# Algorithmen im Chip-Entwurf 9

## Kanalverdrahtung
## und
## globale Verdrahtung

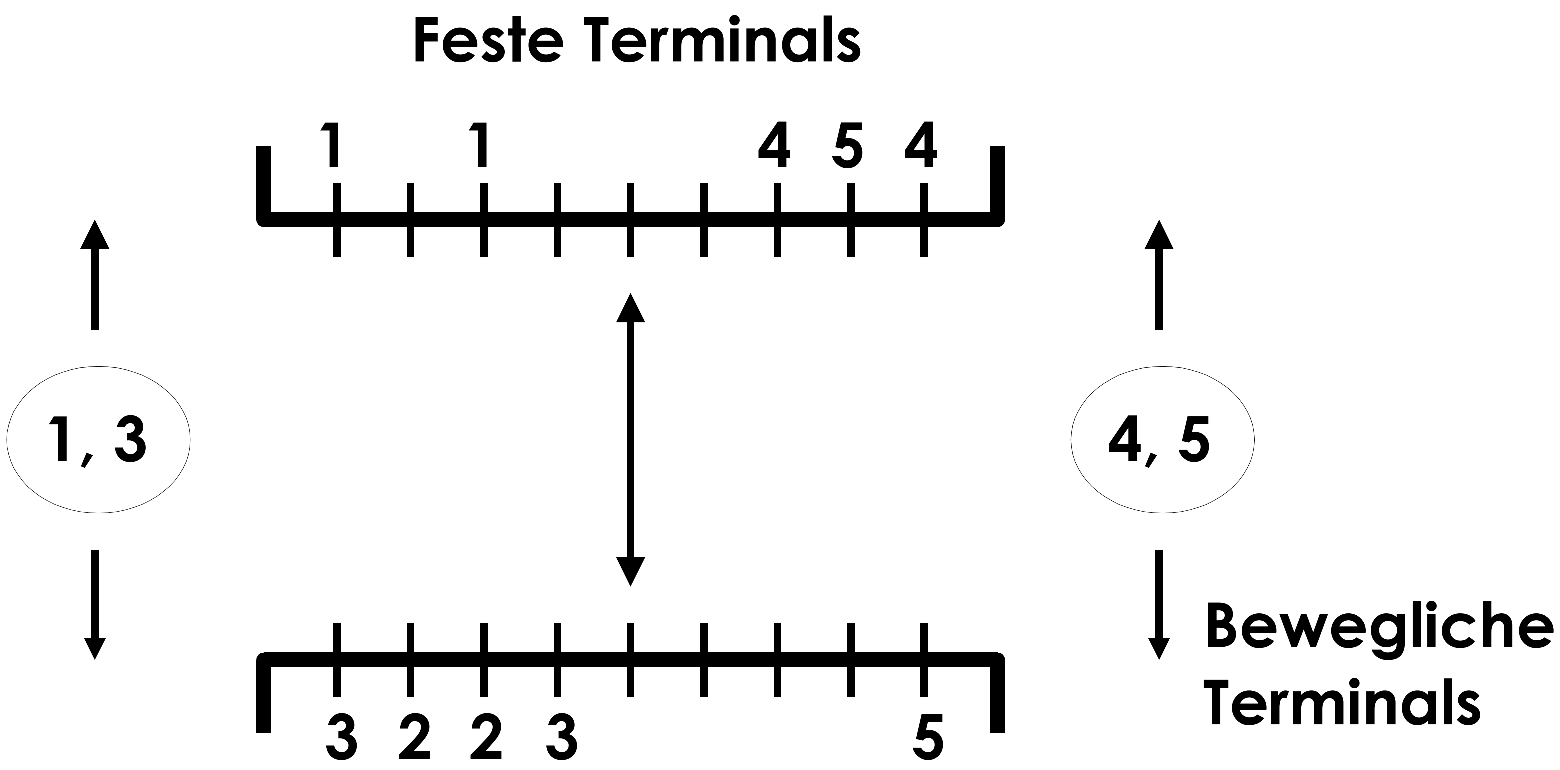
Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

# Überblick

- Wiederholung
  - H- und V-Einschränkungen
- Kanalverdrahtung
  - Yoeli's Robuster Router
  - Beispiel
- Globale Verdrahtung
- Konstruktion von Steiner-Bäumen
- Zusammenfassung

# Kanalverdrahtung 1

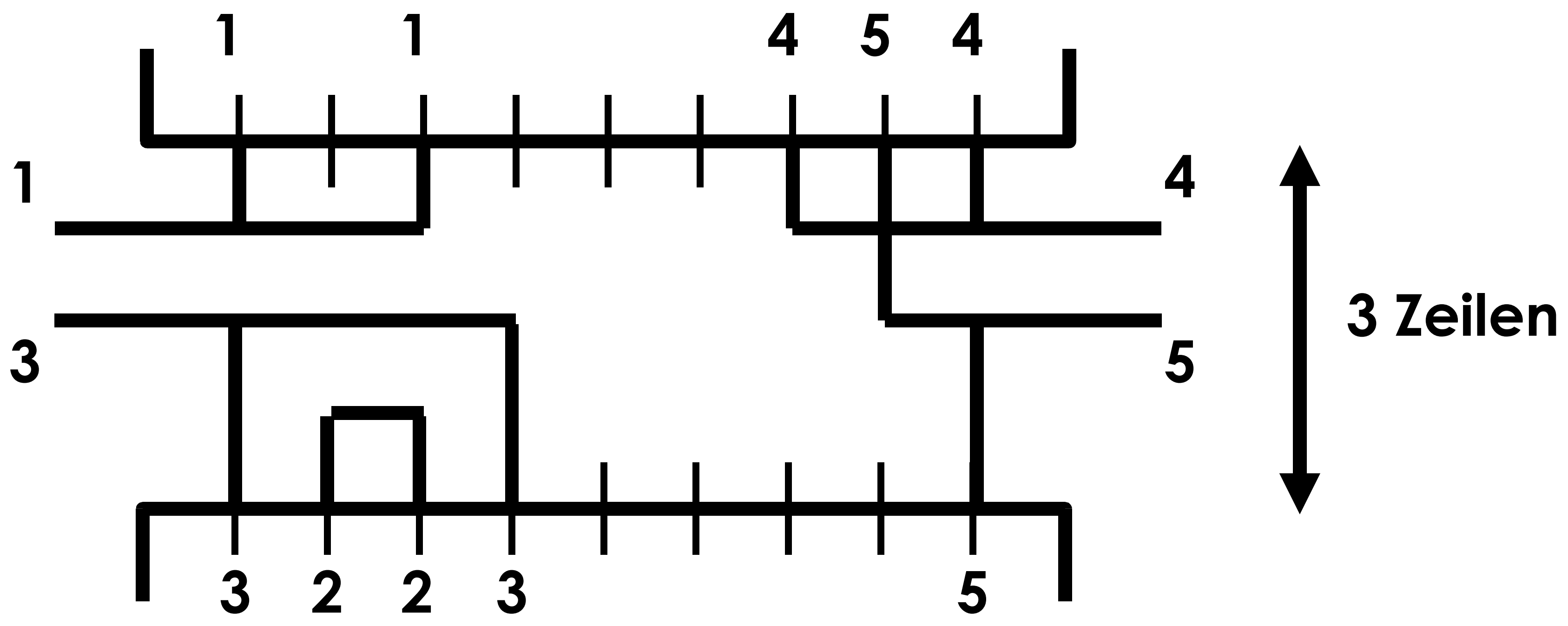
- Verdrahtung von Netzen in rechteckigem Kanal




- Ziel: min. Fläche, (min. Länge, min. Vias)

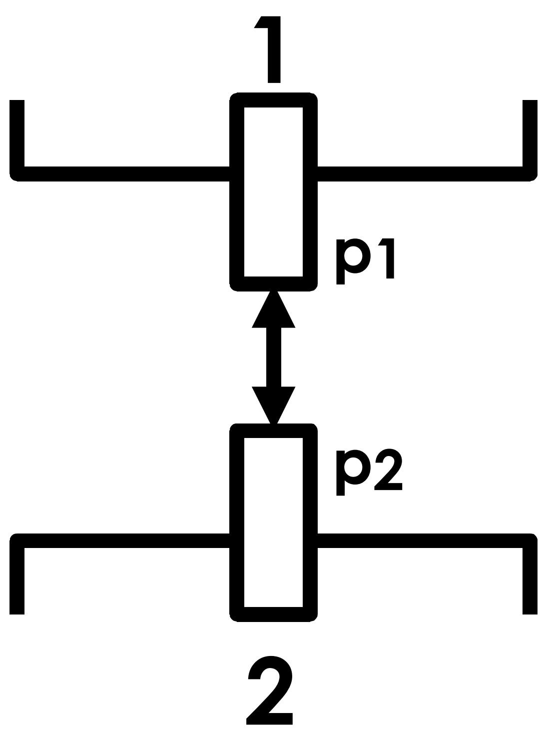
# Kanalverdrahtung 2

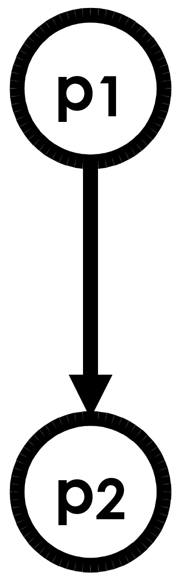
- Beispiel gelöst im klassischen Modell



# Vertikale Einschränkungen

- Zwei gegenüberliegende Terminals
  - Oberes Segment in den Kanal *muß* über unterem Segment in den Kanal liegen
    - Sonst Kurzschluß




Vertical Constraint Graph (VCG)

# Horizontale Einschränkungen

- Im klassischen Modell
  - Keine Überlappung zwischen H-Segmenten verschiedener Netze in gleicher Zeile
  - Sonst Kurzschluß

➔ Horizontale Einschränkung

- Falls keine vertikalen Einschränkungen
  - Keine gegenüberliegenden Terminals
  - ➢ Lösung durch Left-Edge Algorithmus (1971)

- Was tun bei H+V Einschränkungen?
  - NP-vollständig!

