Karlsruher Institut für Technologie

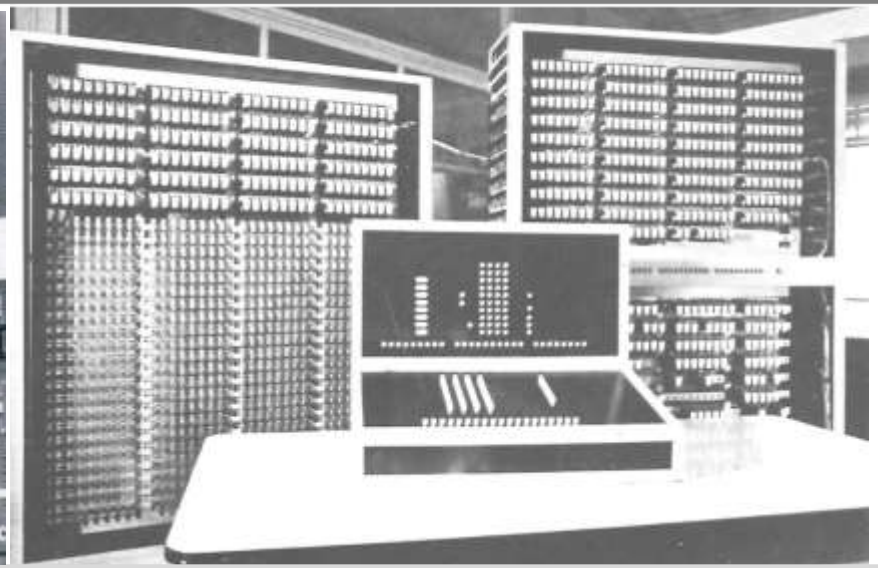
Institut für Technik der Informationsverarbeitung

Institutsleitung
Prof. Dr.-Ing. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

5. Optimierungsalgorithmen

Informationstechnik II

Prof. Dr.-Ing. Eric Sax

Inhalt IT2

4. Algorithmen auf Graphen

- Graphendefinition
- Binäre Suche
- Breitensuche
- Tiefensuche
- Zyklensuche
- Kürzester und kritischer Pfad

5. Optimierungsalgorithmen

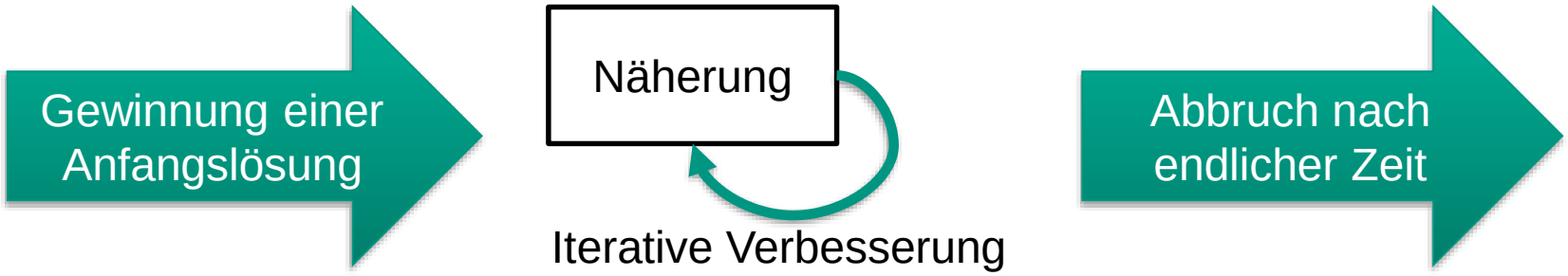
- Partitionierung
- Anlagerungsverfahren
- Random Interchange
- Kernighan-Lin
- Greedy
- Simulated Annealing

Was ist ein Optimierungsalgorithmus?

- Bei „komplexen“ Problemen ist die exakte Lösung oft nicht möglich oder sinnvoll.
  - Es gibt eine Vielzahl zulässiger Lösungen
  - Jeder Lösung ist ein Wert zugeordnet
  - Finden der Lösung mit dem „besten“ → dem „optimierten“ Wert
- Mögliche Gründe:
  - Formulierung als Algorithmus ist nicht möglich
  - Exakter Algorithmus hat Rechenzeit der Ordnung $O(n^x)$, $O(x^n)$ oder sogar $O(n!)$

Problemstellung

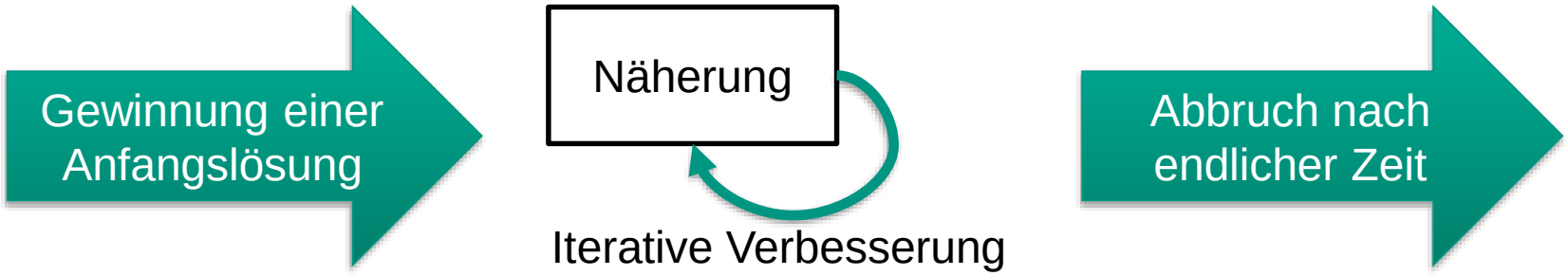
- Pragmatische Lösung: Heuristische Verfahren
  - Heuristik bezeichnet die Kunst, mit begrenztem Wissen und wenig Zeit zu guten Lösungen zu kommen.
  - Es bezeichnet ein analytisches Vorgehen, bei dem mit Hilfe von Mutmaßungen Schlussfolgerungen über das System getroffen werden.



