

Algorithmen im Chip-Entwurf 3

Timing-Analyse und Heuristiken

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Organisatorisches

- **Gute Nachricht für 2 SWS'ler**
 - Ohne vollständiges Programmierprojekt
 - Erste Aufgabe nur für Klausurpunkte
- **Vereinfachung**
 - Keine Timing-Analyse mehr erforderlich
- **Gestreckter Zeitplan**
 - Abgabe am Montag, den 20.11.06
 - Bis 23:59 Uhr
- **Nochmal: Das gilt nicht für 4 SWS'ler**
 - Mit vollständigem Programmierprojekt
 - Dafür ohne Klausur

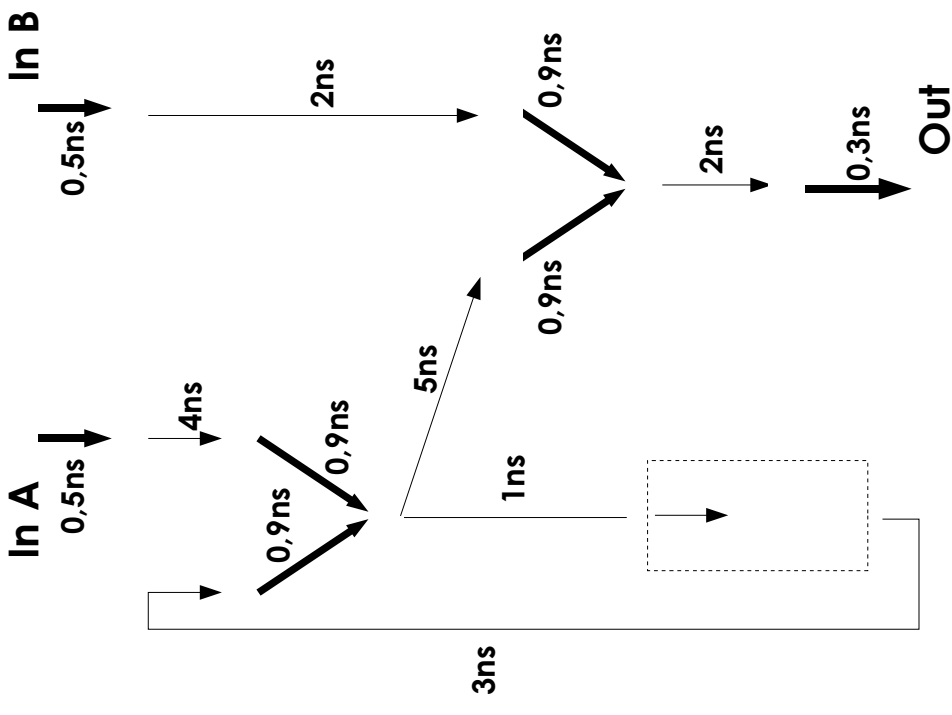
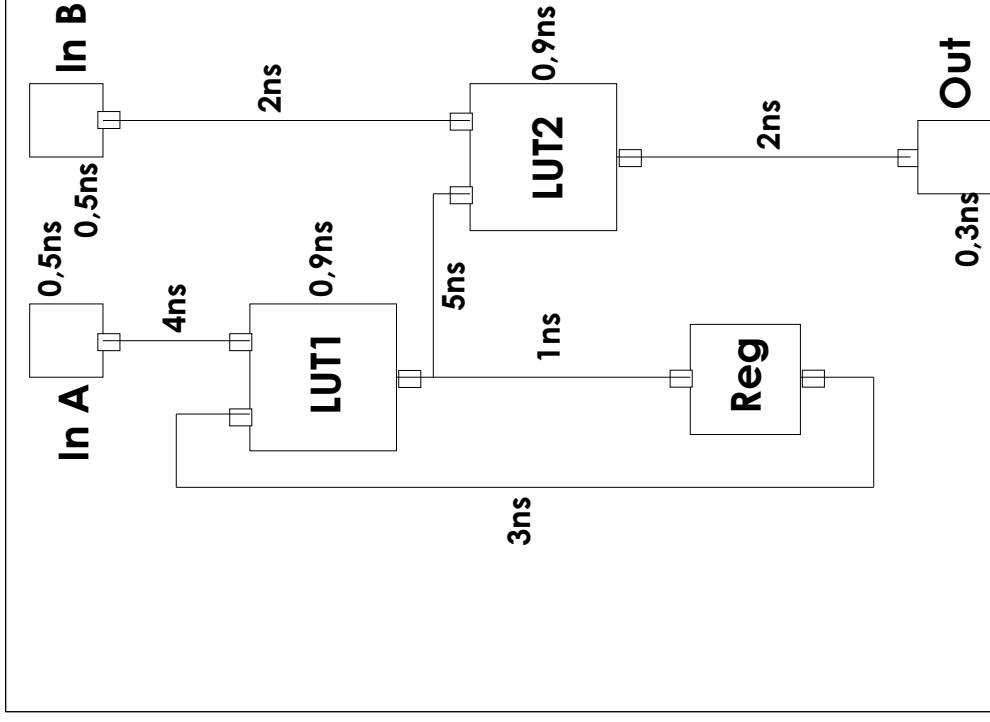
Übersicht