# Robuster Kanal-Router 1

- Heuristik (Yoeli 1991)
- Algorithmus
  - Iteriert über alle Zeilen im Kanal
  - Verkleinert Problem mit jeder Iteration
  - Wechselt zwischen oberster / unterster Zeile
    - Arbeitet sich zur Kanalmitte vor
  - Zwei Phasen
    - Berechnen von Gewichten für Netze
      - Wie gut wäre aktuelle Zeile für Netz?
    - Selektion von Untermenge mit maximalem Gewicht
      - Heuristik bei Verletzung vertikaler Einschränkungen

# Robuster Kanal-Router 2

- **Berechnung der Gewichte $w_i$ für Netz $i$**
  - ❶ **Falls $i$ Spalten der maximalen Dichte überspannt,**

    $w_i \mathrel{+}= B$ (B groß)

    - ◆ Hoffe auf Verringerung der max. Dichte, unabhängig von Seite (steepest descent)
  - ❷ **Falls $i$ ein Terminal auf der <u>aktuellen</u> Seite (oben / unten) auf Spalte x hat,**

    $w_i \mathrel{+}= d(x)$ (für alle Spalten x)

    - ◆ Bevorzuge Netze mit Terminals auf aktueller Seite
  - ❸ **Für alle Spalten x bei denen eine vertikale Einschränkung verletzt würde,**

    $w_i \mathrel{-}= K\,d(x)$ ($5 \leq K \leq 10$)

    - ◆ Bestrafe verletzte Einschränkungen

# Robuster Kanal-Router 3

- Regeln typisch für Heuristiken
- Robust
  - Unempfindlich gegen kleine Änderungen
- Nach Bestimmung der Gewichte
  - Finde Netz-Untermenge mit maximalem Gewicht, die in selbe Zeile passen
    - Ohne Verletzung horizontaler Einschränkungen
  - Verwendet Intervallgraph
    - Kante zwischen Knoten überlappender Intervalle

# Robuster Kanal-Router 4

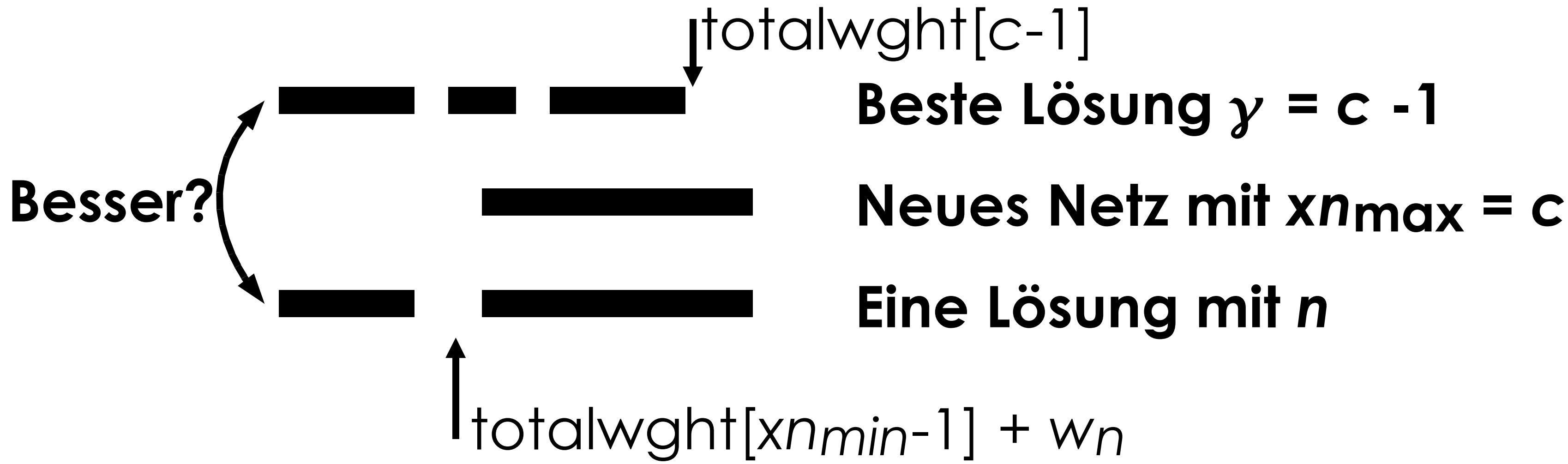
- Unabhängige Menge
  - Menge unverbundener Knoten
- Also gesucht:
  - Unabhängige Mengen maximalen Gewichts
    - Im allgemeinen NP-vollständig
    - Aber für Intervallgraphen in P !
- Vorgehensweise
  - Dynamic Programming
  - Konstruiere optimale Lösung aus Teillösungen
    - Komplexitätsparameter [illegible]: $1 \leq$ [illegible] $\leq$ Kanallänge

# Robuster Kanal-Router 5

- **$\gamma$ = Spalte c**
  - **Betrachte nur Netze mit rechtem Ende $\leq$ c**
- **Beispiel**
  - $i_1=[1,4]$, $i_2=[12,15]$, $i_3=[7,13]$, $i_4=[3,8]$, $i_5=[5,10]$, $i_6=[2,6]$, $i_7=[9, 14]$
  - $\gamma = 0$, $\gamma = 1$, $\gamma = 2$, $\gamma = 3$: $\emptyset$
  - $\gamma = 4$, $\gamma = 5$: {i1}
  - $\gamma = 6$, $\gamma = 7$: {i1, i6}
  - $\gamma = 8$: {i1, i6, i4}
  - ...

# Robuster Kanal-Router 6

- **Bestimme Lösung $\gamma$=c aus Lösung $\gamma$<c**
  - **Altes Maximalgewicht *plus***
    **Netz *n* mit rechtem Ende in Spalte c**
    - **Es ex. max. zwei solcher Netze (Terminals oben & unten)**
  - ***n* Teil der optimalen Lösung, falls**
    - **Gewicht von *n* plus Gewicht bestehender Netze <u>ohne Überlappung</u> mit *n* >= max. Gewicht ohne *n***



# Robuster Kanal-Router 7

- **Für Spalte c ausgewähltes Netz merken**
  - **In selected_net[c]**
  - **Kann leer sein (=0, kein neues dazugekommen)**
  - **Letztes (=rechtes) Netz immer in Lösung**
  - **Dann nach links suchen**
    - **Nach <u>nicht-überlappendem</u> Netz**
  - **Wiederhole bis linker Rand erreicht!**

- **Beispiel: ...,** $i_2=[5,9]$, $i_3=[4,6]$, ..., $i_7=[1,3]$, ...

| c= | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| s_n[c]= | 0 | 0 | 7 | 0 | 0 | 3 | 0 | 0 | 2 |

  - **$i_2$ in Lösung, überspringe $i_3$, $i_7$ in Lösung**

# Robuster Kanal-Router 8

- **Annahme: $d_{max}$ Durchgänge reichen**
  - **Wäre dann optimale Lösung**
- **Iteration**
  - **Gewichtsberechnung**
  - **Konstruiere**
    - **Maximal-gewichtige unabhängige Menge**
- **Aber:**
  - **Nur *Versuch* der Vermeidung von V-Konflikten**
    - **Keine Garantie!**

# Robuster Kanal-Router 9

- Falls V-Konflikt unvermeidbar
  - Entferne ein oder mehrere Netze
    - Welche?
    - Heuristik!
  - Verdrahte Netz(e) mit Maze-Routing
    - Gute Umgebung: Viele Hindernisse!
  - Vorgehensweise genannt: Rip-up and Reroute
- Auch hier: Keine Garantie auf Lösung

➔ Erneuter Durchlauf mit zusätzlicher Zeile
  - $d_{max}$ war nur untere Schranke für Zeilenzahl

- Ggf. auch zusätzliche Spalte

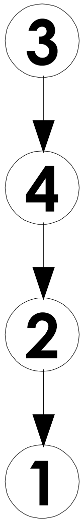
# Robuster Kanal-Router 10

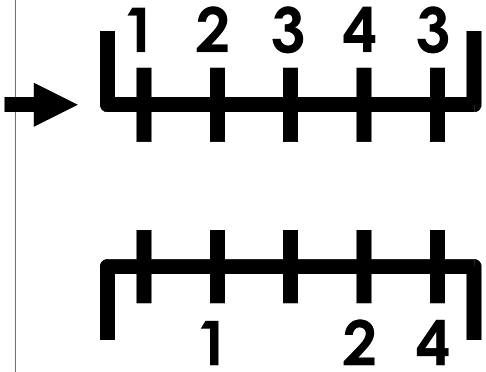
```
robust_router(placed_netlist N) {
  set<int> row;
  seq<set<int>> S;
  int[channel_width+1] totalwght, selected_net;
  bool top;
  int height, c, r, i;
  top := true;
  height := N.dmax();
  for (r := 1; r ≤ height; ++r) {
    forall "Netze i in netlist N"
      wi := i.compute_weight(N, top);
    totalwght[0] := 0;
    for (c:=1; c ≤ channel_width; ++c) {
      selected_net[c] := 0;
      totalwght[c] := totalwght[c-1];
      if (n = "Netz mit rechtem Term. oben in Spalte c") {
        if (wn + totalwght[xnmin-1] > totalwght[c]) {
          totalwght[c] := wn + totalwght[xnmin-1];
          selected_net[c] := n;
        }
      if (n = "Netz mit rechtem Term. unten in Spalte c") {
        if (wn + totalwght[xnmin-1] > totalwght[c]) {
          totalwght[c] := wn + totalwght[xnmin-1];
          selected_net[c] := n;
        }
      }
    }
    row := ∅;
    c := channel_width;
    while (c > 0)
      if (selected_net[c] != 0) {
        n := selected_net[c];
        row := row ∪ {n};
        c := xnmin - 1;
      } else
        --c;
    S.append(row);
    top := !top;
    N := "N ohne Netze in row";
  }
  "Maze-Routing bei V-Konflikten"
}
```

- Ggf. Wiederholung mit
  - Erhöhter Breite
  - Erhöhter Länge

# Robuster Kanal-Router 11

VCG




| | | | | | |
|---|---|---|---|---|---|
| x | 1 | 2 | 3 | 4 | 5 |
| d(x) | 1 | 2 | 2 | 3 | 2 |

***B* = 1000, *K* = 5**

$w_1 = (0) + (1) + (-5 \cdot 2) = -9$

$w_2 = (1000) + (2) + (-5 \cdot (2+3)) = 977$

$w_3 = (1000) + (2+2) + (-5 \cdot 0) = 1004$

$w_4 = (1000) + (3) + (-5 \cdot 2) = 993$

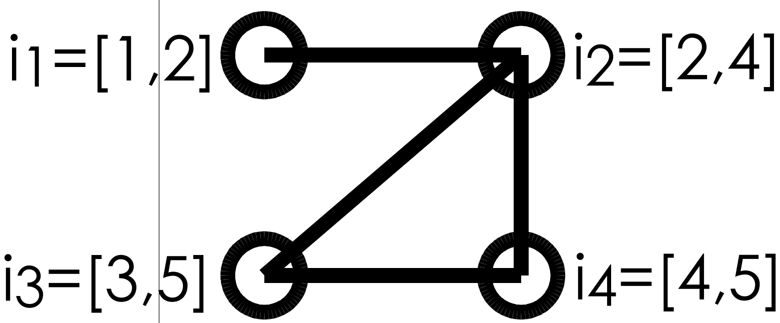
totalwght[1]=0 sel[1]=0

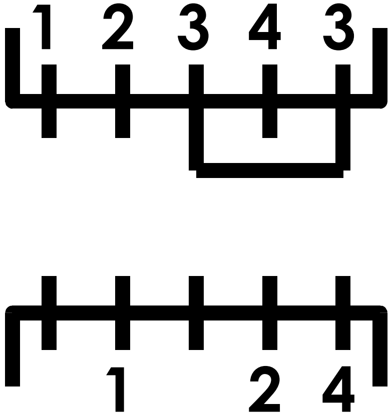
totalwght[2]=max(0,0-9)=0 sel[2]=0

totalwght[3]=0 sel[3]=0

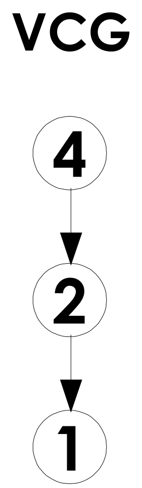
totalwght[4]=max(0,0+977)=977 sel[4]=2

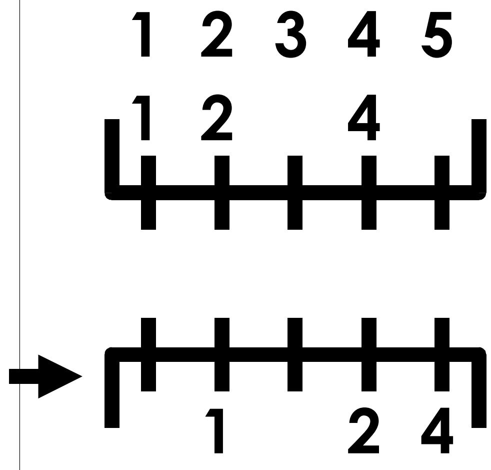
totalwght[5]=max(977,0+1004,0+993)=1004 sel[5]=3