Problemstellung: Optimierung

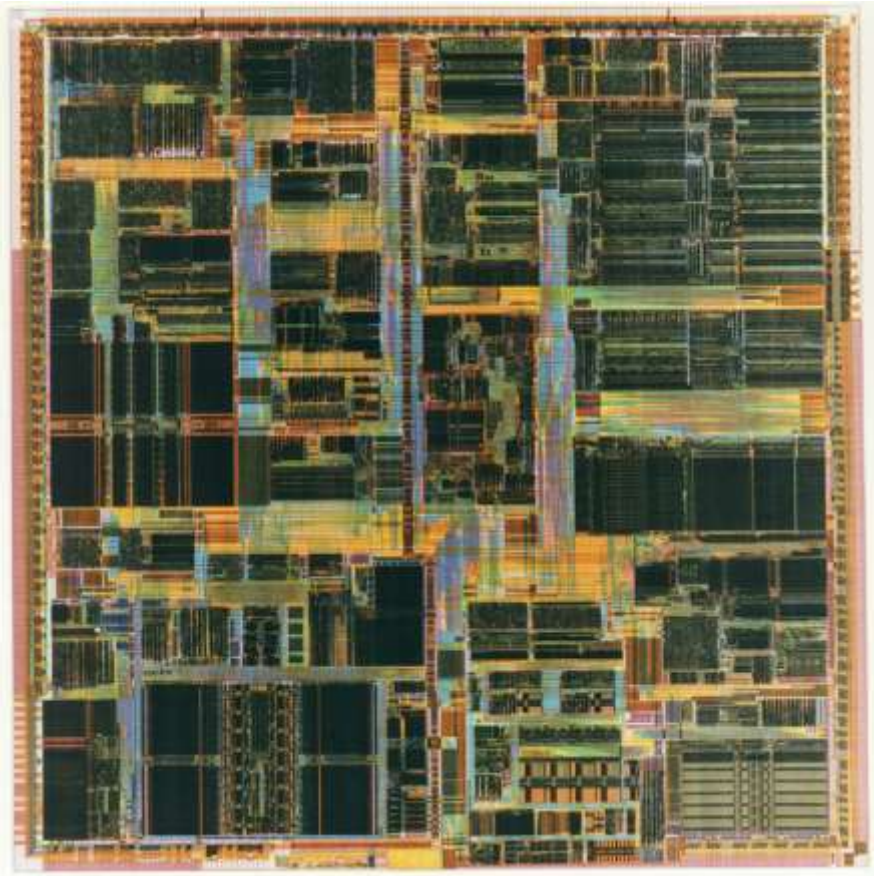
- Voraussetzung:
  - Bewertungsfunktion, Kostenfunktion
- Optimierungsziel (lokal):
  - Minimierung, Maximierung
- Ggf. auch nur ein Pareto-Optimum :

  Also ein Zustand, in dem es nicht möglich ist, eine Eigenschaft zu verbessern, ohne zugleich eine andere verschlechtern zu müssen.
- Pareto: 80-zu-20-Regel
  - 80 % der Ergebnisse mit 20 % des Gesamtaufwandes erreichen
  - Verbleibende 20 % der Ergebnisse benötigen mit 80 % die meiste Arbeit und werden vernachlässigt.



Anwendungsbeispiel – Chiplayout

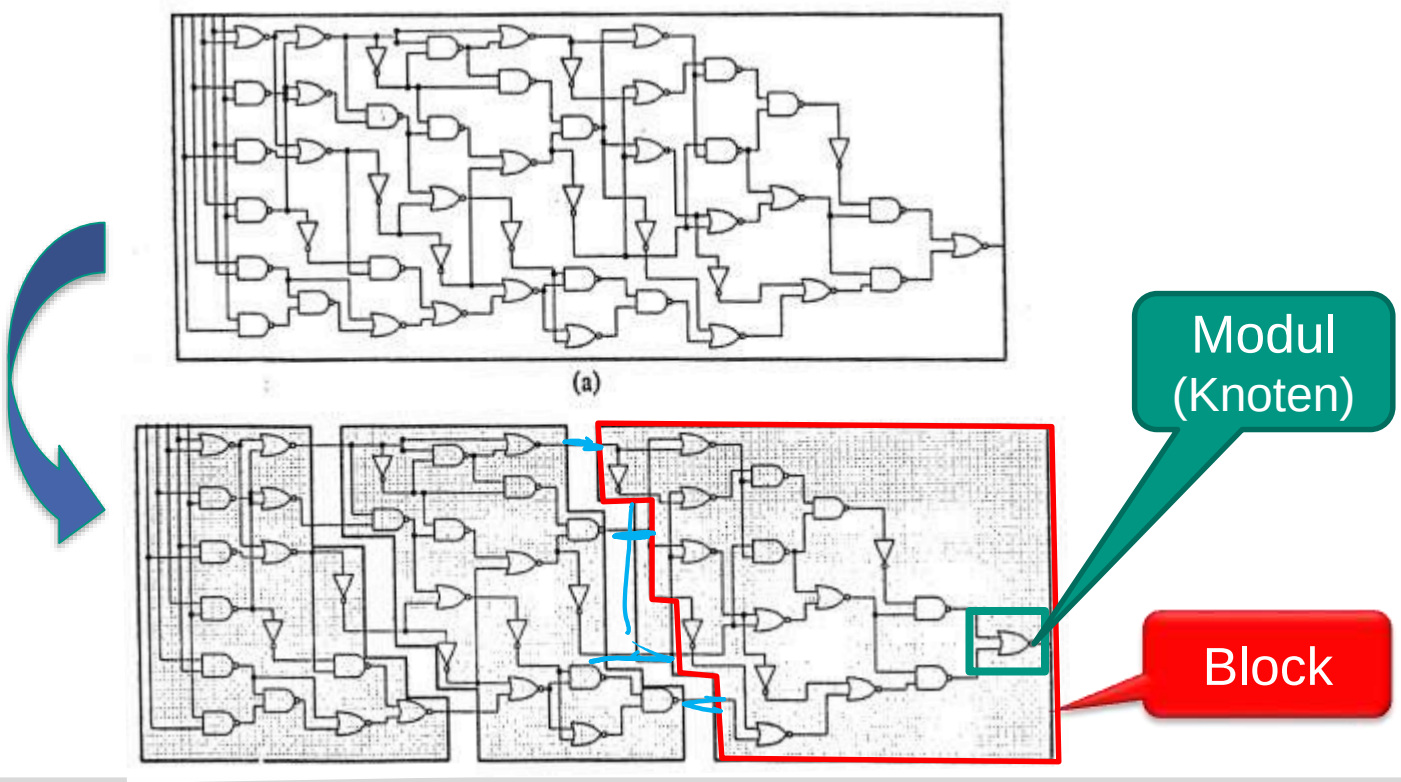
- Teilaufgaben beim Entwurf eines Chiplayout:
  - Partitionierung
  - Platzierung
  - Verdrahtung
- Viele Nebenbedingungen:
  - Geringe Gesamtfläche
  - Temperaturverhalten
  - Konnektivität
  - ...