- **Timing-Analyse**
- **Vereinfachtes Beispielproblem**
 - **Unit-Size Placement Problem (Upp)**
- **Heuristiken**
 - **Nachbarsuche**
 - **Simulated Annealing**
 - **Tabu-Suche**
 - **Genetische Algorithmen**
- **Zusammenfassung**

Grundlagen Timing-Analyse

- **Wozu?**
 - Analysiere fertige Layouts
 - Analysiere einzelne Verbindungen während Layouterzeugung
 - ◆ Erkenne kritische Verbindungen
 - ◆ Behandle diese mit Vorrang
- **Worauf?**
 - Schaltungselemente
 - ◆ Gatter, Wertetabellen (*LUT*), Register, I/O-Blöcke, ..
 - ◆ Bleiben konstant, exakte Verzögerungen bekannt
 - Netze
 - ◆ Nur nach Layouterzeug. bekannt, vorher schätzen

Modellierung



- Auf „4-partitem“ Graph
- Externe Ein-/Ausgänge, Ein-/Ausgangs-Ports

Berechnung Ankunftszeit

- Ankunftszeit (Arrival) an Knoten v :

$$T_a(v) = \underset{(u,v) \in E}{\text{Max}} (T_a(u) + w(u, v))$$

- Idee: BFS oder zyklenfreier LP
 - Beginne mit $T_a(v) = 0$ mit Knoten v :
 - ◆ Externer Eingang, Registerausgang
 - Bearbeite Knoten mit bearbeiteten Vorgängern
 - Späteste Gesamtankunftszeit $D_{\max} = \text{Taktper.}$
 - ◆ An externem Ausgang oder Registereingang
 - ❖ Im Beispiel 13,6ns

Spätestmögliche Ankunftszeit

- **Wie unwichtig sind unkritische Netze?**
 - Idee: Verschiebbare Elemente bei Kompakt.
 - Hier auf Zeitintervalle anwenden (*slack*)
 - „Wieviel langsamer kann ein Netz werden, ohne dass die gesamte Schaltung leidet?“

- **Berechnung**
 - **Mittels spätestmöglicher Ankunftszeit**
 - ◆ Required time $T_r(u)$ an Knoten u
 - ◆ Spätestmöglicher Ankunftszeitpunkt von Signalen
 - ❖ Sonst Verlangsamung der ganzen Schaltung
 - ◆ **Analog Kompaktierungsbeispiel**
 - ❖ Rechteste Position ohne Breitenvergrößerung

Berechnung $T_r(u)$ & $\text{slack}(u, v)$

- Beginne mit $T_r(u) = D_{\max}$ bei Knoten u :
 - Externer Ausgang, Registereingang
- Nun BFS/LP rückwärts
- Bearbeite Knoten
 - Nur mit komplett bearbeiteten Vorgängern
 - ◆ Rückwärts: Vorgänger hier sind sonst Nachfolger!