# Robuster Kanal-Router 12




| | | | | | |
|---|---|---|---|---|---|
| x | 1 | 2 | 3 | 4 | 5 |
| d(x) | 1 | 2 | 1 | 2 | 1 |

$w_1 = (1000) + (2) + (-5 \cdot 0) = 1002$

$w_2 = (1000) + (2) + (-5 \cdot 2) = 992$

$w_4 = (1000) + (1) + (-5 \cdot 2) = 991$

totalwght[0]=0 sel[0]=0

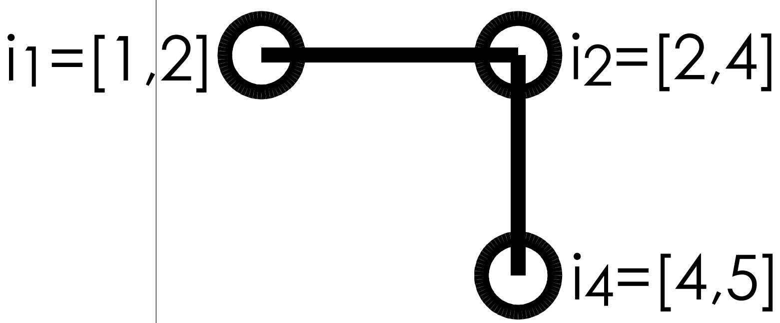
totalwght[1]=0 sel[1]=0

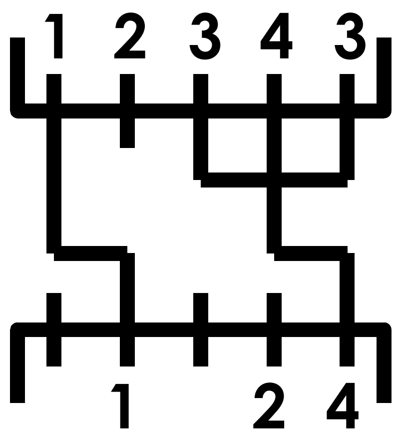
totalwght[2]=max(0,0+1002)=1002 sel[2]=1

totalwght[3]=1002 sel[3]=0

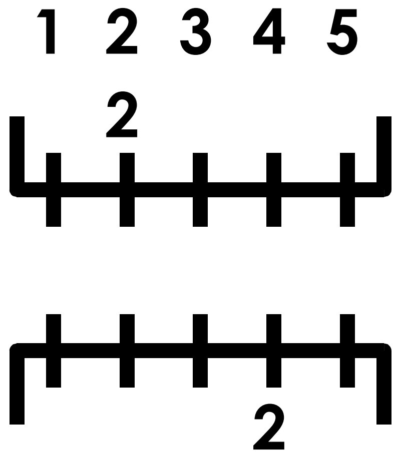
totalwght[4]=max(1002,0+992)=1002 sel[4]=0

totalwght[5]=max(1002,1002+991)=1993 sel[5]=4

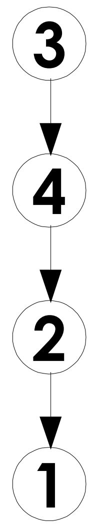


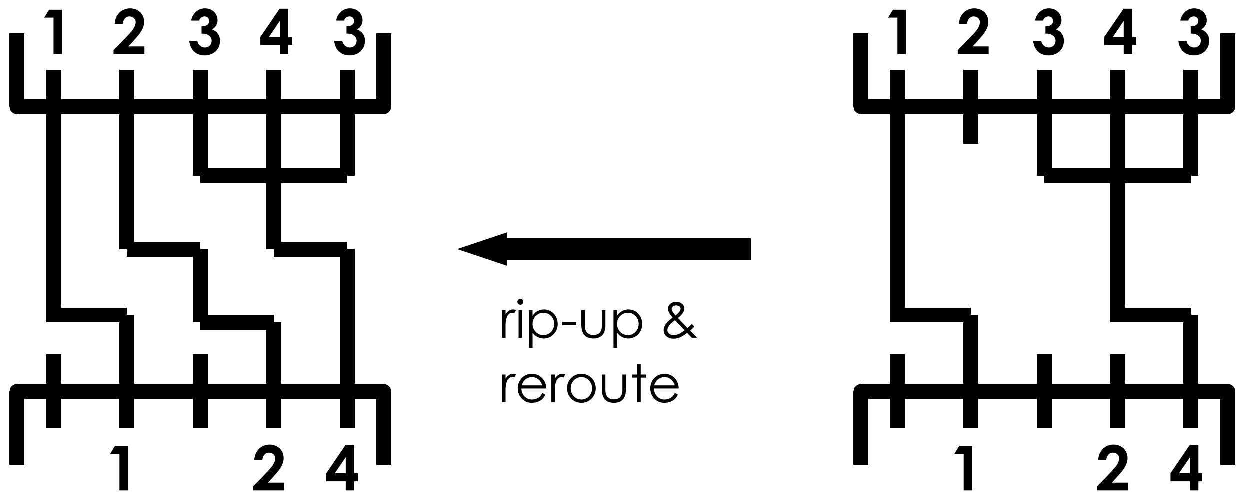



# Robuster Kanal-Router 13




- Trivial: Netz 2 in Zeile 2
- Kombinierte Lösung
  - Anordnung mit V-Konflikt
    - 1. Zeile: Netz 3
    - 2. Zeile: Netz 2
    - 3. Zeile: Netz 1, Netz 4



# Überblick Globalverdrahtung

- Wo kommen die Terminalpositionen her?
- Globalverdrahtung
  - Problem
  - Modellierung
  - Vorgehensweisen
- Algorithmus
  - Für Standardzellen
  - Steiner-Bäume
    - Konstruktionsheuristik
    - Optimierung

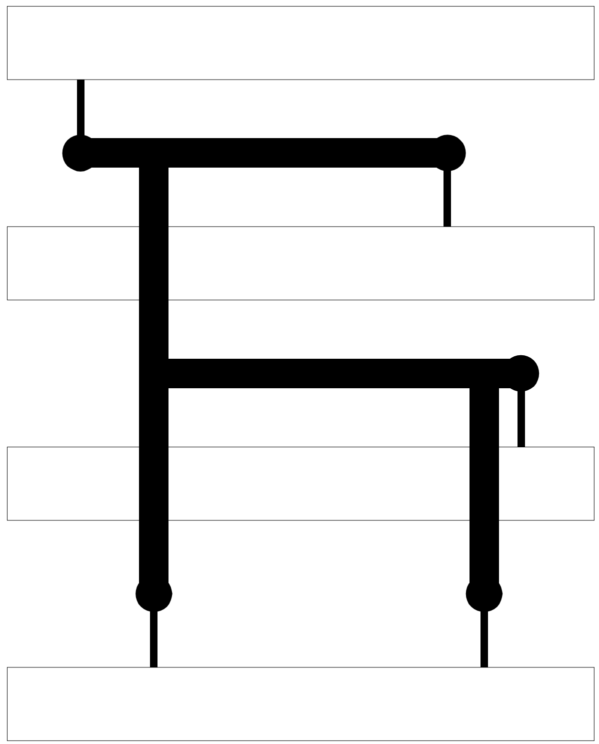
# Globale Verdrahtung 1

- Im Entwurfsfluß
  - Nach Platzierung
  - Vor lokaler Verdrahtung
- Verteilt Signale auf Kanäle
  - Führung innerhalb der Kanäle bleibt offen
- Optimiert auf
  - Minimale Fläche
  - Einhalten der Zeitvorgaben
- Hängt von Zieltechnologie ab

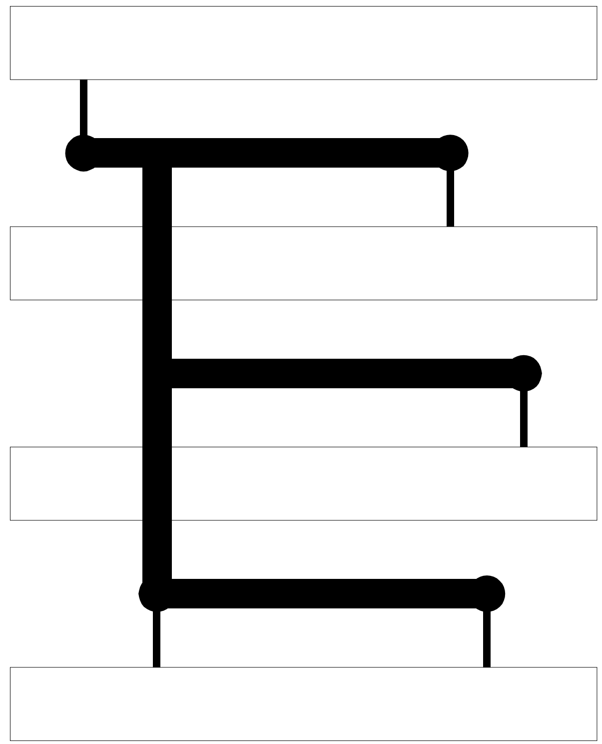
# Globale Verdrahtung 2

- Hier: Im Standardzellen-Entwurf
- Alle Terminals eines Netzes an einem Kanal?
  - Falls ja: Nur lokale Verdrahtung erforderlich
- Sonst: Globale Verdrahtung
  - Trennt Netz auf einzelne Kanäle auf
  - Übergang zwischen Kanälen
    - Reservierte Verdrahtungsebenen
    - Feedthroughs einfügen (beeinflußt Platzierung)
    - Vorgegebene Feedthrough-Leitungen allozieren
  - Idee: Rechtwinkliger Minimaler Steiner Baum (RSMT)
    - Ggf. höhere Kosten für vertikale Segmente (feedthroughs)
    - Wenn begrenzte Ressource

# Globale Verdrahtung 3



Rechtwinkliger Steiner-Baum
mit minimaler Länge



Rechtwinkliger Steiner-Baum
mit minimalen Übergängen

# Globale Verdrahtung 4

- RSMT nicht immer beste Lösung
  - Neben Länge zu berücksichtigen:
    - Begrenzte Anzahl von Feedthroughs
    - Zeitvorgaben (timing-driven)
      - Kritische Netze kurz halten
- Hier nur durch Gewichtung der Kosten möglich
    - Kann sehr ungenau werden

# Globale Verdrahtung 5

- Bessere Verzögerungsmodelle
  - Nur Verdrahtungslänge ungenau
    - Hier Widerstand und Kapazität zusammengeworfen
  - Besser:
    - R, C getrennt für einzelne Segmente
    - Bewährt: Elmore-Modell
      - Auch in VPR verwendet
- Dann andere Routing-Verfahren verwenden
  - Multicommodity Flow
  - Pattern-based
  - Hierarchical