P6 Dieshot

Partitionierung

- Chip-Entwicklung in kleinen Teilaufgaben (Wiederverwendung, Intellectual Property, Partnerentwicklungen, …)
  - ► Aufteilung der (geschätzten) Fläche eines Chips in einzelne funktionale Blöcke, die mit anderen Blöcken durch möglichst wenige kurze Leitungen verbunden sind:



Partitionierung – Aufgabenstellung

- Aufgabe: Finde eine Partition P der Modulmenge M = { m_1, ..., m_i, ...,m_n } aus Blöcken B mit den Unterscheidungskriterien:
  - Zahl der Verbindungen
  - Anzahl der Module
  - ...
- Vereinfachung:
  - Finde eine Bi-Partition P = { B_1, B_2 } mit minimaler Kostenfunktion
- Heuristische Verfahren:
  - Konstruktive Methoden:
    - Anlagerungsverfahren
  - Iteratives Verbessern:
    - Random Interchange
    - Kernighan-Lin
    - Greedy
    - Simulated Annealing
    - Evolutionäre Algorithmen (hier nicht)

Partitionierung – Überdeckung
(mathematische Definition)

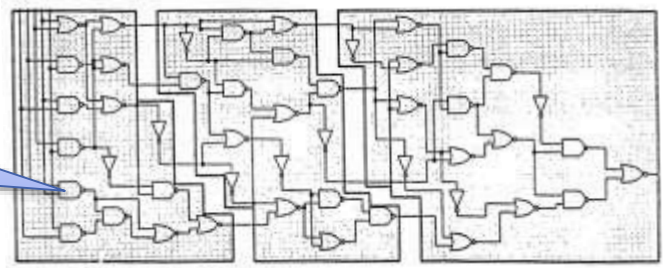
- Für „Partitionierung“ gilt:

  Modulmenge M = { m_1, ..., m_i, ..., m_n }

  Partition P = { B_1, B_2, B_3, …, B_k }

$$\bigcup_{i=1}^{k} B_i = M \quad \text{und} \quad \forall i, j \,|\, i \neq j \rangle \; B_i \bigcap B_j = \varnothing$$

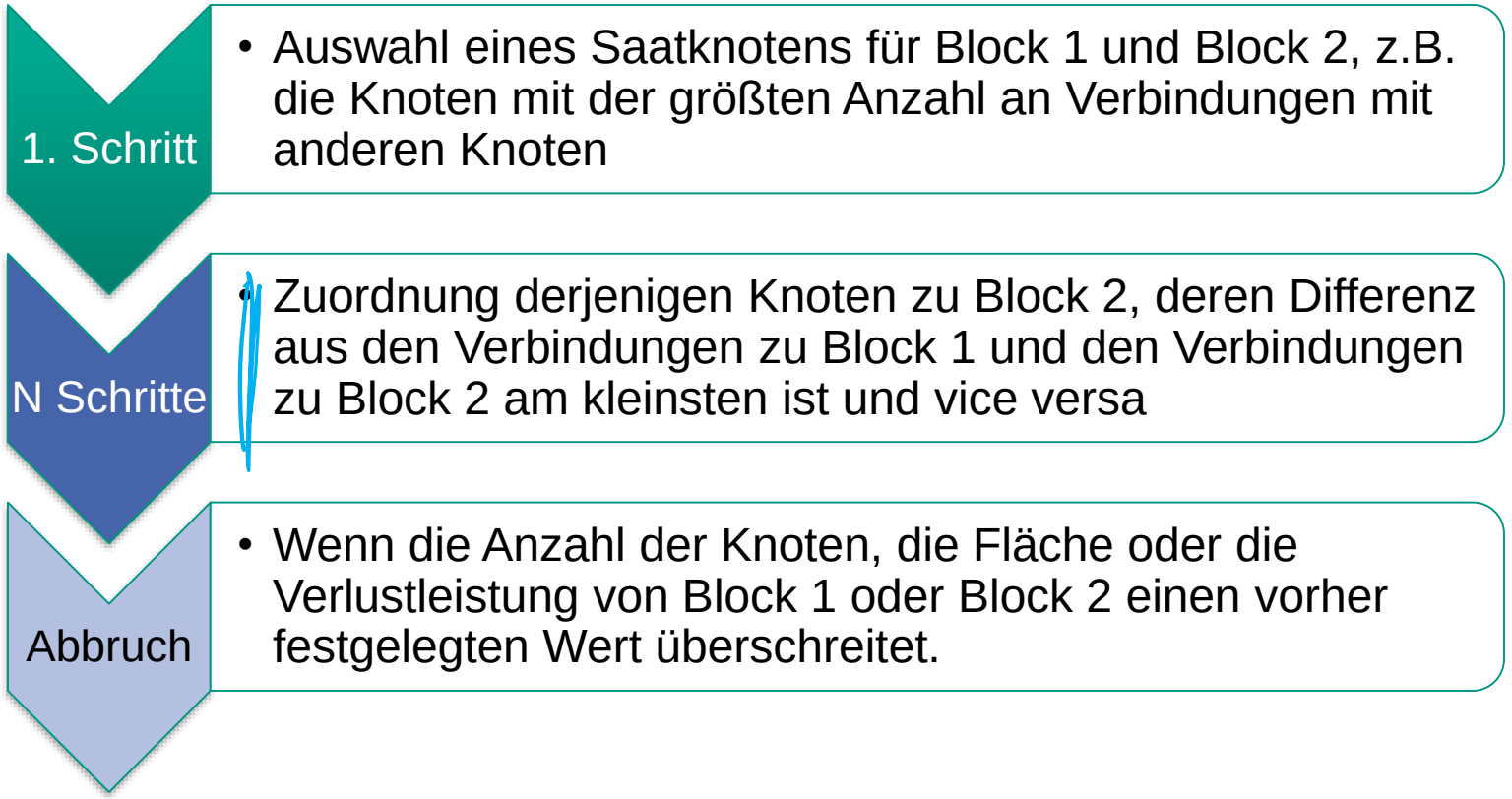
„Alle berücksichtigt“
„Keine Überlappungen“