$$T_r(u) = \underset{(u, v) \in E}{\text{Min}} (T_r(v) - w(u, v))$$

- Slack einer Verbindung von u nach v

$$\text{sk}(u, v) = T_r(v) - T_a(u) - w(u, v)$$

- Beachte: Auf kritischem Pfad $\text{slack} = 0$

Beispiel s27.critical

```
Node: 4 INPAD_SOURCE Block #2 (s27_in_3_)
T_arr: 0 T_req: -3.88578e-16 Tdel: 5e-10

Node: 5 INPAD_OPIN Block #2 (s27_in_3_)
Pin: 0
T_arr: 5e-10 T_req: 5e-10 Tdel: 5e-09
Net to next node: #2 (s27_in_3_). Pins on net: 5.

Node: 12 CLB_IPIN Block #6 (s27_out)
Pin: 0
T_arr: 5.5e-09 T_req: 5.5e-09 Tdel: 0

Node: 17 SUBBLK_IPIN Block #6 (s27_out)
Pin: 0 Subblock #0
T_arr: 5.5e-09 T_req: 5.5e-09 Tdel: 9e-10

Node: 21 SUBBLK_OPIN Block #6 (s27_out)
Pin: 4 Subblock #0
T_arr: 6.4e-09 T_req: 6.4e-09 Tdel: 0

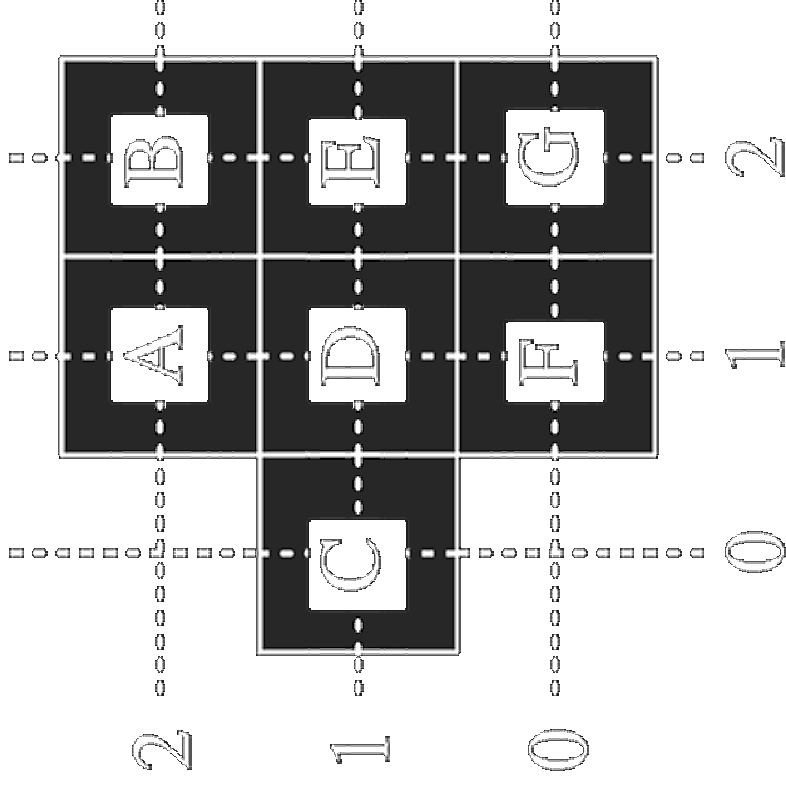
Node: 16 CLB_OPIN Block #6 (s27_out)
Pin: 4
T_arr: 6.4e-09 T_req: 6.4e-09 Tdel: 1e-09
Net to next node: #5 (s27_out). Pins on net: 2.

Node: 10 OUTPAD_IPIN Block #5 (out:s27_out)
Pin: 0
T_arr: 7.4e-09 T_req: 7.4e-09 Tdel: 3e-10

Node: 11 OUTPAD_SINK Block #5 (out:s27_out)
T_arr: 7.7e-09 T_req: 7.7e-09
```

```
Tnodes on crit. path: 8 Non-global nets on crit. path: 2.
Global nets on crit. path: 0.
Total logic delay: 1.7e-09 (s) Total net delay: 6e-09 (s)
```

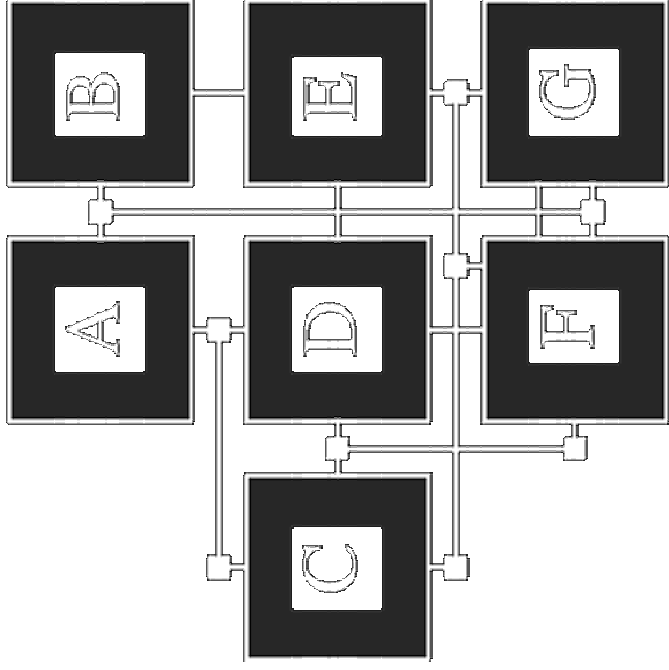
Beispielanwendung UPP 1



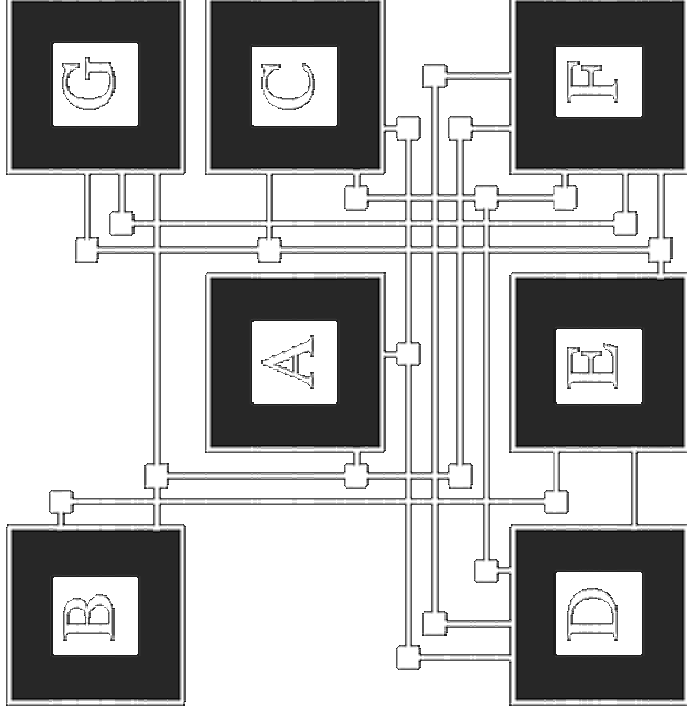
- Eingabe:
 - 1x1 Zellen
 - Netzliste
- Platziere Zellen
 - Auf 1x1 Raster
 - Überlappungsfrei
- Minimiere Fläche
 - Platz für Verdrahtung

n1: A, B, F, G n5: C, D, F
n2: B, E n6: C, E, F, G
n3: D, E n7: D, F
n4: A, C, D n8: F, G