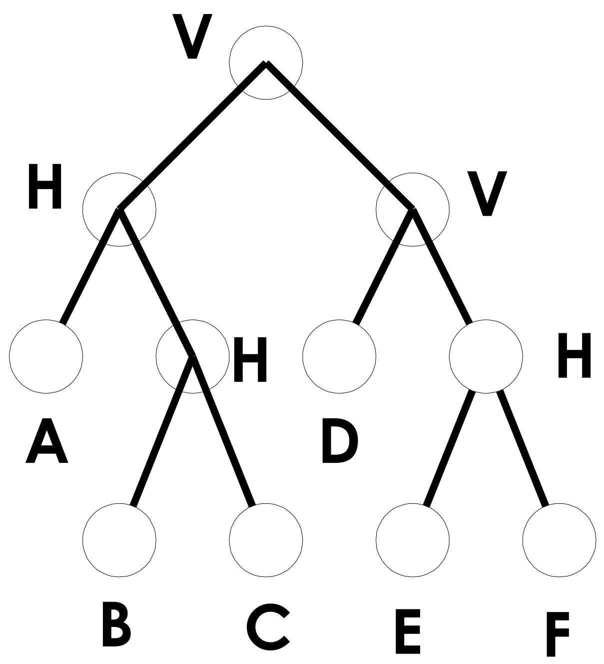
# Globale Verdrahtung 6

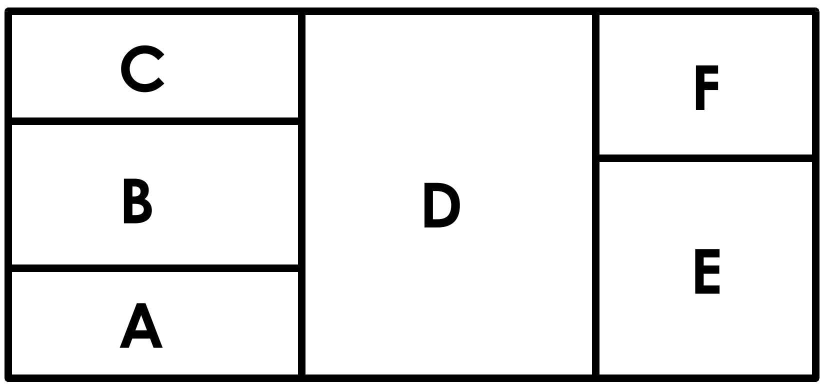
- **Annahme hier: Unidirektionale Sicht**
  - 1 Quelle /  n Senken
- **Mögliche Teiloptimierungsziele**
  - Kurzer Weg zu kritischer Senke
  - Gleich lange Wege (kleiner skew)
    - Verdrahtung von Takt-Leitungen (H-Trees)
- **Gesamtziel**
  - Minimiere Verdrahtungsfläche
  - Schätze Kanalbreiten ab

# Globale Verdrahtung 7

- Nun: Building-Block Layout
- Komplizierter!
- Irreguläre Freiflächen zwischen Zellen
  - Was sind überhaupt die Kanäle?
- Wie Flächen in Kanäle aufteilen?
  - Channel Definition Problem (CDP)
- Kanäle in welcher Reihenfolge verdrahten?
  - Channel Ordering Problem (COP)

# Exkurs Slicing Floorplans



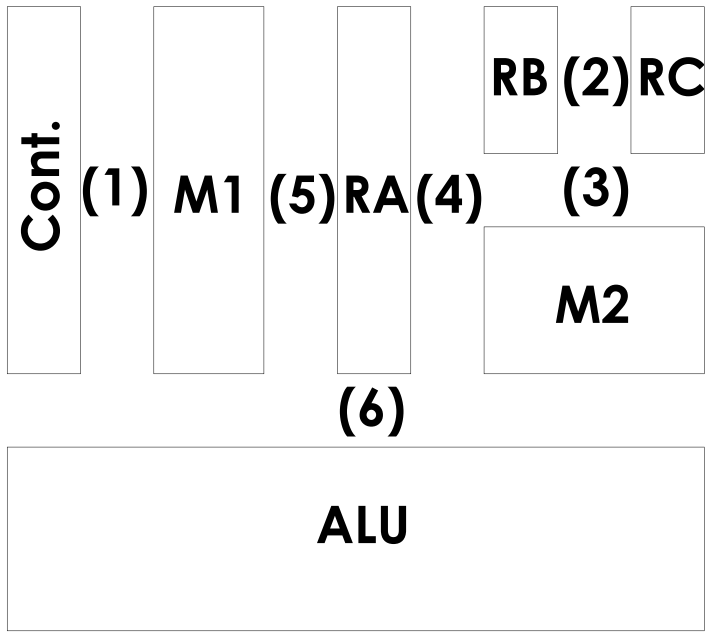



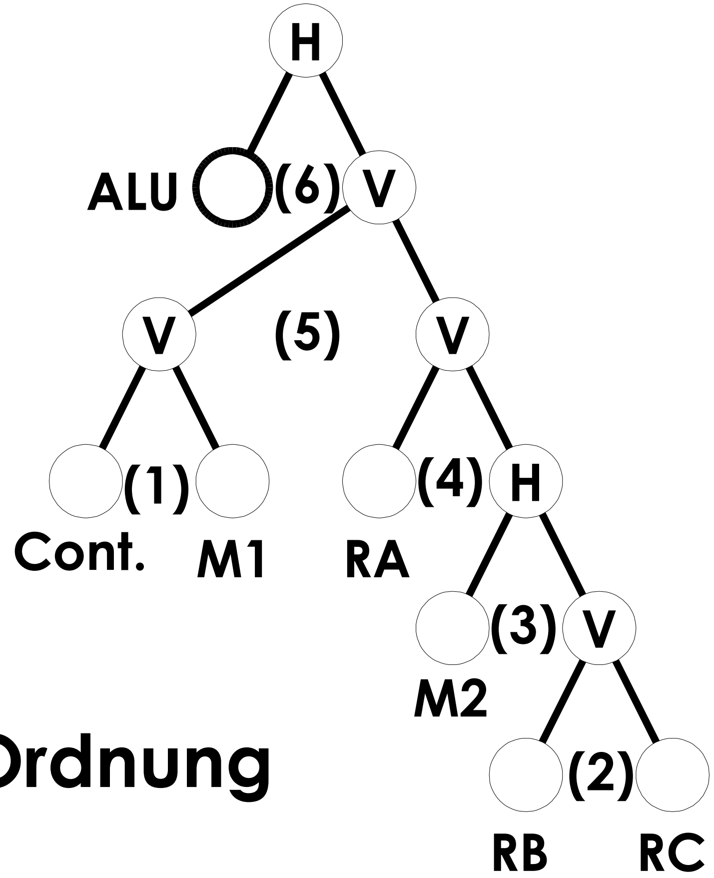

- **Darstellung durch Slicing Tree**
  - Knoten sind Schnitte oder Blattzellen
  - Schnitte nach Richtung getrennt
    - V: Linker Unterbaum *LINKS* von rechtem
    - H: Linker Unterbaum *UNTER* rechtem
- **Wird erzeugt z.B. bei Platzierung mit MinCut**
  - Hier aber allgemeiner!

# Globale Verdrahtung 7

- Für Slicing Floorplan: Einfach zu lösen
- CDP
  - Schnittlinien sind Kanäle
  - Kanalform abhängig von Reihenfolge
  - Festgelegt im Channel Ordering Problem
- COP
  - Grundlage ist Slicing Tree
  - DFS mit Post-Order Traversal
    - Numeriere bearbeitete Knoten aufsteigend
    - V-Schnitt: V-Kanal, Länge=Ober/Unterkante der Zellen
    - H-Schnitt: H-Kanal, Länge=linke/rechte Seite der Zellen

# Globale Verdrahtung 8




- CDP via Schnittrichtung, Ordnung
- COP via Slicing Tree
  - Post-Order DFS
  - Reihenfolge für Kanalverdrahtung

# Globale Verdrahtung 9

- Bei Non-Slicing Floorplans
  - Reine Kanalverdrahtung nicht ausreichend
  - Braucht
    - Switchbox Router
    - Dreiseitige Kanal-Router
      - Nur eine Kanalseite hat bewegliche Terminals
      - Verdrahtungsfläche ist fest (ähnl. Switchbox)

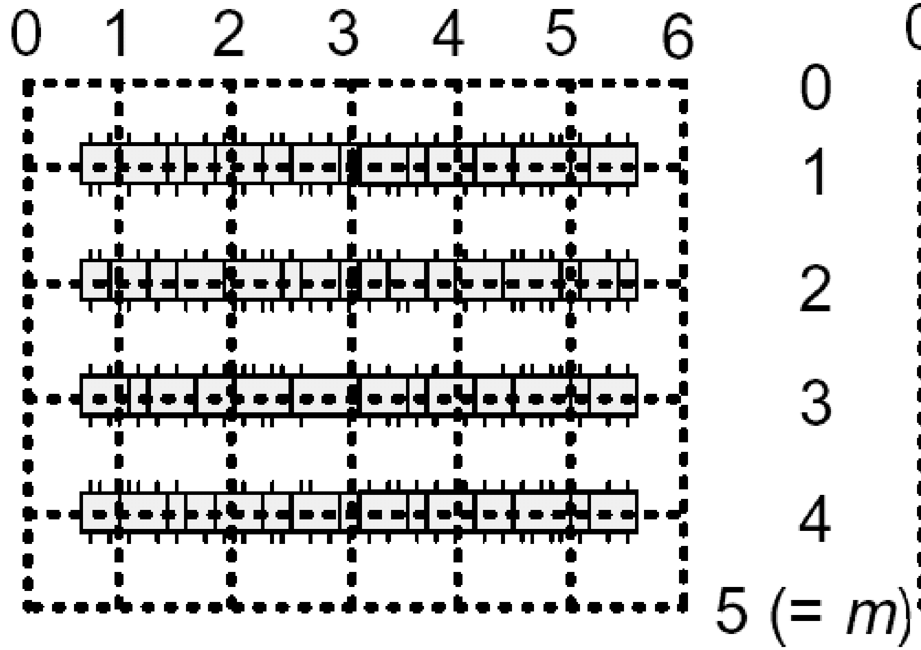
- Nach Lösung des CDP: Steiner-Baum
  - Bei Building Blocks der Regel keine Feedthroughs
  - Verdrahtung nur innerhalb der Kanäle
    - Sehe Kanäle als Kanten in Graph an
    - Löse Graphen-Version des minimalen Steiner-Baumes

# Zwischenstand

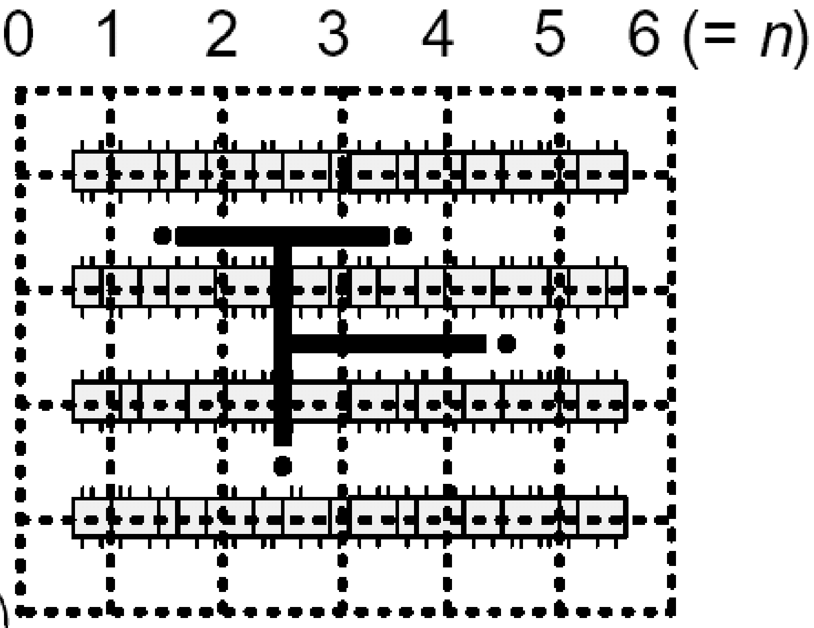
- **Kanalverdrahtung**
  - **Alle Terminals angrenzend an *einem* Kanal**
  - **Nun auch mit H- und V- Einschränkungen**
  - **Leitungsführung auf Zeilenebene in Kanal**
- **Globalverdrahtung**
  - **Terminals an *verschiedenen* Kanälen**
    - **Standardzellen**
  - **Leitungsführung auf Kanalebene**
  - **Nicht auf Zeilenebene**
  - **Teilweise erforderlich (building block layout)**
    - **Festlegen von Kanälen überhaupt (CDP)**
    - **Festlegen der Bearbeitungsreihenfolge (COP)**
    - **Einfach machbar bei Slicing Layouts**

