + w((*u*,*v*)) von allen direkten Nachbarn *v* absteigend sortiert merken
- benachbarte Differenzen berechnen: *branch slacks*



Längster Pfad

- Beginn bei *s*
- Dann jeweils Kante mit maximaler T_{sink}
  - Bis *t* erreicht

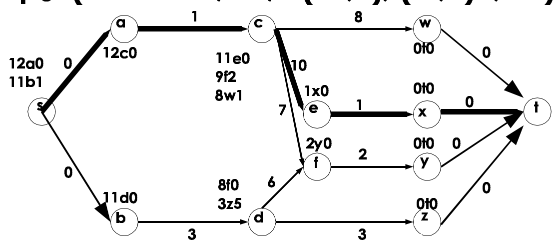


Datenstruktur

```
struct branch {                    ordered<branch, decreasing Tsink> succ(v)
        edge          e;
        unsigned int  slack;
}

struct path {
        list<vertex>                                      vertices;
        unsigned int                                      delay;
        ordered<branch,increasing branch.slack>           branches;
        unsigned int                                      nextdelay;
}
```

p0=(<sacext>, 12, <(sb,1), (cf,2)>, 11)



Vorgehen 1

- **longest_path(list<vertex> head)**
  - Verlängert *head* zu längstmöglichem Pfad
    - Wählt dazu jeweils Nachbar mit max. T*sink*
  - Merkt sich Nachbarn mit nächstkleinerer T*sink*
    - Also: Den mit kleinstem branch slack
  - Berechnet aktuelles und nächstkleineres Delay
- **branch_path(path p)**
  - Zweigt an Stelle *v* mit min. branch slack von *p* ab
  - Markiert Abzweigung in *p* als „genommen“
    - Berechne nächstkleineres Delay von *p* neu
  - Berechnet nun longest_path(p.vertices+<*v*>)

Vorgehen 2

- **Kernalgorithmus**
  - Annotiere Graph mit T_{sink} und branch slacks
  - Berechne längsten Pfad p_0=longest_path(<*s*>)
  - Merkt sich p_0 in *P*
  - Wiederholt, bis genug Pfade oder Delay=0:
    - Finde *p* aus *P* mit nächstkleinerem Delay = Max > 0
    - Generiere neuen Pfad *p'* = branch_path(*p*)
      - Verwende langsamste Abzweigung (min. branch slack)
    - Nimm *p'* in *P* auf
- ***P* enthält danach die gesuchten Pfade**

Beispiel

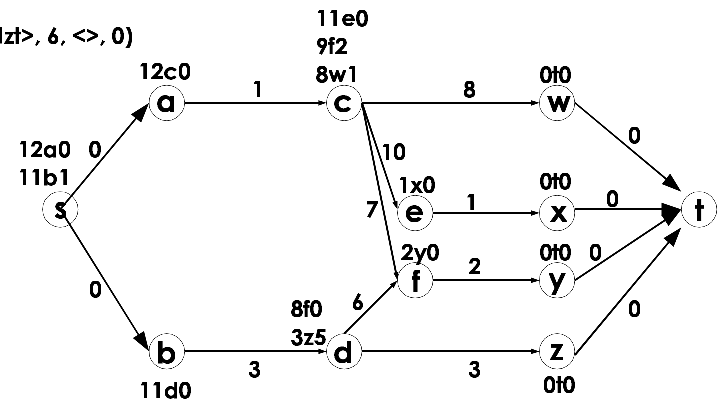
p_0 = (<sacext>, 12, <~~(sb,1)~~, ~~(cf,2)~~>, ~~11~~) ~~10~~ 0

p_1 = (<|sbdfyt>, 11, <~~(dz,5)~~>, ~~6~~) 0

p_2 = (<sa|cfyt>, 10, <~~(ew,1)~~>, ~~9~~) 0

p_3 = (<sa|cwt>, 9, <>, 0)

p_4 = (<sb|dzt>, 6, <>, 0)



VPR

- **Versatile Place and Route**
  - Betz und Marquardt, U Toronto
- **Platzierer**
  - Simulated Annealing-basiert
    - Adaptive Annealing Schedule
  - Optimiert gleichzeitig
    - Leitungslänge
    - Verzögerung