Unit-Size Placement 2



**Plazierung mit
Verdrahtung**



**Schlechtere Plazierung
■ Mehr Verdrahtungsspuren**

- Problem: Bestimmung der Qualität**
- **Komplette Verdrahtung dauert zu lange**
 - **Abschätzen**

Art der Probleme

- **Viele Probleme im Bereich VLSI CAD sind**
 - NP-vollständig
 - NP-hart
 - ◆ Mindestens so aufwendig wie NP-vollständig
- **Exakt lösbar nur für kleine Problemgrößen**
- **Falls sub-optimale Lösungen akzeptabel**
 - Näherungsverfahren
 - ◆ Garantieren eine vorgegebene Lösungsqualität
 - ◆ Nicht allgemein formulierbar
 - Heuristiken
 - ◆ In der Praxis: Schwankende Lösungsqualität

Darstellung einer „Lösung“

- **Problem-spezifisch**
- **Algorithmen-spezifisch**
- **Grundsätzlich unterscheidbar**
 - **Vollständige Lösung**
 - ◆ **Alle Unbekannten haben gültige Werte**
 - **Algorithmus könnte beliebig beendet werden**
 - **Unvollständige Lösung**
 - ◆ **Einige/alle Unbekannte sind noch unbestimmt**
 - **Algorithmus muss weiterrechnen**

Definitionen

- Instanz $I = (F, c)$
 - Lösungsraum F
 - Kostenfunktion $c: F \rightarrow \mathbb{R}$
- Lösung $\underline{f} \in F: \underline{f} = [f_1, \dots, f_n]^T$
 - Explizite Einschränkungen: Wertebereiche f_i
 - Implizite Einschränkungen: Abhängigkeiten
- Teillösung $\tilde{\underline{f}}$
 - Einige f_i undefiniert
 - Spannt Unterraum von F auf

Nachbarsuche 1

- **Starte mit einer vollständigen Lösung**
- **Bestimme „Nachbarn“ der Lösung**
 - **Andere Lösungen „nahe“ an existierender**
 - **Definition von „Nähe“ ist problemspezifisch**
- **Wähle „besseren“ Nachbarn aus**
- **Wiederhole**

Nachbarsuche 2

- **Formal**
 - Problem $I = (F, c)$
 - Lösung $f \in F$
 - Nachbarschaft $N: F \rightarrow 2^F$
 - ◆ Potenzmenge 2^F : Menge der Untermengen von F
 - Nachbar $g \in N(f)$
- **Beispielzug UPP: Vertausche zwei Zellen**
 - n Zellen, $(n-1)$ Partner
 - Komplexere Züge möglich
 - ◆ Tausche 3 Zellen, tausche Regionen, ...

$$|N(\vec{f})| = \frac{n(n-1)}{2}$$

Nachbarsuche 3

- Welches $g \in N(\underline{f})$ wählen?
- Ziel: Kostenreduzierung bezüglich c
- Also wähle g mit
$$c(g) < c(\underline{f})$$
- Ende mit \underline{f} bei
$$c(g) \geq c(\underline{f}) \text{ für alle } g \in N(\underline{f})$$

Nachbarsuche 4

```
local_search() {
  feasible_solution f;
  set<feasible_solution> G;

  f := initial_solution();
  do {
    G := {g | g ∈ N(f) ∧ c(g) < c(f)};
    if (G ≠ ∅)
      f := G.pickany();
  } while (G ≠ ∅);
  report(f);
}
```

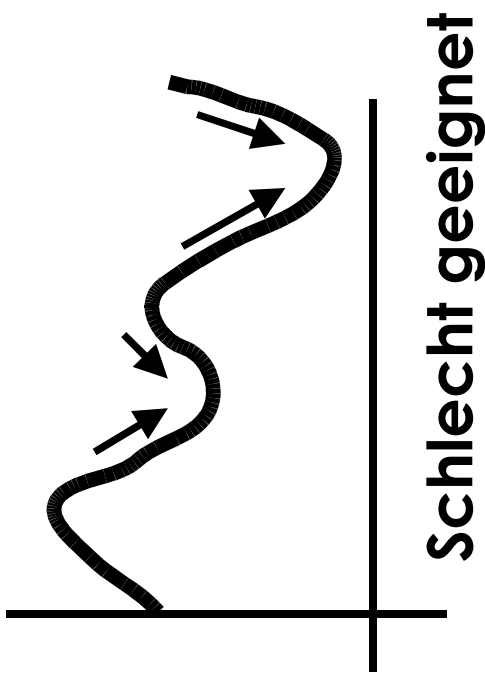
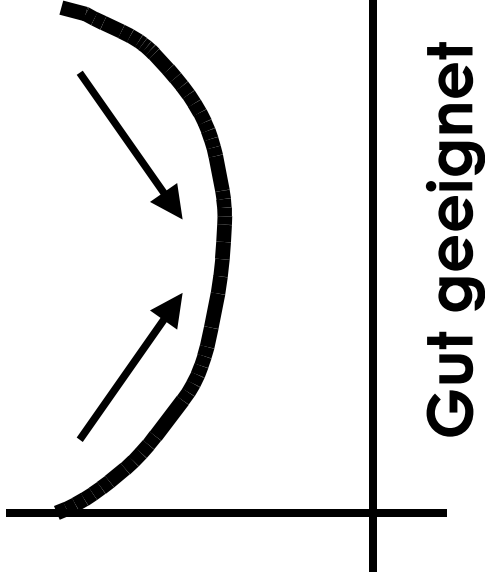
■ Initialisierung