# Modellierung 1

- Für Standardzelltechnologie
- Modellierung der Baum-Geometrie




m x n Matrix
V-Abstand variabel




Eingebetteter Baum
Verschmolzene Terminals

# Modellierung 2

- **Lokale vertikale Dichte $d_v(i,j)$**
  - **Leitungen durch V-Segment $i$-1,$i$ in Spalte $j$**
- **Lokale horizontale Dichte $d_h(i,j)$**
  - **Leitungen durch H-Segment $j$-1,$j$ in Zeile $i$**
- **Kanaldichte**

$$D_v(i) = \max_{j=1}^{n} d_v(i, j)$$

- **Gesamtkanaldichte**

$$D_T = \sum_{i=1}^{m} D_v(i)$$

- **Ziel: Minimiere $D_T$ mit $d_h(i,j) \leq M_{ij}$**
  - **$M_{ij}$: Verfügbare vertikale Feedthroughs im H-Segment $j$-1,$j$ in Zeile $i$**

# Mögliche Vorgehensweisen 1

- Variante von Lees Algorithmus
  - Erhöhe Überquerungskosten je Segment
    - Nach jedem Netz
  - Probleme
    - Versagt bei Auswahl aus vielen gleich guten Routen
    - Qualität abhängig von Netzreihenfolge

# Mögliche Vorgehensweisen 2

- Sequentieller Aufbau von RSMT je Netz
  - Bestimme Kantenkosten aus $d_v$, $d_h$
    - Umgehung von verstopften Gebieten *während* des Routings
    - Gute einzelne Routing-Ergebnisse
  - Qualität noch abhängig von Reihenfolge

# Mögliche Vorgehensweisen 3

- **Pseudo-simultanes Routing**
  - **Konstruiere unabhängigen RSMT je Netz**
    - **Immer optimale Route, unabhängig von Reihenfolge**
  - **Korrigiere Verstopfung (congestion) *später***

# Variante

- Hierarchische Vorgehensweise
  - Beginne mit 2x2 Raster über gesamten Chip
  - Löse globales Verdrahtungsproblem
- Für jeden der Quadranten
  - Unteraufteilung in eigenes 2x2 Raster
  - Löse globales Verdrahtungsproblem erneut
- Divide-and-Conquer Vorgehen

# Variante

- Im Extremfall: Bis hin zu einzelnen Terminals
  - Erledigt komplette Verdrahtung
  - Inklusive Kanalverdrahtung
- Optimalitätsprinzip gilt aber nicht!
  - Leitungen aus Partition hinaus beeinflussen Unterentscheidungen

# RSMT Problem

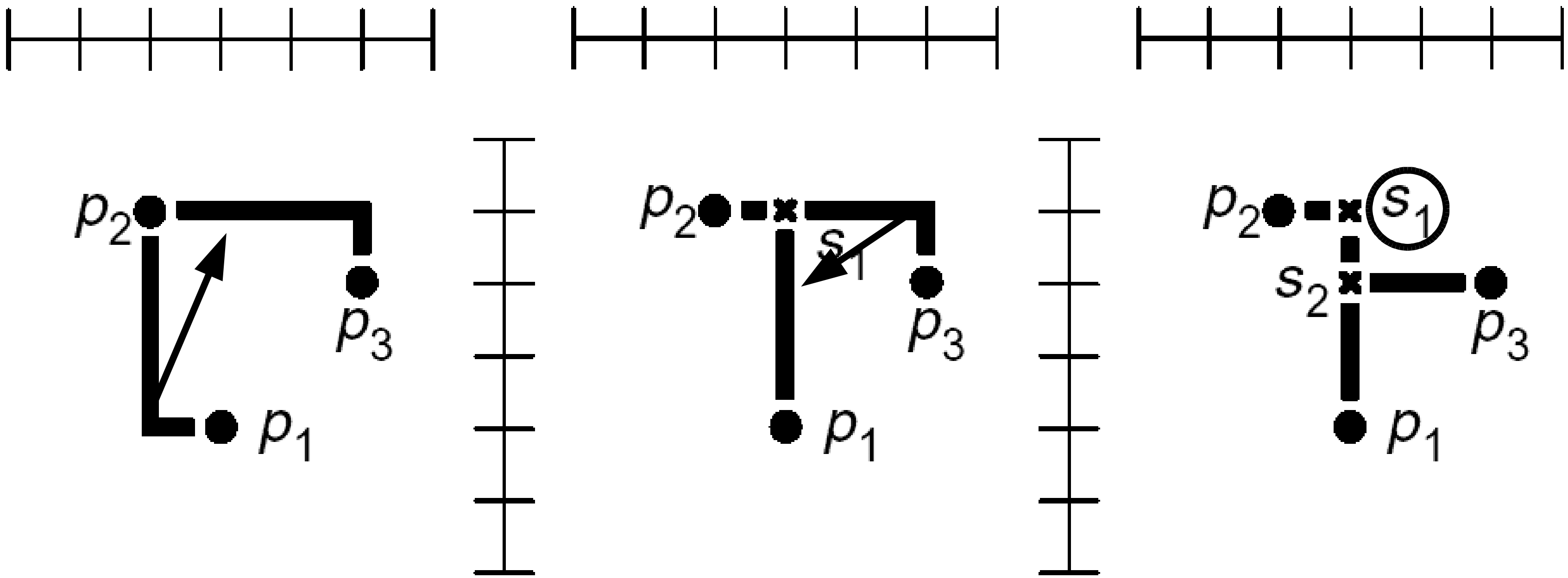
- **Rechtwinklige minimale Steiner-Bäume**
  - Nützlich zur Lösung von glob. Verdrahtungsproblemen
- **Gegeben**
  - P = {p1, p2, ...}: Punktmenge in der Ebene (2-D)
  - Distanzmetrik: $|x_i - x_j| + |y_i - y_j|$ (=Manhattan-Distanz)
- **Gesucht**
  - Finde verbindenden Baum für Punkte in P
    - Mit minimaler Gesamlänge!
  - Erlaube zusätzliche Punkte im Baum
    - Wenn sie zu kürzerer Gesamtlänge führen
    - Sogenannte „Steiner-Punkte“
- **Hier vernachlässigt**
  - Timing, Übersprechen

# Lösung

- Exakt: NP-vollständig
- Approximieren durch MRST
  - Minimaler rechtwinkliger aufspannender Baum
  - Prims Algorithmus: $O(n^2)$
    - Maximal 1.5x länger als echter Steiner-Baum
  - Idee: Hinterher Ergebnis verbessern
- Ausblick: Neuere Heuristiken
  - Verbesserter MRST max. 11/8x länger als RSMT
    - Fössmeier et al. 1997

# MRST Optimierung

- **Beispiel: Lokales Umlegen von L-Stücken**
  - **Führt zu Steiner Punkten**
  - **Ziel: Verschmelzen von Segmenten**
    - **Reduktion der Gesamtlänge**




- **Steiner-Punkte haben Grad $\geq$ 3**
  - **s1 verschwindet (kein Steiner-Punkt mehr)**

# Besser: *MRST*-Erweiterung

- Vorteil: Nicht schlechter als 4/3x RSMT
  - Auch wenn *MRST* schlechtestes Ergebnis liefert
    - Wenn MRST = 1.5x RSMT, verbesserter MRST ≤ 1.33x RSMT
- Beginnt mit *MRST* nach Prim
- Verfeinert dann schrittweise
  - Nimmt jeweils einzelnen Punkt *s* zu *P* hinzu
    - *s* ist also Steiner-Punkt
  - Wählt *s* dabei so, dass *MRST* ($P \cup \{s\}$) minimal
  - Wird „1-Steiner-Baum-Problem" genannt
- Wiederhole!
- Liefert beweisbar gute Ergebnisse
  - Kann aber keine optimale Lösung garantieren

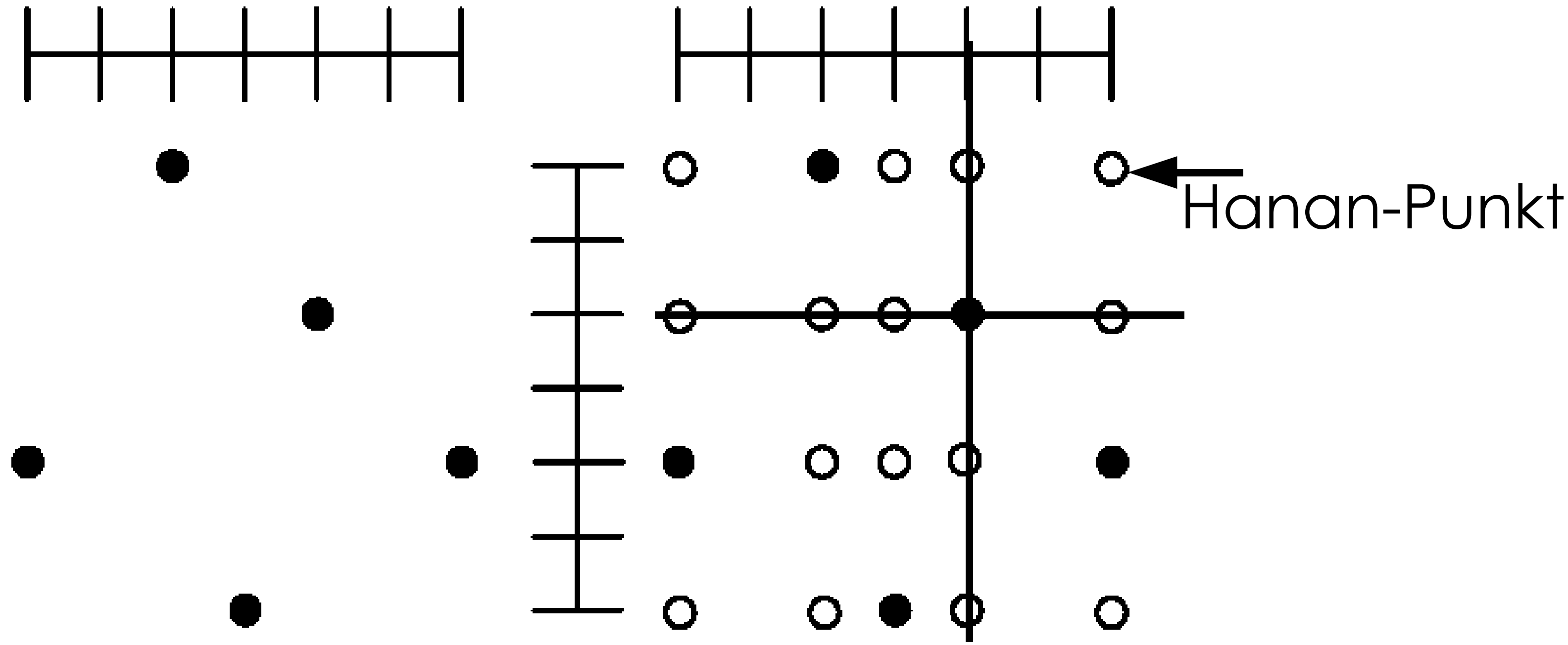
# Algorithmus steiner

```
pair<set<vertex>,set<edge>>
steiner(set<vertex> P) {
      set<vertex>     T;
      set<edge>       E, F;
      int gain;   // Längenverkürzung

      E = P.primMRST();
      (T,F,gain) = oneSteiner(P, E);
      while (gain > 0) {
            P = T;
            E = F;
            (T,F,gain) = oneSteiner(P, E);
      }
      return (P,E);
}
```