Züge

- **Paarweises Austauschen von Blöcken**
  - **N_{blocks} = Größe der Schaltung**
- **Aber nicht ganz wahllos**
  - **Beschränkung der Entfernung**

Starttemperatur

- **Wird automatisch bestimmt**
  - **Für aktuelle Schaltung passend**
- **Idee:**
  - **Anfangs fast alle Züge akzeptieren**
  - **Wie hoch muss die Starttemperatur sein?**
- **Vorgehen**
  - **N_{blocks} paarweise Austausche**
  - **Beobachte Änderung der Kostenfunktion x**
    - **Standardabweichung**

$$s_x = \sqrt{\frac{1}{n-1}} \left(\sum_i x_i^2 - n \bar{x}^2 \right)$$

  - **Starttemperatur = 20 x s_x**

Thermal Equilibrium

- **Anzahl von Schritten pro Temperaturstufe:**

$$10\, N_{blocks}^{4/3}$$

- **10x schneller, aber ca. 10% schlechter:**

$$N_{blocks}^{4/3}$$

Abkühlen 1

- **Beobachtung**
  - **Anfangs: T hoch, fast alle Züge akzeptiert**
    - **Im wesentlichen zufälliges Bewegen**
    - **Keine echte Verbesserung der Kostenfunktion**
  - **Ende: T niedrig, kaum Züge akzeptiert**
    - **Fast keine Bewegung mehr**
    - **Wenig Veränderung in Kostenfunktion**
- **Idee**
  - **Meiste Optimierung passiert dazwischen**
  - **Bringe T schnell in den produktiven Bereich**
  - **Halte T lange im produktiven Bereich**
- **Vorgehen**
  - **Steuere T anhand der Akzeptanzrate**

Abkühlen 2

- $T_{new} = \alpha \, T_{old}$

α	Acceptance Rate R_a
0.50	$R_a > 0.96$
0.90	$0.80 < R_a \leq 0.96$
0.95	$0.15 < R_a \leq 0.80$
0.80	$R_a \leq 0.15$

Abkühlen 3

- Vorahnung
  - Gute Fortschritte bei $R_a \approx 0,5$
- Am effizientesten $R_a = 0,44$
  - Beste Fortschritte
- Idee
  - R_a möglichst auf diesem Wert halten
  - *Nicht* temperaturbasiert (kühlt nur ab!)
  - Sondern: Auswirkungen der Züge beeinflussen
  - Beobachtung
    - Weite Züge: Grosse Änderung der Kostenfunktion
    - Kurze Züge: Kleine Änderung der Kostenfunktion
- Vorgehen
  - Variiere Zugweite D_{limit}, um $R_a \approx 0.44$ zu halten

Abkühlen 4

- D_{limit} klein
  - Kleine Zugreichweite
  - Kleine Änderungen der Kostenfunktion
  - Kleine Verschlechterungen
    - Werden eher angenommen
  - R_a steigt
- D_{limit} gross
  - Grosse Zugreichweite
  - Grosse Änderungen der Kostenfunktion
  - Große Verschlechterungen
    - Werden eher abgelehnt
  - R_a sinkt

Abkühlen 5

- Anfangs: D_{limit} = ganzer Chip L_{Chip}
- Bei jedem Abkühlschritt:

$$D_{limit}^{new} = D_{limit}^{old}\left(1 + R_a^{old} - 0.44\right),\ 1 \leq D_{limit}^{new} \leq L_{Chip}$$

  - Zuviel akzeptiert: D_{limit} grösser machen
  - Zuwenig akzeptiert: D_{limit} kleiner machen

Abbruchbedingung

- **Wann Abkühlung beenden?**
- **Idee**
  - Erkennung von Stillstand
- **Vorgehen**
  - Jeder Zug beeinflusst mindestens ein Netz
  - Bestimme die durchschnittlichen Kosten pro Netz
  - Wenn T kleiner als Bruchteil davon ...
    - Nur noch kleine Chance, dass Zug akzeptiert wird
    - T < 0.005 Cost/#Nets