■ Strategien

- Erste Verbesserung
- Steilster Abstieg

Nachbarsuche 5

■ „Form“ der Kostenfunktionen



- **Steckenbleiben in lokalen Minima**
 - Betrachte größere Nachbarschaften
 - Mehrere Läufe mit anderen Startlösungen
 - Adaptiere Größe der Nachbarschaft

Simulated Annealing 1

- **Akzeptiere verschlechternde Züge**
 - **Aber bessere Strategie als reiner Zufall!**
- **Simulated Annealing (SA)**
 - **Simuliertes Erstarren**
 - **Inspiziert vom physikalischen Erstarrungsprozessen**
 - ◆ **Schnelles Erstarren ("Schockfrostern")**
 - ❖ **Hohe innere Spannung = hohe Energie**
 - ◆ **Langsames Abkühlen**
 - ❖ **Niedrige innere Spannung = niedrige Energie**

Simulated Annealing 2

- **Physik**
 - **Hohe Anfangstemperatur (flüssiges Material)**
 - **Moleküle können sich frei anordnen**
 - **Langsames Abkühlen**
 - **Bewegungsfreiheit wird schrittweise weiter eingeschränkt**
 - **Moleküle ordnen sich in Konfiguration niedrigster Energie an**
 - **Am besten bei *sehr langsamer* Abkühlung**

Simulated Annealing 3

- Optimierung
 - Energie entspricht Kostenfunktion
 - Bewegung der Moleküle entspricht Zügen
 - Temperatur entspricht Kontrollparameter T
 - ◆ Wie frei dürfen sich Moleküle bewegen?
= Welche Züge sind noch akzeptabel?
 - ◆ Niedrigere Energie/Kosten: Immer akzeptiert

$$c(\vec{g}) \leq c(\vec{f})$$

- ◆ Höhere Energie/Kosten: Akzeptiert bei

$$\Delta c = c(\vec{g}) - c(\vec{f}) \quad \text{mit} \quad e^{-\frac{\Delta c}{T}}$$

Simulated Annealing 4

$$\Delta c = c(\vec{g}) - c(\vec{f})$$

mit

$$e^{\frac{-\Delta c}{T}} = \frac{1}{e^{\frac{\Delta c}{T}}}$$

- **Hohe Temperaturen**
 - Akzeptiere fast alle schlechten Züge
- **Niedrige Temperaturen**
 - Akzeptiere fast keine schlechten Züge mehr
- **Physik: Boltzmann-Verteilung**
 - Statistische Mechanik

Simulated Annealing 5

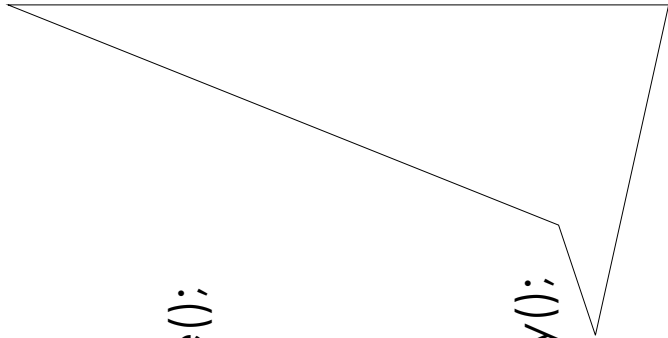
```
feasible_solution bsf;

simulated_annealing() {
  feasible_solution f, g;
  float T;

  T := initial_temperature();
  f := initial_solution();
  bsf := f;

  do {
    do {
      g := N(f).pickany();
      if (accept(f, g))
        f := g;
    } while (!thermal_equilibrium(T));
    T := new_temperature(T);
  } while (!stop());

  report(bsf);
}
```



```
int accept(feasible_solution f, g) {
  float Δc;

  Δc := c(g) - c(f);
  if (Δc ≤ 0) {
    if (c(g) < c(bsf))
      bsf := g;
    return (1);
  } else
    return (exp(-Δc/T) > random(1));
}
```


Simulated Annealing 6

- **initial_temperature()**
 - Bestimmt ausreichend hohe Starttemperatur
- **initial_solution()**
 - Bestimmt Startlösung
 - ◆ Zufällige, aber gültige Lösung OK!
- **thermal_equilibrium()**
 - Gleichgewicht auf einer Temperaturstufe
- **new_temperature()**
 - Bestimmt nächsten Temperaturschritt
- **stop()**
 - Abbruchkriterium
- **BSF: „Best so far“**, **beste bisherige Lsg.**
 - Letzte Lösung ist nicht immer die beste!