- *Remark: Für „Überdeckung“ gilt lediglich*

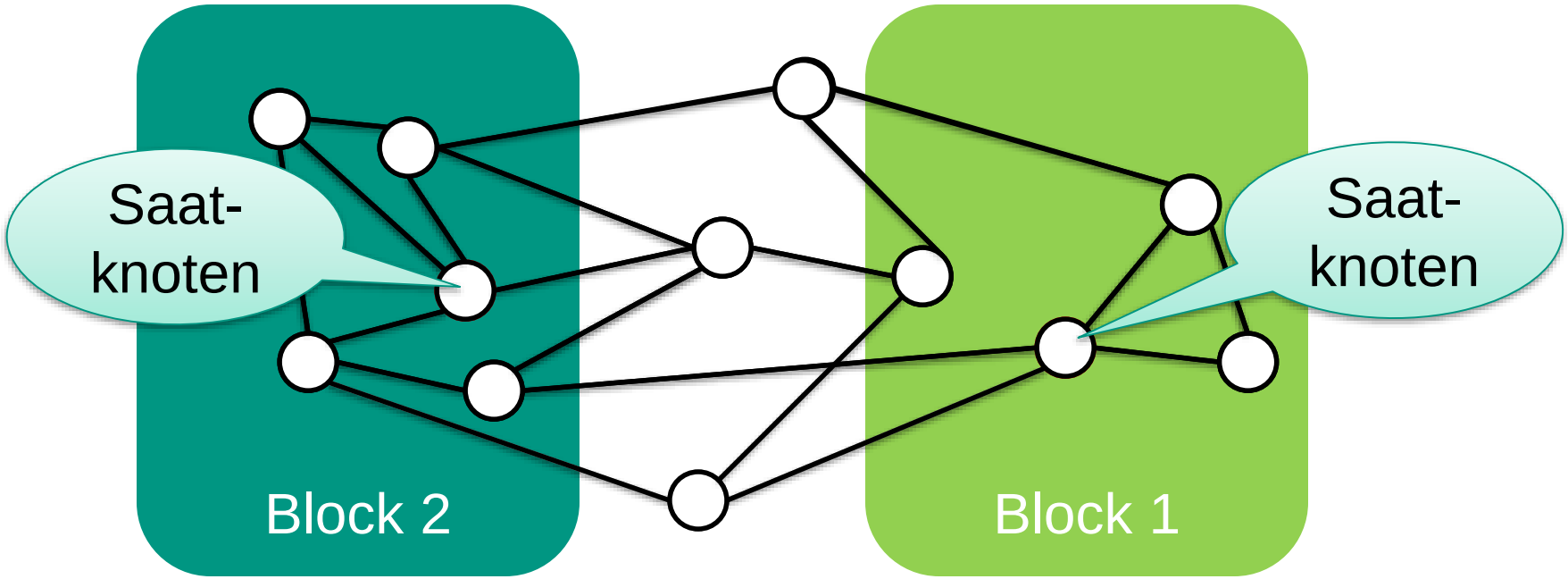
$$\bigcup_{i=1}^{k} B_i = M$$

Anlagerungsverfahren
(Bipartition)