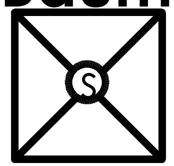
# 1-Steiner-Baum Konstruktion 1

- **Wie den Punkt *s* bestimmen?**
  - **Alle Punkte ausserhalb von *P* ausprobieren**
  - **... geht aber besser!**
- **Auf Hanan-Punkte beschränken (1966)**
  - **Hanan-Punkte liegen auf vorbesetzten Rasterlinien**
  - **Erlaubt trotzdem Finden des Optimums**



# 1-Steiner-Baum Konstruktion 2

- **Für Auswahl des besten Punktes *s***
  - Immer wieder MRST ($P \cup \{s\}$) via Prim bestimmen
    - Punkt mit kürzestem Baum wird genommen
  - Geht auch besser ...
- **Inkrementelle Berechnung des MRST**
  - Aus MRST ($P$) hin zu MRST ($P \cup \{s\}$)
    - In linearer Zeit $O(n)$, mit $n = |P|$
- **Idee**
  - Punkte im Baum haben max. Grad 4
  - *s* muss an Baum für *P* angeschlossen werden
  - Lage des *s* nächstgelegenen Punktes im Baum für *P*
    - In einer der Regionen N,E,S,W um *s*
      - *N,S:* $|d_x| \leq |d_y|$, *E,W:* $|d_y| \leq |d_x|$



# Algorithmus oneSteiner

```
triple<set<vertex>,set<edge>,int>
oneSteiner(set<vertex> V, set<edge> E) {
    int maxgain;        vertex maxpoint;
    int gain;
    set<vertex> W;      set<edge> F;

    maxgain = 0;
    foreach s ∈ „Hanan-Punkte von V“ do {
        (W,F,gain) = spanningUpdate(V,E,s);
        if (gain > maxgain) {
            maxgain    = gain;
            maxpoint   = s;
        }
    }
    if (maxgain > 0) {
        (W,F,gain) = spanningUpdate(V,E,maxpoint);
        return (W,F,gain);
    } else
        return (V,E,0);
}
```

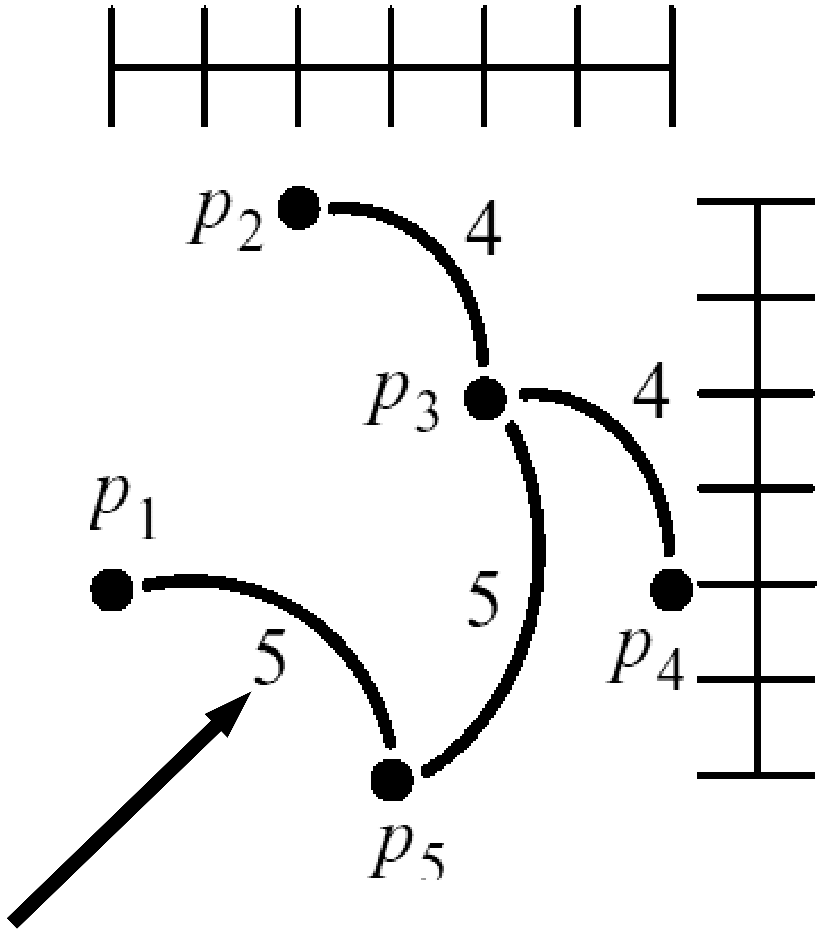
# Algorithmus spanningUpdate

```
triple<set<vertex>,set<edge>,int>
spanningUpdate(set<vertex> V, set<edge> E, vertex s) {
        int         delta;              // Längenverkürzung
        vertex      u, v, w;

        delta = 0;
        V = V ∪ {s};
        foreach dir ∈ {NORTH, EAST, SOUTH, WEST} do {
                u = s.closestPointInTree(V, dir);
                E = E ∪ {(s,u)};        // s an alle Partner anschliessen
                delta = delta – distance(s,u); // Hier Verlängerung!
                if (hasCycle(V, E)) {
                        (v,w) = findLongestCycleSegment(V, E);
                        E = E \ {(v,w)};
                        delta = delta + distance(v,w); // wieder verkürzen
                }
        }
        return (V, E, delta);
}
```

# Beispiel Schritt 1

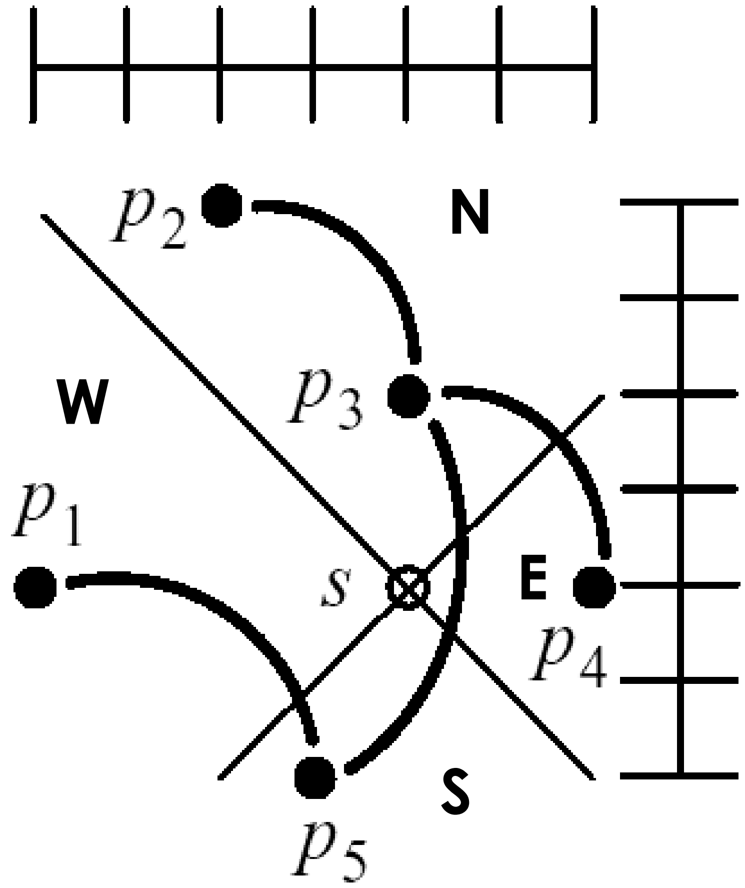
**Eingabe: MRST, z.B. via Prims Algorithmus**




**Bögen geben nur Distanz an, noch keine genaue Führung**

# Beispiel Schritt 2

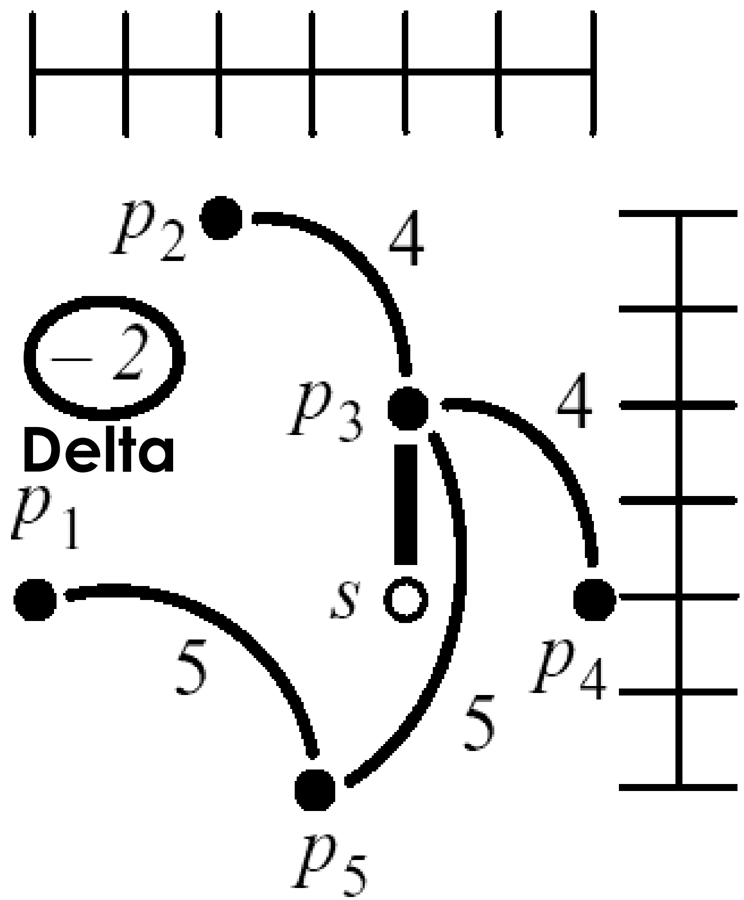
**Hinzunahme eines ersten Hanan-Punktes *s***