Simulated Annealing 7

- **TimberWolf: Standard Cell-Placer**
 - **Start mit $T = 4.000.000$**
 - **Stop bei $T < 0,1$**
 - **Equilibrium abhängig von Problemgröße**
 - ◆ **100 Züge pro Zelle bei 200 Zellen**
 - ◆ **700 Züge pro Zelle bei 3000 Zellen**
 - **Abkühlen**
 - ◆ **Anfangs mit $T_n = 0,8 T$**
 - ◆ **Im Mittelbereich mit $T_n = 0,95 T$**
 - ◆ **Gegen Ende mit $T_n = 0,8 T$**
- Cooling Schedule**

Simulated Annealing 8

- Bei geeigneter Cooling Schedule
 - SA findet *immer* die optimale Lösung
 - Praktisch aber nicht relevant (zu langsam)
- Viele Variationsmöglichkeiten
 - stop() abhängig von accept()
 - Adaptive Cooling Schedules
- Bibliotheken: ASA, EBSA
- SA ist allgemein verwendbar
- Aber: Spezialisierte Lösungen sind besser

Tabu Suche 1

- **Simulated Annealing**
 - Aufsteigende Züge zu Beginn akzeptiert
- **Tabu-Suche (TS)**
 - Aufsteigende Züge werden *immer* akzeptiert
 - Gehe *immer* zu $g \in N(\underline{f})$ mit

$$c(\vec{g}) = \min_{\vec{h} \in N(\vec{f})} c(\vec{h})$$

- Auch, wenn $c(\underline{g}) > c(\underline{f})$!
- Problem: Zyklen
 - ◆ Ständige Wiederholung der letzten Züge

Tabu-Suche 2

- Lösung: Verbiete letzte k Lösungen
 - Lösungen sind als „tabu“ markiert
 - Vermeidet Zyklen der Länge k
- Realisierung
 - FIFO der Länge k von Lösungen

Tabu-Suche 3

```
tabu_search() {
  feasible_solution f, g, bsf;
  set<feasible_solution> G;
  FIFO<feasible_solution,k> Q;

  Q := ∅;
  f := initial_solution();
  bsf := f;
  do {
    G := {s | s ∈ N(f) ∧ s ∉ Q};
    if (G ≠ ∅) {
      g := G.findmin(c);
      Q.shifftin(g);
      f := g;
      if (c(f) < c(bsf))
        bsf := f;
    }
  } while (G ≠ ∅ or stop());
  report(bsf);
}
```

Tabu-Suche 4

- **stop()**
 - „Keine Verbesserung in den letzten k Zügen“
- **UPP-Beispiel 10.000 Zellen**
 - Lösung beschreibt 10.000 Koordinatenpaare
 - Sehr große Tabu-Liste
- **Abhilfe: Setze nur einzelne Züge Tabu**
- **Aber: Einschränkung des Lösungsraumes**
- **Viele Variationsmöglichkeiten**
- **Kein theoretischer Hintergrund**
 - Erreichen des Optimums?
 - Wie stop() oder k wählen?

Genetische Algorithmen 1

- Auch hier
 - Umgang mit vollständigen Lösungen
- Aber: **Gleichzeitig mehrere Lösungen**
 - Menge P von Lösungen: Population
 - Generation k
 - Ersetze $P^{(k)}$ durch $P^{(k+1)}$ während Optimierung
- Bestimmung von $\bar{f}^{(k+1)} \in P^{(k+1)}$ mit
 - $\bar{f}^{(k)}, \bar{g}^{(k)} \in P^{(k)}$: Eltern von $\bar{f}^{(k+1)}$
 - Vererbung von Eigenschaften von $\bar{f}^{(k)}, \bar{g}^{(k)}$
 - ◆ Crossover
 - Ggf. Mutation von $\bar{f}^{(k+1)}$