- Problem: Kann in lokalem Minimum der Kostenfunktion enden

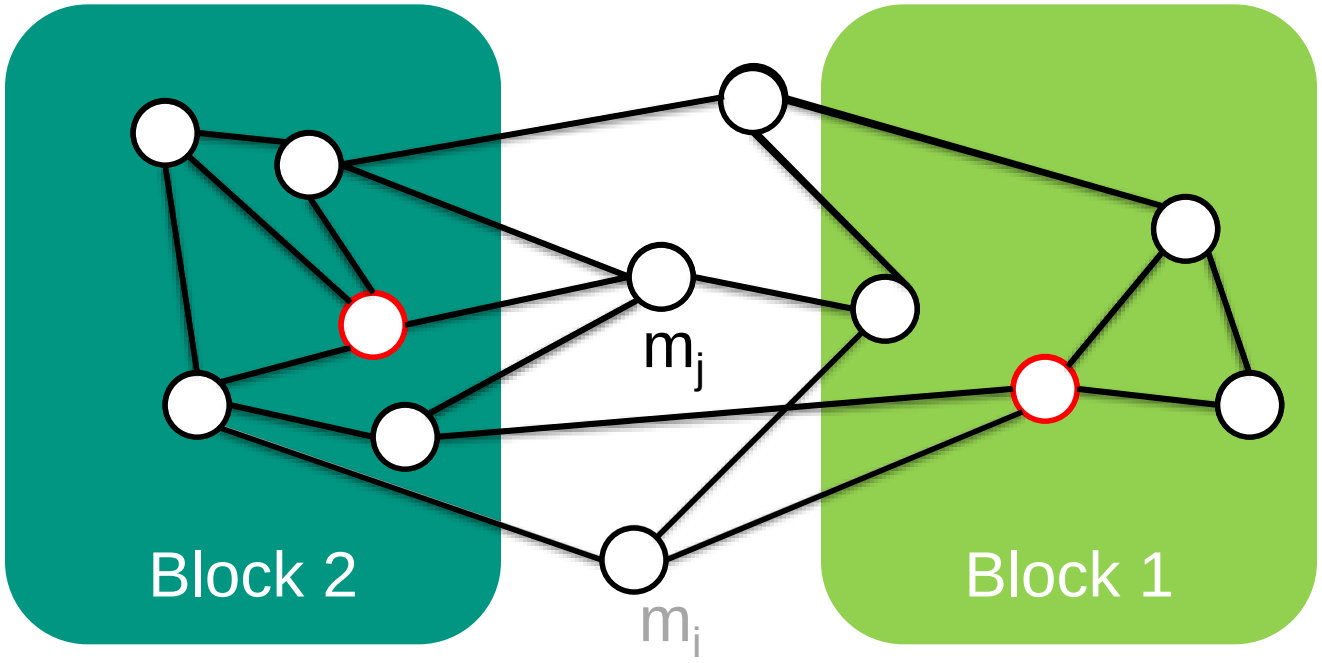
Beispiel: Anlagerungsverfahren



1. Schritt

- Auswahl eines Saatknotens für Block 1 und Block 2, z.B. die Knoten mit der größten Anzahl an Verbindungen mit anderen Knoten