***s*** **nahegelegenste Punkte aus *P*: $p_3$, $p_4$, $p_5$, $p_1$**

# Beispiel Schritt 3

**Anbinden an den ersten *s* benachbarten Punkt $p_3$ im N**

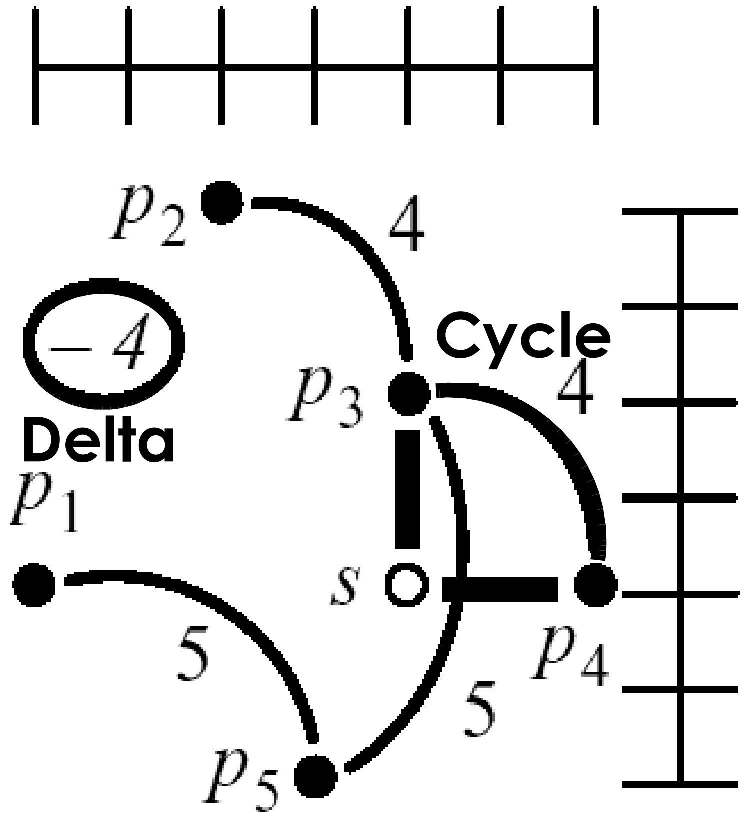



**Nun festgelegte kürzeste Führung, Erhöhung der Länge**

**- Feste Verbindung für Punkte auf derselben Rasterlinie**

# Beispiel Schritt 4

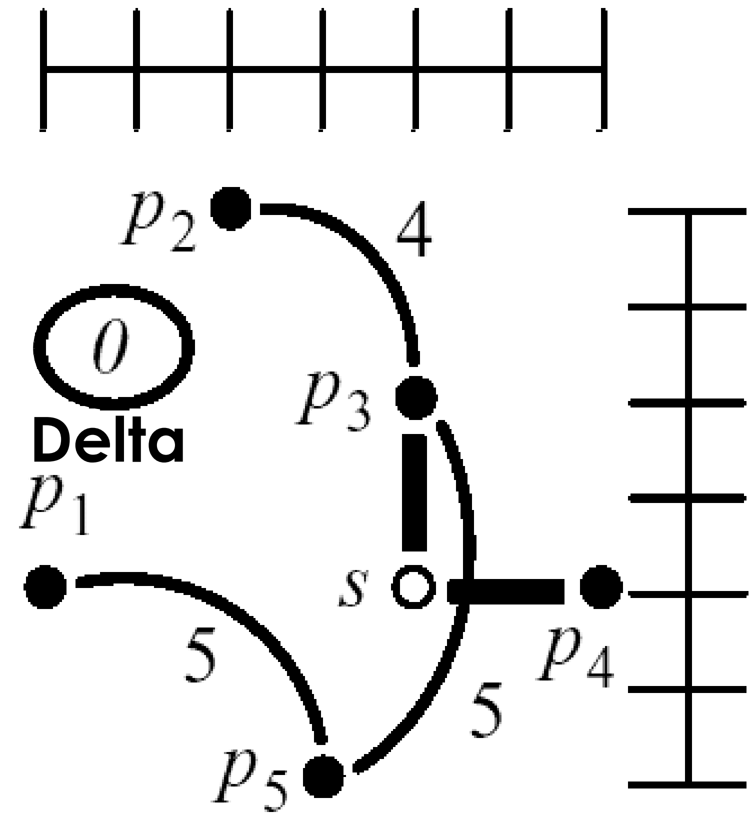
**Anbinden an den zweiten *s* benachbarten Punkt $p_4$ im E**




**Auch festgelegte Führung und Erhöhung der Länge, Zyklus**

# Beispiel Schritt 5

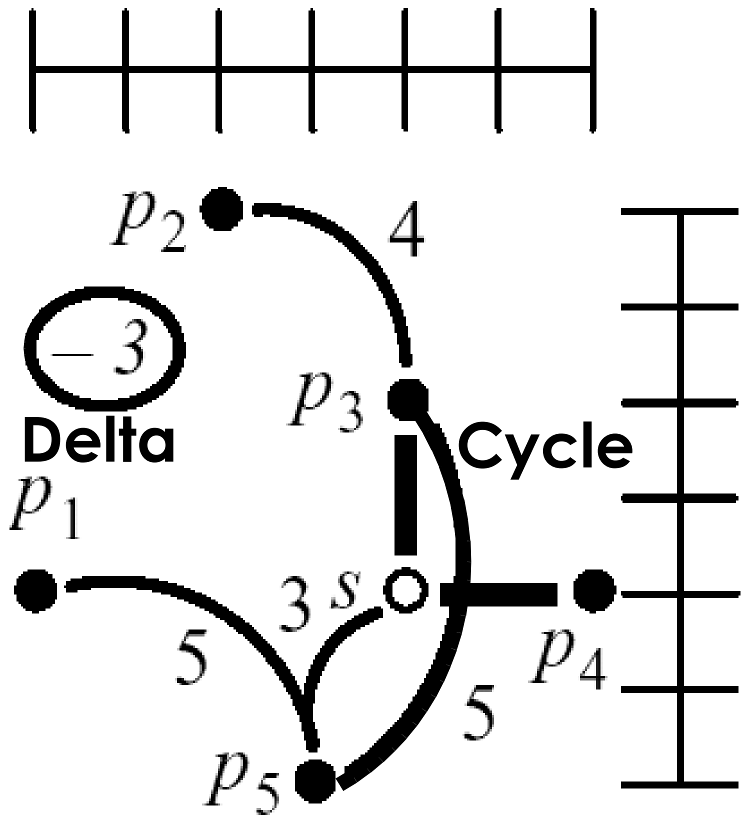
**Entferne längste Kante d({$p_4$,$p_3$})=4 aus Zyklus**




**Gesamtlänge verkürzt sich nun um 4**

# Beispiel Schritt 6

**Anbinden an den dritten *s* benachbarten Punkt $p_5$ im *S***

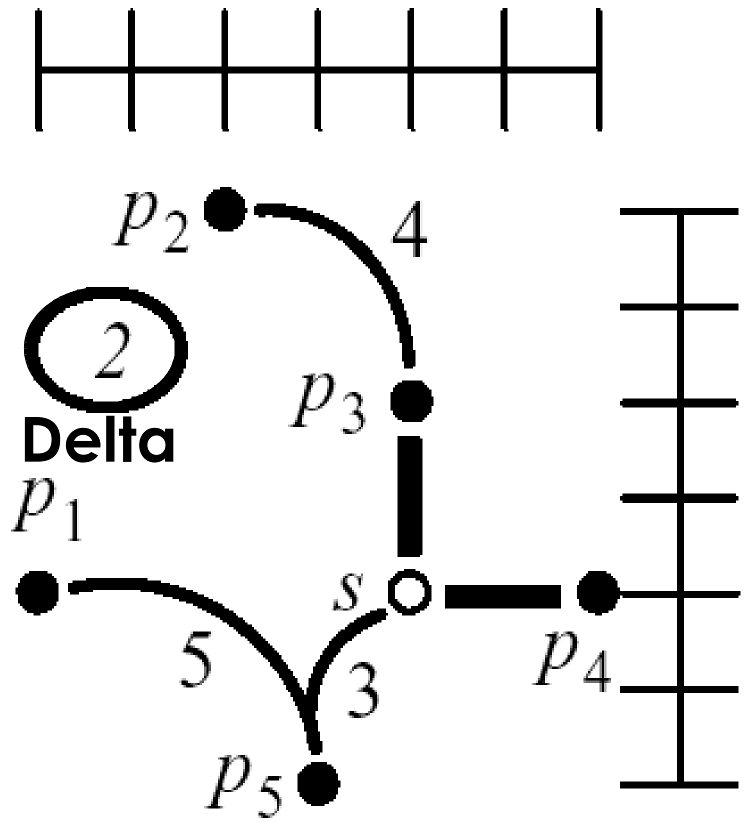



**Noch keine feste Führung, Gesamtlänge erhöht sich, Zyklus**

**- *s* und *p5* nicht auf derselben Rasterlinie**

# Beispiel Schritt 7

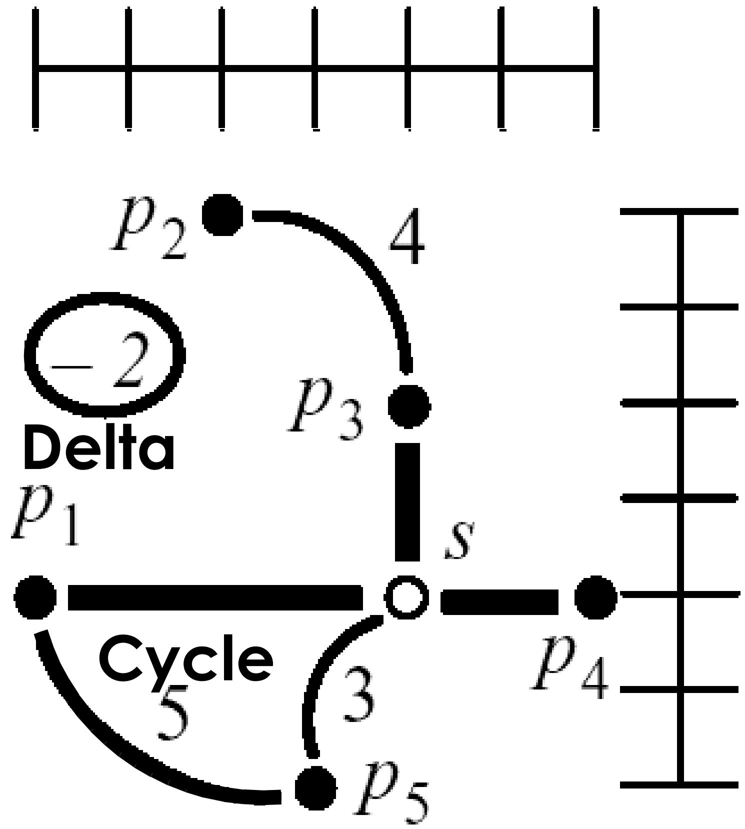
**Entferne längste Kante d({$p_5$,$p_3$})=5 aus Zyklus**