Kostenfunktion

- **Gleichzeitig optimieren**
  - Zeitverhalten
  - Verdrahtungslänge
- **Verdrahtungslänge**
  - Bestimmt als korrigierter halber Netzumfang

$$c_w = \sum_{n \in N} q(n_{pincount})[bb_x(n) + bb_y(n)]$$

q(*i*) = 1 für *i*=1..3, =2.79 für i=50 (Cheng 1994)

- **Web-Seite: Paper, Datei mit Korrekturfaktoren q(*i*)**

Inkrementelle Berechnung 1

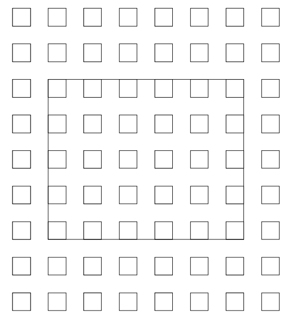
- **Berechnung des Netzumfangs**
  - Simpel: O(*k*), *k* Anzahl der Pins
  - Problem: k = 100 ... 1000 realistisch
  - Nach jedem Zug neu berechnen
- **Besser:**
  - Nach Möglichkeit nur bewegte Pins neu berechnen
    - Ein Pin ist nur in einem Netz
    - Ein Block hat aber mehrere Pins
- **Vorgehen**
  - Je Netz umspannendes Rechteck speichern
    - (x_{min}, x_{max}, y_{min}, y_{max})
      - Position der Seiten
    - (N_{xmin}, N_{xmax}, N_{ymin}, N_{ymax})
      - Anzahl Pins direkt auf den Seiten

Inkrementelle Berechnung 2

Betrachtet nur linke Seite (xmin)

- Bewege Terminal von x_{old} nach x_{new}
- Netz an Terminal: n

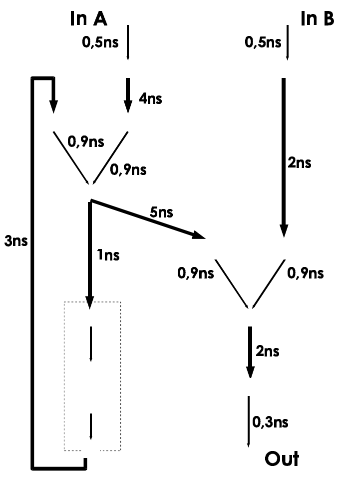
```
If (xnew != xold) {  // horiz. bewegt
    if (xnew < n.xmin) {
        n.xmin = xnew;
        n.Nxmin = 1;
    } else if (xnew == n.xmin) {
        n.Nxmin++;
    } else if (xold == n.xmin) {
        if (n.Nxmin > 1) {
            n.Nxmin--;
        } else {
            BruteForce(n);
        }
    }
}
```



xmin=2
xmax=7
ymin=3
ymax=7

Nxmin=1
Nxmax=2
Nymin=3
Nymax=1

Zeitverhalten 1



- **Betrachte**
  - Platzierungs-abhängiges Zeitverhalten
- **Punkt-zu-Punkt Verbind.**
- **Von**
  - Netzquelle *u*
- **Zu**
  - Jeder Netzsenke *v*
- **Sicht: *Two-Terminal-Nets***
  - $E_{NetTiming} \subset E_{Timing}$
- ***Zeitverhalten***
  - Bestimmt aus Slacks
  - Nicht auf Pfaden (langsam)

Zeitverhalten 2

- **„Wichtigkeit“ einer Verbindung**
  - Punkt-zu-Punkt zwischen Terminals *u* und *v*

$$\text{Criticality}(u,v) = 1 - \frac{\text{slack}(u,v)}{D_{max}}$$

  - (u,v) auf kritischem Pfad
    - slack(u,v) = 0 ⇔ Criticality(u,v) = 1
  - (u,v) absolut unkritisch
    - slack(u,v) = D_{max} ⇔ Criticality(u,v) = 0
- **Timing Cost: Delay(*u*,*v*) ist Schätzung!**
  - Noch kein „echtes“ Routing

$$c_t = \sum_{(u,v)\in E_{NetTiming}} \text{Delay}(u,v)\,\text{Criticality}(u,v)^{CriticalityExponent}$$