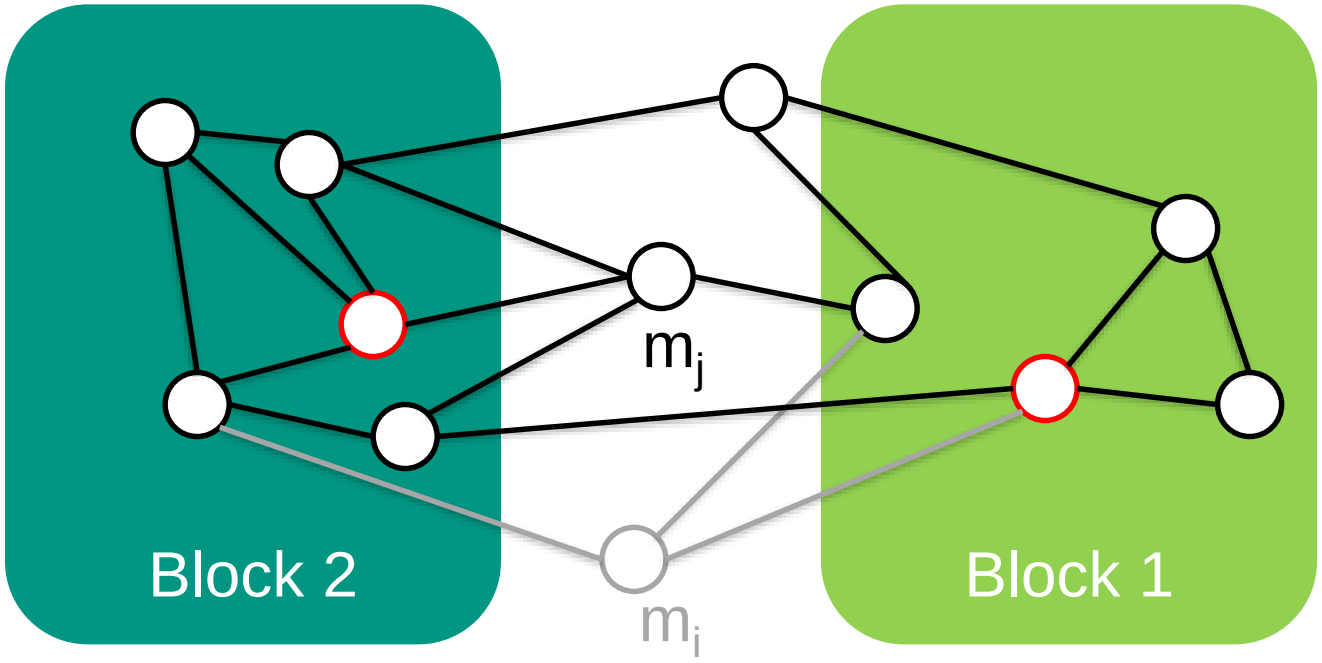
Beispiel: Anlagerungsverfahren



N Schritte

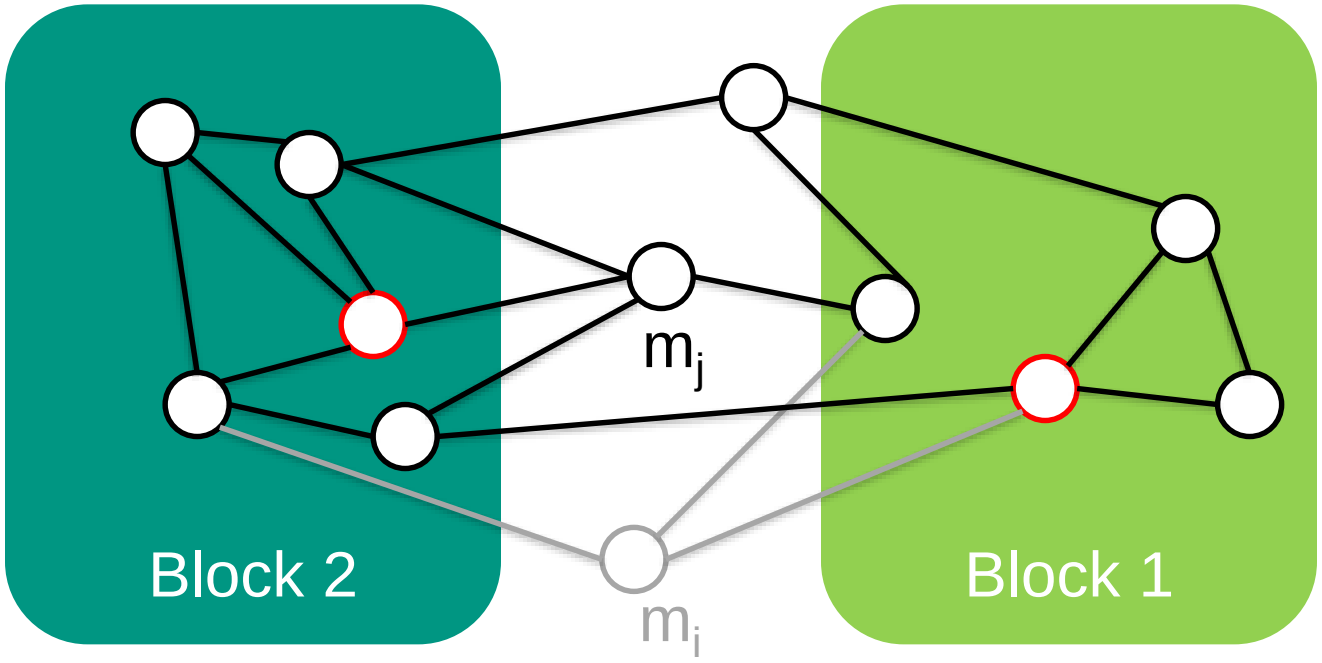
- Zuordnung derjenigen Knoten zu Block 2, deren Differenz aus den Verbindungen zu Block 1 und den Verbindungen zu Block 2 am kleinsten ist und vice versa

Beispiel: Anlagerungsverfahren



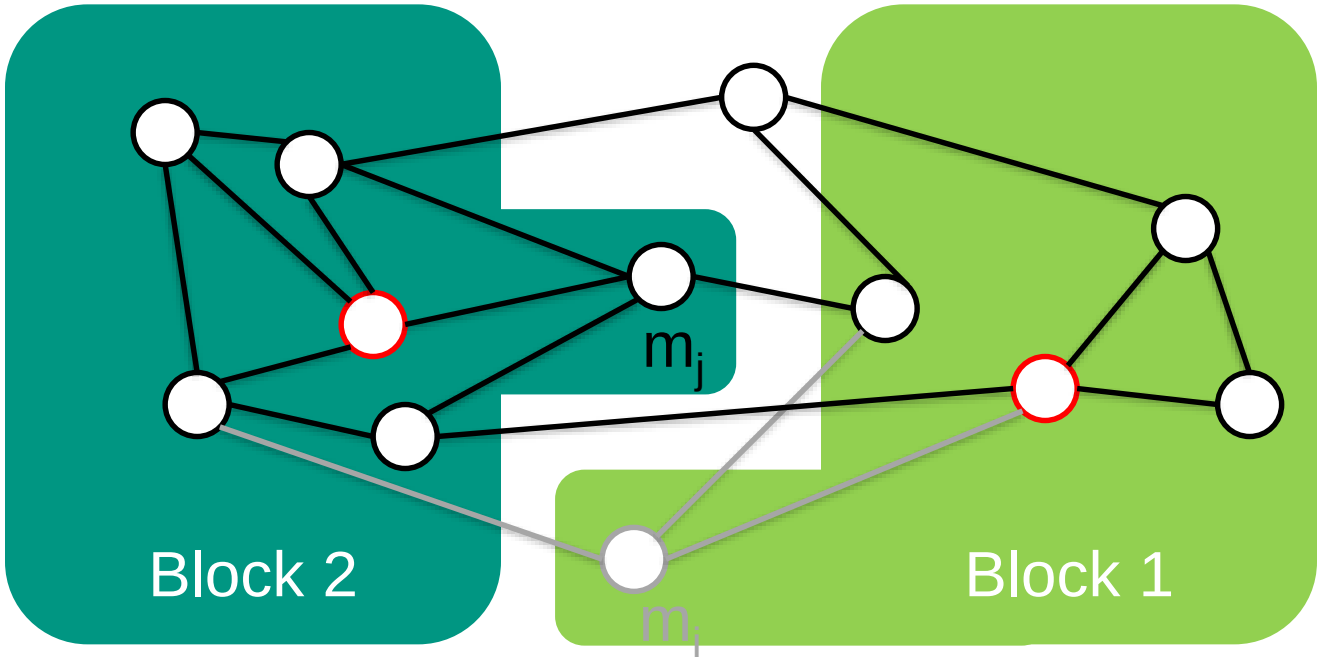
- Verbindungen zu Block 1: … m_j: 1
- Verbindungen zu Block 2: … m_j: 3
- Differenzen … -2
- Zuordnungen … ⇒ Block 2