Genetische Algorithmen 2

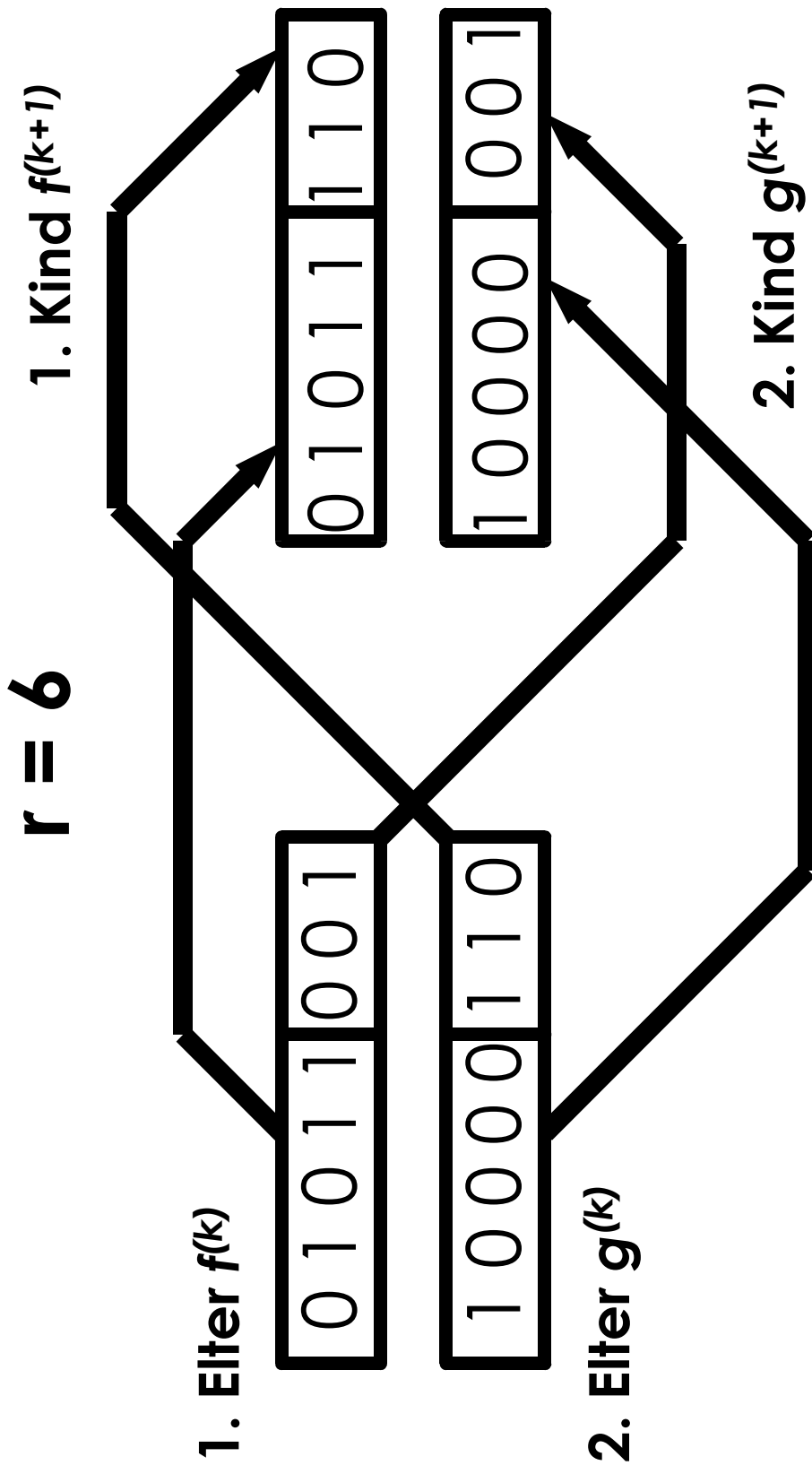
- **Kodierung bestimmt Operationen**
 - **Beispiel: Bitfolge für Lösungsvektor \bar{f}**
 - ◆ Chromosom
 - **UPP: 100 Zellen, 10x10 Raster**
 - ◆ 4 bit pro Koordinate
 - ◆ 8 bit pro Koordinatenpaar
 - ◆ 100 x 8 bit = 800 bit lange Bitfolge als Chromosom
 - ◆ L = Länge des Chromosoms in Bit
- **Wichtige Unterscheidung zwischen**
 - **Lösung**
 - ◆ Biologie: Phänotyp
 - **Kodierung der Lösung**
 - ◆ Biologie: Genotyp
- **Hier aber äquivalent benutzt**

Genetische Algorithmen 3

- **Vererbung mit dem Crossover-Operator**
 - Kombiniere die Bitfolgen der Eltern
- **Verschiedenste Realisierungen**
- **1. Beispiel**
 - Wähle zufällige Crossover-Position $1 \leq r \leq L$
 - Kopiere bits $1 \dots (r-1)$ aus $f^{(k)}$ nach $\bar{f}^{(k+1)}$
 - Kopiere bits $r \dots L$ aus $g^{(k)}$ nach $\bar{f}^{(k+1)}$
 - ◆ Ggf.: Erzeuge 2. Kind $g^{(k+1)}$ mit vertauschten Rollen

Genetische Algorithmen 4

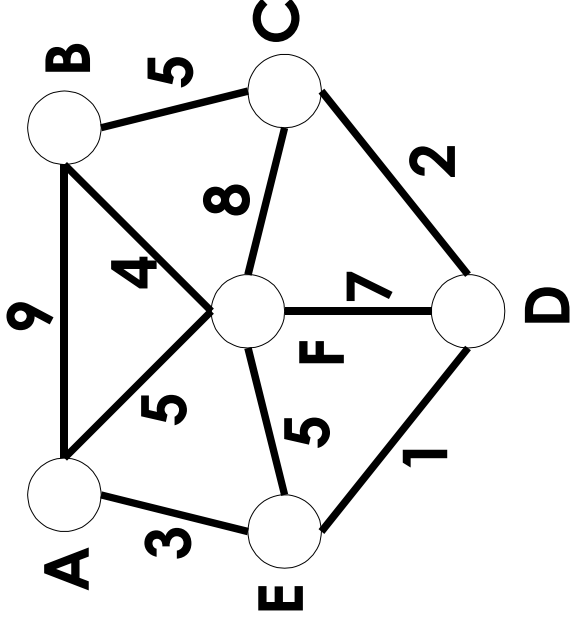
Beispiel: UPP im 10x10 Raster, platziere einzelne Zelle