Zeitverhalten 3

- **Criticality Exponent**
  - Gewichtet kritischere Verbindungen höher
    - Wenige kritische Verbindungen dominieren c_t
  - Untergewichtet unkritischere Verbindungen
    - Fallen fast ganz aus c_t Berechnung heraus
- **Idee**
  - Gegen Ende auf kritische Netze konzentrieren
- **Vorgehen:**
  - Steigern von ce_{start}=1 auf ce_{final}=8 (experimentell)

$$\text{CritExp} = \left(1 - \frac{R_{limit}^{now} - 1}{R_{limit}^{start} - 1}\right) \cdot (ce_{final} - ce_{start}) + ce_{start}$$

Zeitverhalten 4

- **slack() ist platzierungsabhängig**
  - Unkritische Netz können kritisch werden
    - Zu lange Leitungslängen
  - Kritische Netze können unkritisch werden
    - Sehr kurze Leitungslängen
- **Slack-Werte müssen aktualisiert werden**
  - Timing-Analyse: T_a, T_r
- **Wie oft?**
  - Nach jedem Zug? Nach N Zügen?
  - N-mal pro Temperaturstufe?
  - Alle N Temperaturstufen?
- **Bewährt:**
  - 1x pro Temperaturstufe

Gesamtkostenfunktion

- **Selbstnormalisierend**

$$\Delta c = \lambda \frac{\Delta c_t}{c_t^{old}} + (1-\lambda) \frac{\Delta c_w}{c_w^{old}}$$

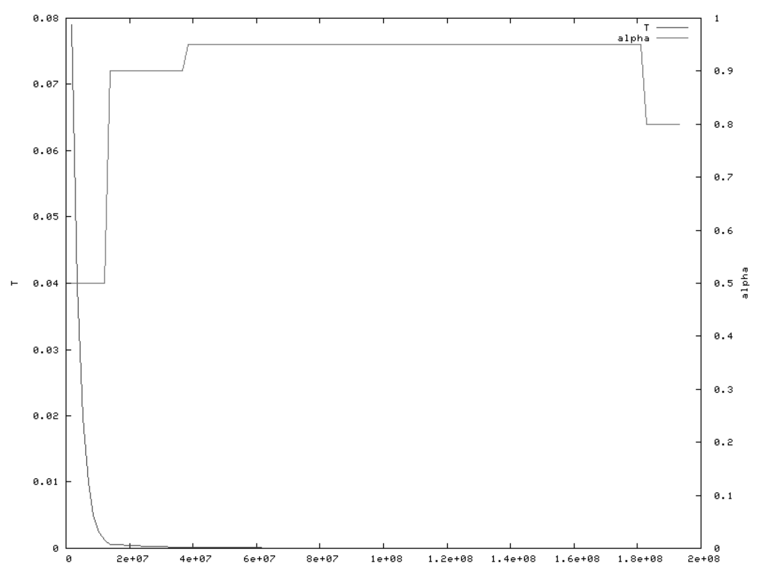
- **λ gewichtet Zeit ./. Längenoptimierung**
  - **Aber λ=1 erzeugt <u>nicht</u> die schnellste Lösung**
  - **Netze wechselnd kritisch/unkritisch**
    - **Nicht erkannt, da Timing-Analyse nur 1x pro Temp.**
  - **Besser λ=0.5**
    - **Längenmaß wirkt als Dämpfer für Oszillation**