Beispiel: Anlagerungsverfahren



- Verbindungen zu Block 1: … m_j: 1 m_i:2
- Verbindungen zu Block 2: … m_j: 3 m_i:1
- Differenzen … -2 1
- Zuordnungen … ➡ Block 2 ➡ Block 1

Beispiel: Anlagerungsverfahren



- Verbindungen zu Block 1: … m_j: 1 m_i:2
- Verbindungen zu Block 2: … m_j: 3 m_i:1
- Differenzen … -2 1
- Zuordnungen … ➡ Block 2 ➡ Block 1

- Anlagerungsverfahren

Random Interchange

- Iteratives Verbessern einer bestehenden Bipartition durch zufälliges Vertauschen von Knoten:

1. – N. Schritt

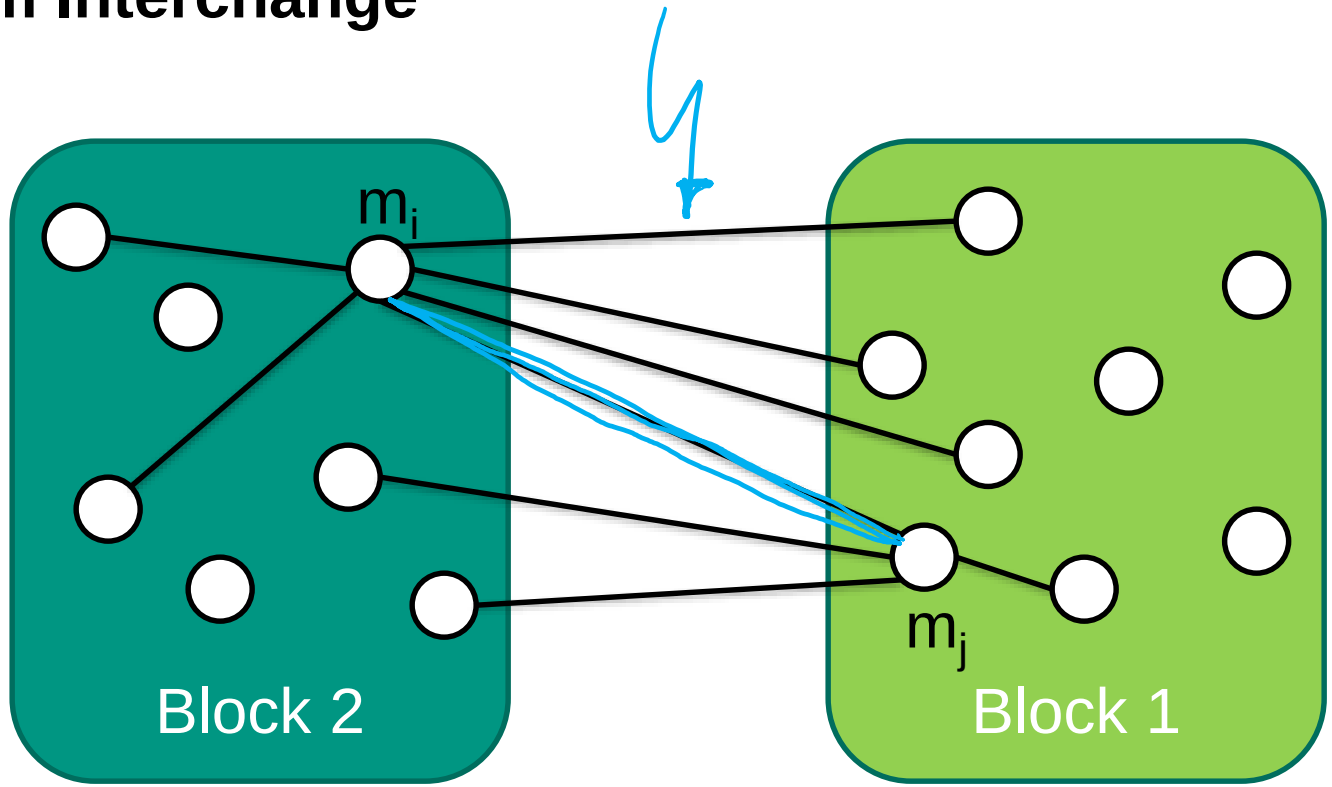
- Zufällige Auswahl zweier Knoten
- Berechnung der Kosten vor und nach dem Austausch
- Beibehalten des Austausches bei Kostenminderung