Genetische Algorithmen 5

- **Crossover erzeugt ungütige Lösungen**
 - **Abhilfe: Mehr Struktur als einfache Bitfolgen**
- **Bei UPP: Folgen von 4-bit Koordinaten**
 - **Nun zwar intern konsistente Koordinaten**
 - **Reicht aber im allgemeinen nicht aus!**

Travelling Salesman Problem



- TSP
- *Einfacher* Zyklus durch alle Knoten mit minimaler Länge
 - Jeder Knoten nur einmal besucht
 - Minimale Kantengewichte
- NP-vollständig

Genetische Algorithmen 6

- Chromosom: Folge von Knoten
- Aber:
 - $\bar{f}^{(k)} = v1v3 \mid v6v5v2v4, \bar{g}^{(k)} = v4v2 \mid v1v5v3v6, r=3$
 - $\bar{f}^{(k+1)} = v1v3v1v5v3v6, \bar{g}^{(k+1)} = v4v2v6v5v2v4$
- Keine gültigen Lösungen!

Genetische Algorithmen 7

- Problem-spezifisches Crossover
- Bei TSP: z.B. Geordnetes Crossover
 - Kopiere Elemente 1...(r-1) aus $\bar{f}^{(k)}$ nach $\bar{f}^{(k+1)}$
 - Kopiere in $\bar{f}^{(k+1)}$ fehlende Elemente nach $\bar{f}^{(k+1)}$
 - ◆ In der Reihenfolge ihre Auftretens in $\bar{g}^{(k)}$
- Beispiel
 - ◆ $\bar{f}^{(k)} = v_1v_3 \mid v_6v_5v_2v_4, \bar{g}^{(k)} = v_4v_2 \mid v_1v_5v_3v_6, r=3$
 - ◆ $\bar{f}^{(k+1)} = v_1v_3v_4v_2v_5v_6, \bar{g}^{(k+1)} = v_4v_2v_1v_3v_6v_5, r=3$

Genetische Algorithmen 8

- Bisher noch keine Optimierung
 - Nur neue Lösungen erzeugt
- Bevorzuge gute Lösungen vor schlechten
 - Wähle „gute“ Eltern aus: Niedrige Kosten
 - Kombiniere gute Eigenschaften in Nachwuchs
 - Aber: Auch Gegenteil möglich (r zufällig)
 - ◆ Vererbung schlechter Eigenschaften
 - ◆ Idee: Schlechte Nachkommen vermeiden in nächster Generation

Genetische Algorithmen 9

```
genetic() {  
  int pop_size;  
  set<chromosome> pop, new_pop;  
  chromosome parent1, parent2, child;  
  
  pop := ∅;  
  for (i:=1; i <= pop.size(); i := i+1)  
    pop := pop ∪ {"Chromosom einer zufälligen Lösung"}  
  
  do {  
    newpop := ∅;  
    for (i:=1; i <= pop.size(); i := i + 1) {  
      parent1 := pop.select();  
      parent2 := pop.select();  
      child := crossover(parent1, parent2);  
      newpop := newpop ∪ {child};  
    }  
    pop := newpop;  
  } while (!stop());  
  report(pop.findmin(c));  
}
```

Genetische Algorithmen 10

- **stop()**
 - Keine Verbesserung in den letzten m Iterationen
 - m problemspezifischer Parameter
- **Mutation**
 - Fehler beim Kopieren
 - Vermeidet Steckenbleiben in lokalen Minima
- **Sehr viele Variationsmöglichkeiten**
 - Komplexes Crossover (mehrere r)
 - Mehrere Generationen gleichzeitig
 - Elite-Selektion
 - Meta-Genetische Algorithmen

Allgemeine Heuristiken

- **Diverse Alternativen**
 - Neuronale Netze
 - Simulierte Evolution
 - Lösen des SAT-Erfüllbarkeitsproblems
- **Bei allen allgemeinen Ansätzen**
 - Immer schlechter als problemspezifische
 - ◆ z.B. Kernighan-Lin für Partitionierung
 - Aber schneller zu realisieren
 - ◆ Bei unbekanntem Problemeigenschaften
- **Hybride Ansätze**
 - z.B. Eingeschränktes SA
 - ◆ SDI, TU Braunschweig

Vorbereitung

- **Bestimmung von T_a , T_r und slack**
 - Für Beispiel auf Folie 5
- **Im Buch lesen**
 - Kapitel 7.2 - 7.6

Zusammenfassung

- **Timing-Analyse**
- **Unit-Size Placement Problem**
- **Gierige Nachbarsuche**
 - **Steckenbleiben in lokalen Optima**
- **Untersuchen schlechterer Lösungen**
 - **Simulated Annealing**
 - **Tabu-Suche**
- **Genetische Algorithmen**
 - **Paralleles Untersuchen mehrerer Lösungen**
- **Hybride Verfahren**