Gesamtalgorithmus

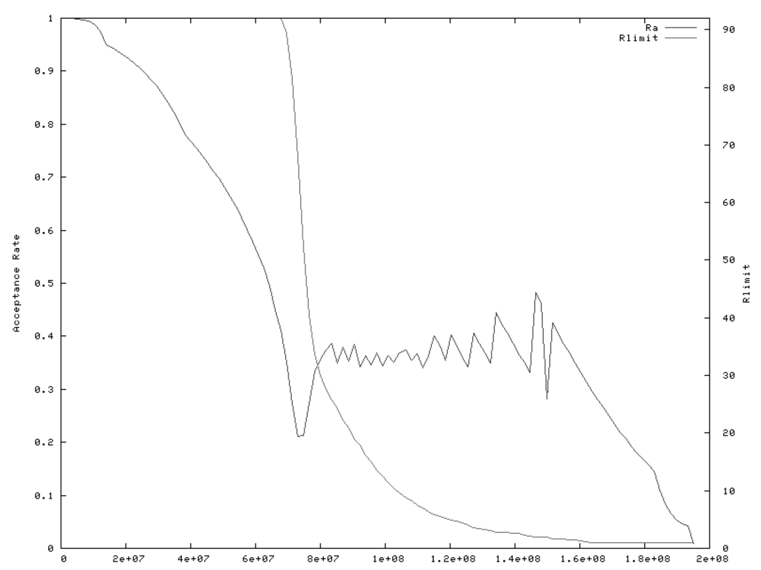
```
S = RandomPlacement();
T = InitialTemperature();
Rlimit = InitialRlimit();
CritExp = ComputeNewExponent(Rlimit);

while (!ExitCriterion()) {
    TimingAnalyze();                    // Bestimme Tα, Tr und slack()
    OldWiringCost = WiringCost(S);      // für Normalisierung der Kostenterme
    OldTimingCost = TimingCost(S);
    while (InnerLoopCriterion()) {      // eine Temperaturstufe
        Snew = GenerateSwap(S, Rlimit);
        ΔtimingCost = TimingCost(Snew) – TimingCost(S);
        ΔwiringCost = WiringCost(Snew) – WiringCost(S);
        ΔC = λ (ΔtimingCost/OldTimingCost) + (1-λ) (ΔwiringCost/OldWiringCost);
        if (ΔC < 0)
            S = Snew;
        else
            if (random(0,1) < exp(-ΔC/T))
                S = Snew
    }
    T = UpdateTemp();
    Rlimit = UpdateRlimit();
    CritExp = ComputeNewExponent(Rlimit);
}
```

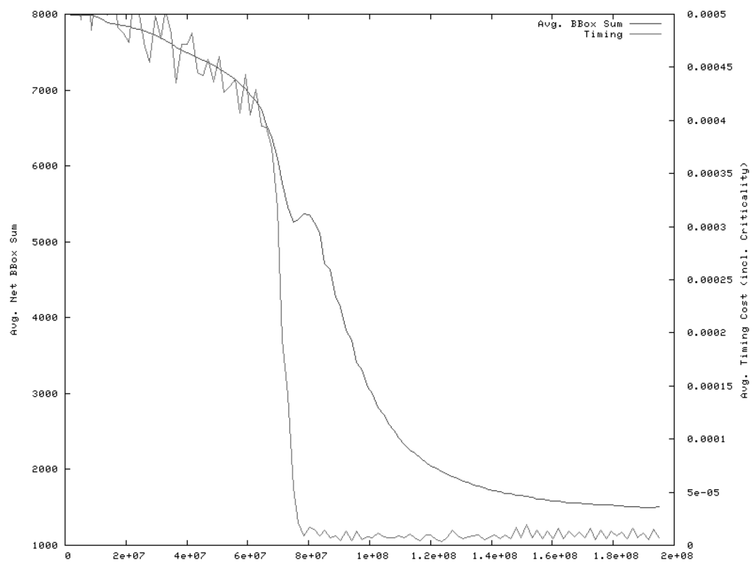

VPR Simulated Annealing 1



VPR Simulated Annealing 2



VPR Simulated Annealing 3



Partitionierung

- Aufteilen eines Graphen
- Hier motiviert durch Plazierung
  - Min-Cut
- Andere Anwendungen
  - Aufteilen einer Schaltung auf mehrere Chips
  - Verkleinern der Problemgröße
    - Vorbearbeitung vor anderem Algorithmus
- Viele Verfahren
  - Beispiel: Kernighan-Lin