**Gesamtlänge verkürzt sich nun um 5**

# Beispiel Schritt 8

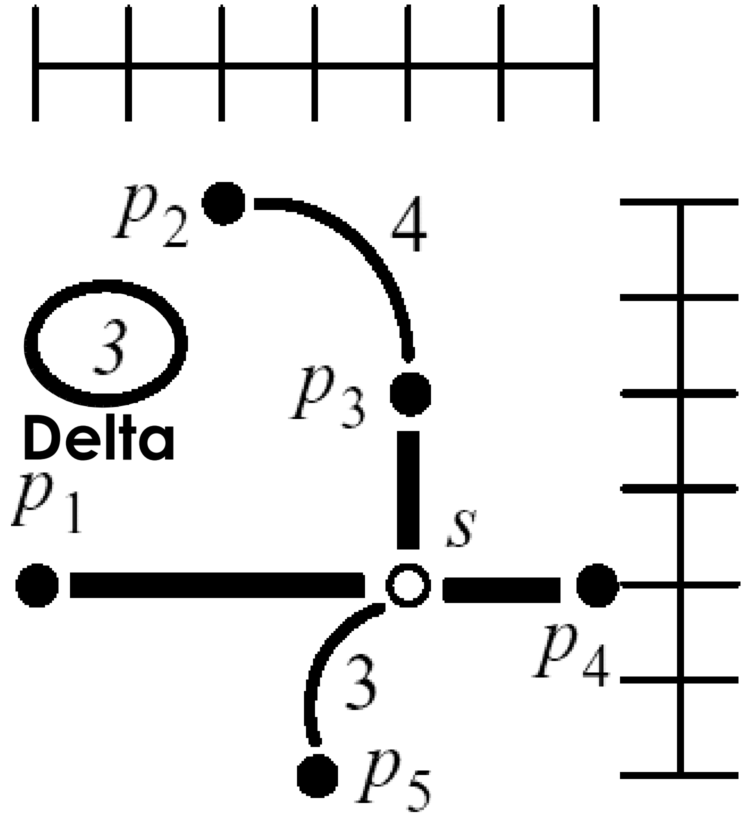
**Anbinden an den vierten *s* benachbarten Punkt $p_1$ im W**




**Feste Führung, Gesamtlänge erhöht sich, Zyklus**

# Beispiel Schritt 9

Entferne längste Kante d({$p_5$,$p_1$})=5 aus Zyklus




Gesamtlänge verkürzt sich nun um 5, Gesamtgewinn ist 3

# Komplexität

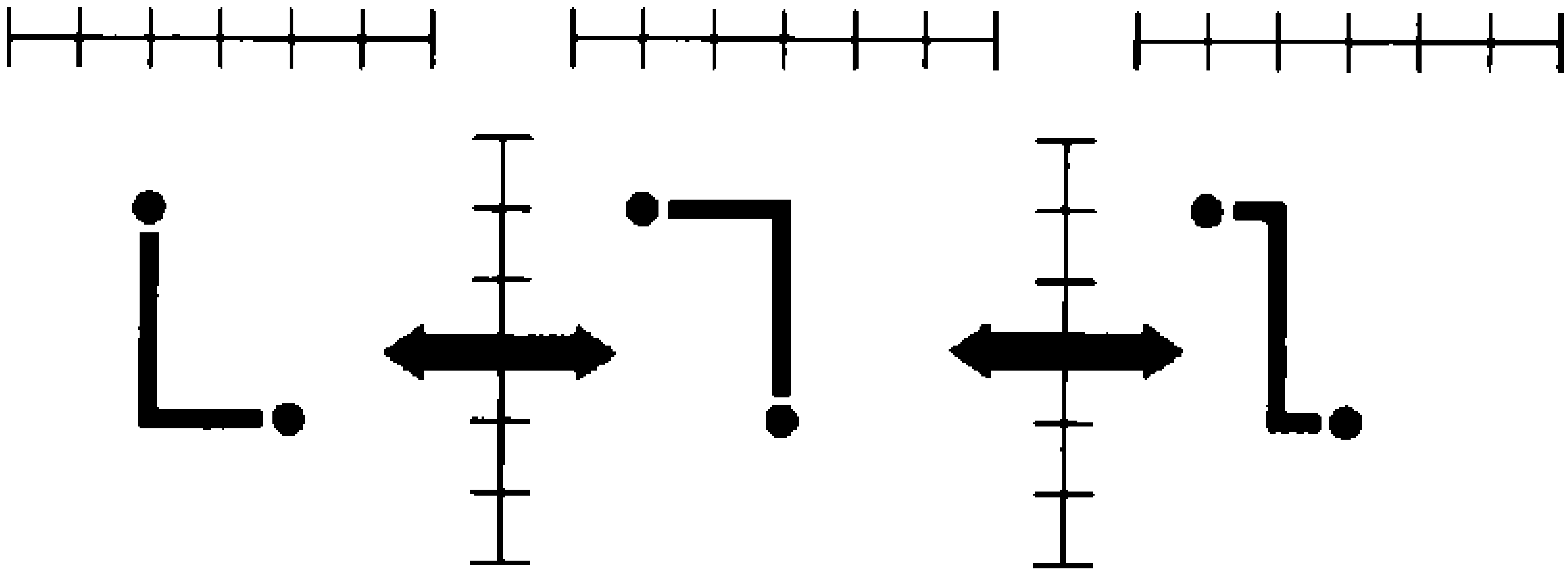
- **spanningUpdate()**
  - 4x closestPoint(): O($n$)
  - hasCycle(): DFS mit History, O($n$)
  - findLongestCycleSegment(): History, O($n$)
  - ⇨ Gesamt: O($n$)
- **Anzahl Hanan-Punkte: O($n^2$)**
- **oneSteiner() Gesamt: O($n^3$)**
- **steiner() Gesamt: O($n^5$)**

- **Im Durchschnitt aber besser**
  - z.B. oneSteiner() nur 2x aufgerufen bei $n$=40
  - ⇨ O($n^3$)

# Beseitigen von Verstopfungen

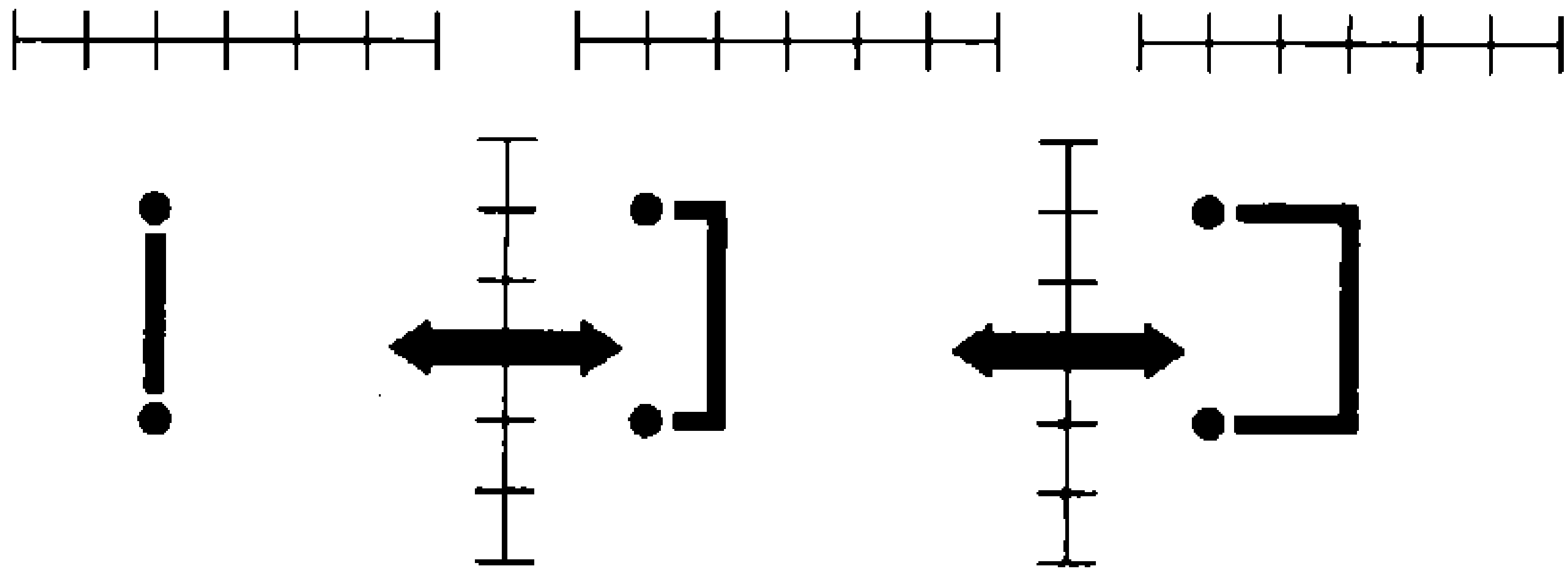
- Bisher unabhängige RSMTs: Einer je Netz
  - Ähnlich dem ersten Durchgang bei PathFinder
- Nachfrage nach V-Feedthroughs
  - Bestimmen
  - Stark verstopfte Stellen entlasten
- Wie? Lokale Transformation der einzelnen RSMTs
  - Kontrolliert durch eigene Optimierung
    - Z.B. Simulated Annealing oder Nachbarsuche

# Lokale Transformation 1



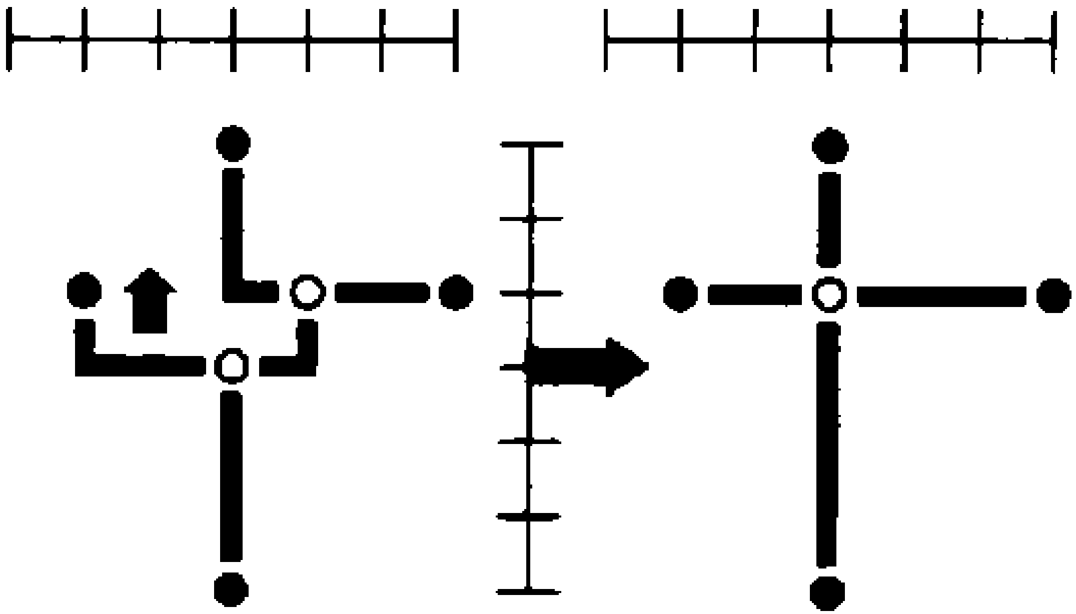
- Variiere konkrete Führung einer Kante
- Länge bleibt gleich

# Lokale Transformation 2



- Länge erhöht sich
  - Kann aber Gesamtkosten senken

# Lokale Transformation 3



- Kompliziertere Verschiebung
  - Vollständiges Entfernen von Steiner-Punkten
- Im Notfall: Maze-Routing
  - Nun bessere Umgebung

# Zusammenfassung

- Yoeli's Robuster Router
  - Beispiel für komplexere Heuristik
    - Regeln
    - Ausführliches Beispiel
- Globalverdrahtung
  - Abhängig von Zieltechnologien
- Steiner-Bäume
    - Optimierungsziele
- Routing in Slicing-Floorplans
    - CDP, COP
- Globale Verdrahtung für Standardzellen
  - Konstruktion von Steiner-Bäumen
  - Lokale Optimierung