Kernighan-Lin Partitionierung 1

- Problem
  - Gewichteter, ungerichteter Graph $G(V,E)$
  - $|V| = 2n$
  - γ_{ab}: Gewicht von $(a,b) \in E$, $\gamma_{ab}=0$ bei $(a,b) \notin E$
  - Finde Mengen A und B mit
    - $A \cup B = V$, $A \cap B = \emptyset$, $|A| = |B| = n$
  - Minimiere

$$\sum_{(a,b)\in A\times B} \gamma_{ab}$$

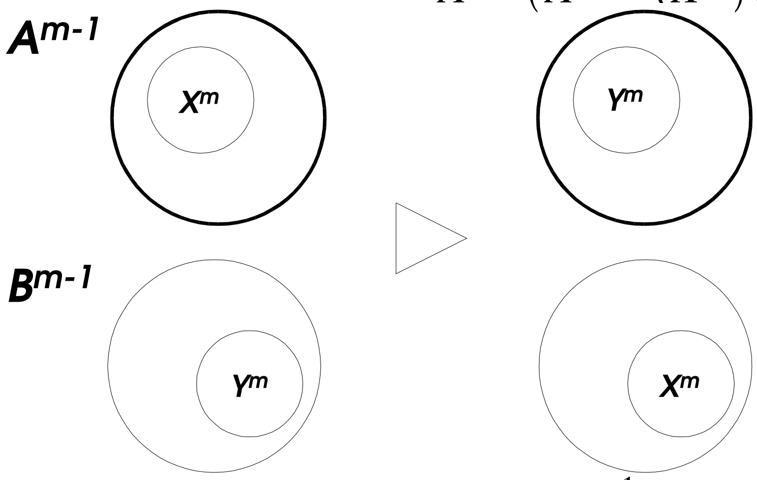
- Arbeitet auf Cliquen-Modell

Kernighan-Lin Partitionierung 2

- Partitionierungsproblem ist NP-vollständig
- KL ist eine Heuristik
  - Im praktischen Einsatz bewährt
- Vorgehensweise
  - Anfangslösung bestehend aus A^0 und B^0
    - I.d.R. nicht optimal
  - Isoliere Untermengen von A^{m-1} und B^{m-1}
  - Tausche diese aus um A^m und B^m zu bestimmen
  - Wiederhole, solange Verbesserung erreichbar

Kernighan-Lin Partitionierung 3

$$A^m = (A^{m-1} \setminus X^m) \cup Y^m$$



$$B^m = (B^{m-1} \setminus Y^m) \cup X^m$$

Kernighan-Lin Partitionierung 4

- Optimum immer in einem Schritt erzielbar
  - Bei geeignetem X^m und Y^m
- Problem: Wie X^m und Y^m bestimmen?
  - Schwer zu finden
- ➔ Suche Lösung in mehreren Schritten
  - Wiederhole, bis keine Verbesserung mehr
- Anzahl Schritte unabhängig von *n*
  - In der Praxis <= 4.

Kernighan-Lin Partitionierung 5

- Konstruktion von X^m und Y^m
- Externe Kosten

$$E_a = \sum_{y \in B^{m-1}} \gamma_{ay} \quad , a \in A^{m-1}$$

- Interne Kosten

$$I_a = \sum_{x \in A^{m-1}} \gamma_{ax} \quad , a \in A^{m-1}$$

- Analog für B

Kernighan-Lin Partitionierung 6

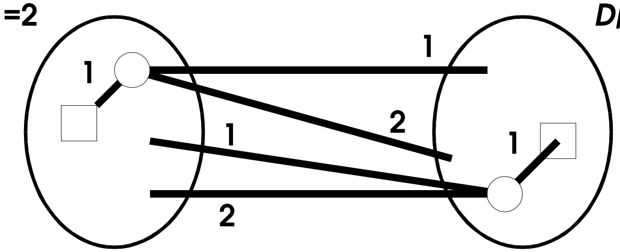
- $D_a = E_a - I_a$ für $a \in A^{m-1}$ (desirability)
  - >0: Knoten sollte nach *B* getauscht werden
  - <0: Knoten sollte in *A* bleiben
- Verbesserung Δ der Schnittkosten
  - Bei Austausch von $a \in A^{m-1}$ und $b \in B^{m-1}$

$$\Delta = D_a + D_b - 2\gamma_{ab}$$

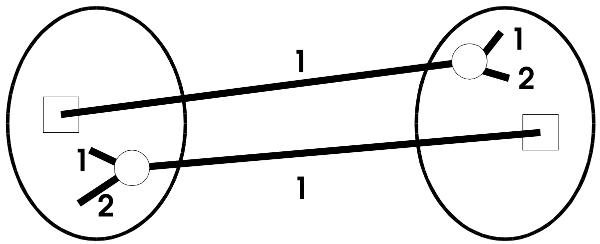
  - Δ kann negativ sein!

Kernighan-Lin Partitionierung 7

D_a=3-1=2 D_b=3-1=2

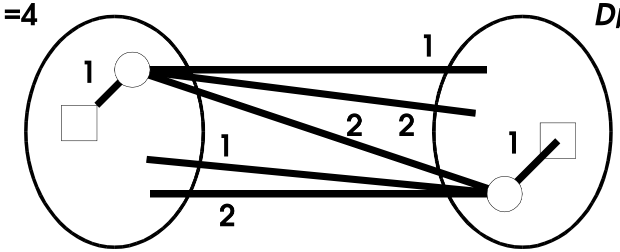


$\Delta = D_a + D_b - 2\,\gamma_{ab} = 2 + 2 - 0 = 4$

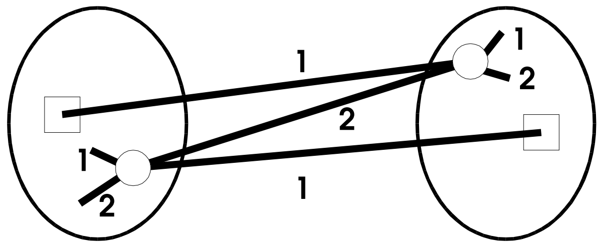


Kernighan-Lin Partitionierung 8

D_a=5-1=4 D_b=5-1=4



$\Delta = D_a + D_b - 2\,\gamma_{ab} = 4 + 4 - 2 \cdot 2 = 4$



Kernighan-Lin Partitionierung 9

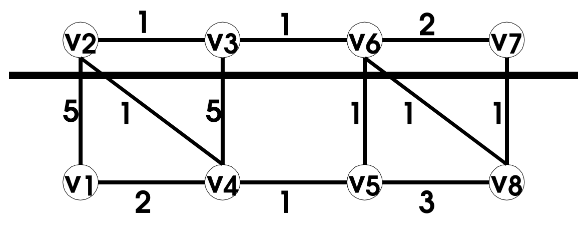
$$D_x = E_x - I_x$$
$$= E_x^{old} + \gamma_{ax} - \gamma_{bx} - (I_x^{old} - \gamma_{ax} + \gamma_{bx})$$
$$= D_x^{old} + 2\gamma_{ax} - 2\gamma_{bx}$$

```
initialize(A⁰, B⁰);
m := 1;
do {
  foreach a ∈ A^(m-1)
    "berechne Da";
  foreach b ∈ B^(m-1)
    "berechne Db"
  for (i:=1; i <= n; ++i) {
    "finde freie ai ∈ A^(m-1), bi ∈ B^(m-1) mit
      Δi := Dai + Dbi - 2 γaibi maximal"
    "sperre ai und bi"
    foreach "freies" x ∈ A^(m-1)
      Dx := Dx + 2 γxai - 2 γxbi;
    foreach "freies" y ∈ B^(m-1)
      Dy := Dy - 2 γyai + 2 γybi;
  }
  "finde ein k mit Σ_{i=1}^{k} Δi ist max."
  G := Σ_{i=1}^{k} Δi
  if (G > 0) {
    X^m := {a1, ... ak};
    Y^m := {b1, ..., bk};
    A^m := (A^(m-1) \ X^m) ∪ Y^m;
    B^m := (B^(m-1) \ Y^m) ∪ X^m;
    "entsperre alle Knoten in A^m and B^m"
    m := m + 1;
  }
} while (G > 0);
```


Kernighan-Lin Partitionierung 10

- **Δ_i kann negativ werden**
- **$\sum \Delta_i$ kann zeitweise auch negativ sein**
  - **Dicht verbundene Teilmengen**
    - **Keine Verbesserung bei Austausch von Einzelknoten**
    - **Erst bei Austauch der gesamten Teilmenge**

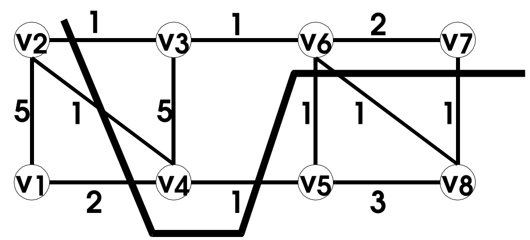
Kernighan-Lin Partitionierung 11



i	A^0 v2	v3	v6	v7	B^0 v1	v4	v5	v8	Δ_i
1	$\boxed{5}$	3	-1	-1	3	$\boxed{3}$	-3	-1	6
2		-5	-1	$\boxed{-1}$	-3		$\boxed{-1}$	-1	-2
3		-5	$\boxed{1}$		-3			$\boxed{3}$	2
4		$\boxed{-3}$			$\boxed{-3}$				-6

$\Sigma\Delta_i$: 6, 4, 6, 0

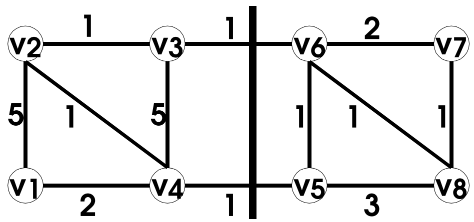
Kernighan-Lin Partitionierung 12



i	A^1 v3	v4	v6	v7	B^1 v1	v2	v5	v8	Δ_i
1	-5	$\boxed{-1}$	-1	-1	-3	-3	-1	$\boxed{-1}$	-2
2	$\boxed{5}$		-3	-3	-7	-5	$\boxed{3}$		8
3			$\boxed{-3}$	-3	$\boxed{-7}$	-7			-10
4				$\boxed{1}$		$\boxed{3}$			4

$\Sigma\Delta_i$: -2, 6, -4, 0

Kernighan-Lin Partitionierung 13



- Danach keine Verbesserung mehr in G
- Innere Schleife: *n* Iterationen
  - Finden des Paares mit bestem Δ: O(n^2)
  - Nach Δ sortiert: O(n log n)

→ O(n^3) oder O(n^2 log n)

Kernighan-Lin Partitionierung 14

- KL: Lokale Suche mit variabler Nachbarschaft
- Schnellere Verfahren
  - Fiduccia-Mattheyses (FM)
    - Wesentlich schneller: O(*n*)
    - Aber schlechtere Qualität der Lösungen
  - QuickCut (QC): avg. O(|E| log *n*)
    - Gleiche Qualität wie KL
- Diverse Alternativen
  - Spectral Partitioning, Multi-Level-FM, ...

Weiteres Vorgehen

- **Bewertung der Abgaben**
  - Bescheid über Platzzuteilung via E-Mail
  - Bis Mittwoch 18:00 Uhr
  - Zuweisung von Kolloquiums-Slot
- **Donnerstag**
  - Gruppenweise 30-minütige Kolloquien
  - Anwesenheitspflicht!
- **Freitag**
  - Gruppenweise 10-minütige Vorträge
  - Nicht überziehen!
  - Ausgabe der nächsten Aufgabe
- **VL Dienstag: 5.2-5.4 Exakte Optim.verfahren**

Zusammenfassung

- **Schnelle pfadorientierte Timing-Analyse**
- **VPR**
  - Adaptives Simulated Annealing
  - Selbstnormalisierende Kostenfunktion
  - Schnelle Netzumfangsberechnung
  - Gesamtalgorithmus
- **Kernighan-Lin MinCut-Partitionierung**
- **Papers auf Web-Seite**
  - Ju & Saleh 1991: Kritische Pfadaufzählung
    - Nur Abschnitt 3 relevant
  - Cheng 1994: q(i) Korrekturfaktoren
    - ... sonst eher schlecht zu lesen
  - Marquardt & Betz: VPR
    - 1997 Grundlagen
    - 2000 Timing-gesteuerte Betriebsart (Criticality, etc.)