Abbruch

- Wenn N Austauschversuche unternommen wurden, folgt der Abbruch

- Kosten entstehen durch interne und externe Kanten

Beispiel: Random Interchange



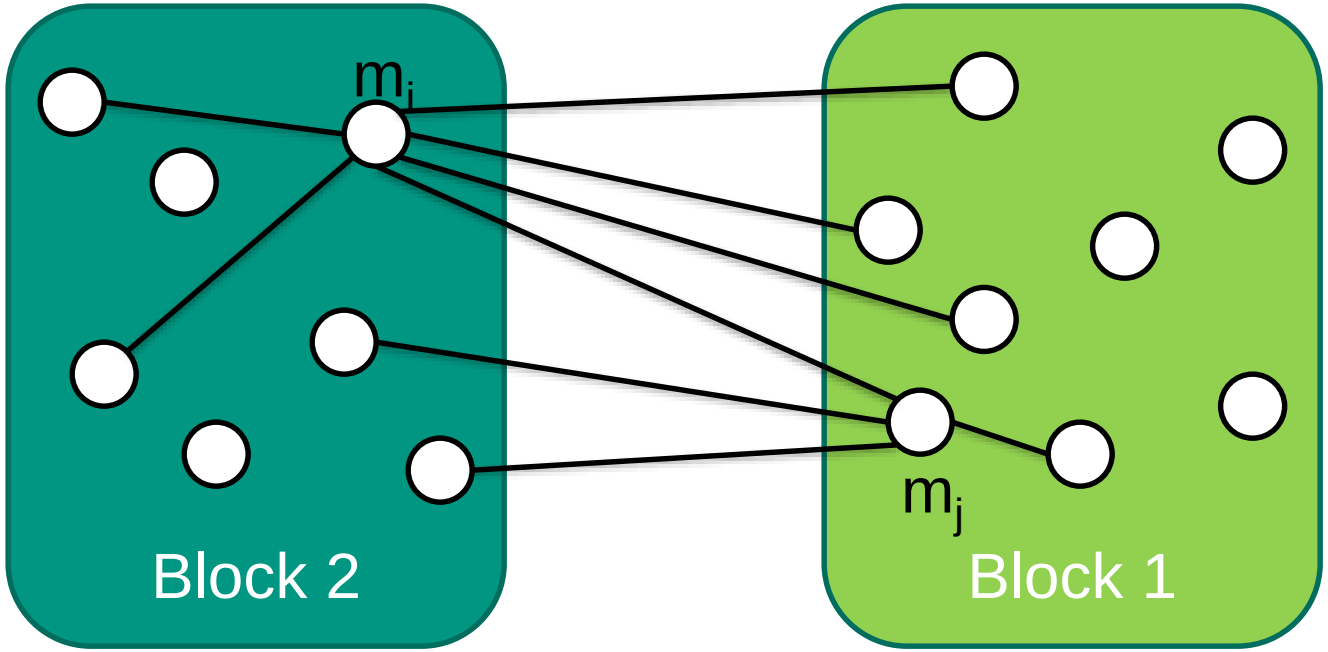
- Ziel ist, dass die Summe der Kantengewichte zwischen Block 1 und Block 2 minimal wird.
- Remark: *Hier sind alle Kantengewichte 1*

Kostenfunktion

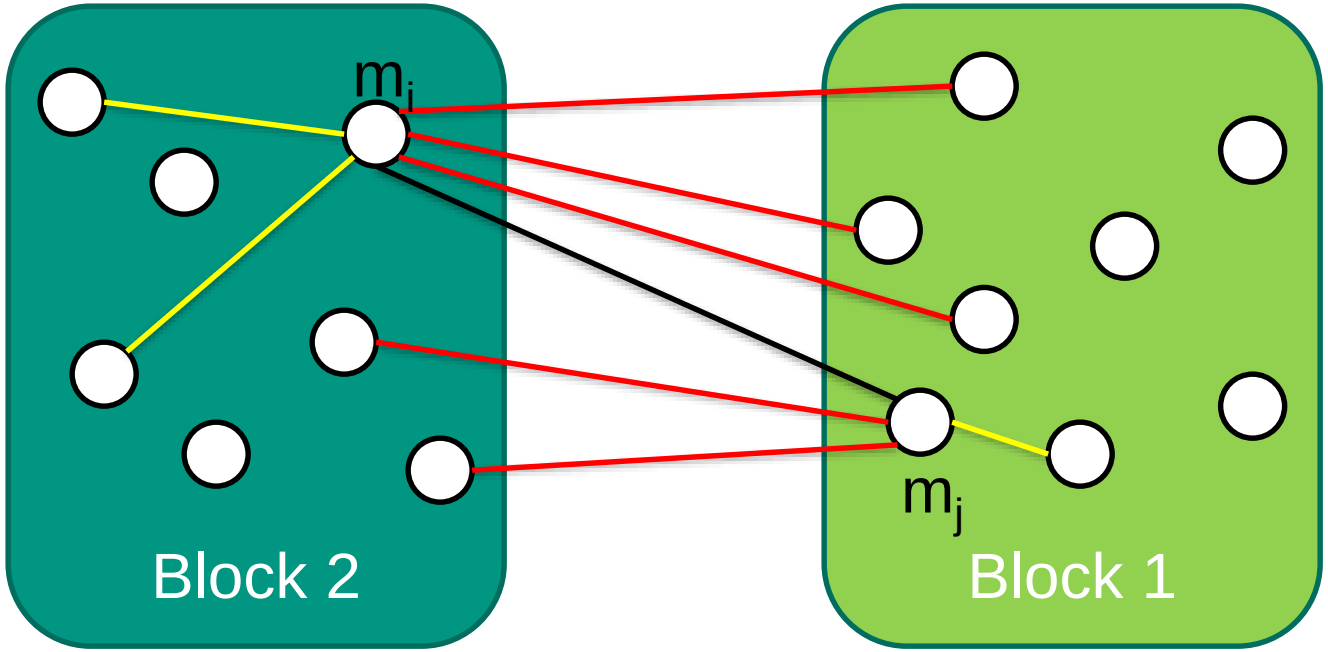
- Die internen Kosten von m_i, also die Summe aller Kantengewichte abgehend vom Knoten m_i in Block 2, wird als $I(m_i)$ bezeichnet.
- $E(m_i)$ seien die externen Kosten vom Knoten m_i, also die Summe aller Kosten der Kanten zwischen m_i und den Knoten in Block 1.
  - $D(m_i) = E(m_i) – I(m_i)$
    ist die Differenz der externen und internen Kantengewichte.
- Werden zwei Knoten m_i und m_j betrachtet, so ist die Kostenfunktion
  - $D = D(m_i) + D(m_j) - 2\, c_{ij}$
    $= E(m_i) + E(m_j) – I(m_i) –I(m_j) - 2\, c_{ij}$

    hierbei sind $c_{i,j}$ die Kosten der Kante zwischen mi und mj.
- Bei positiver Kostenfunktion wird getauscht.
- Remark zu c_{ij}: *Die direkte Verbindung der beiden Knoten würde zweimal gezählt und verbindet auch weiterhin die beiden Knoten.*

Beispiel: Random Interchange

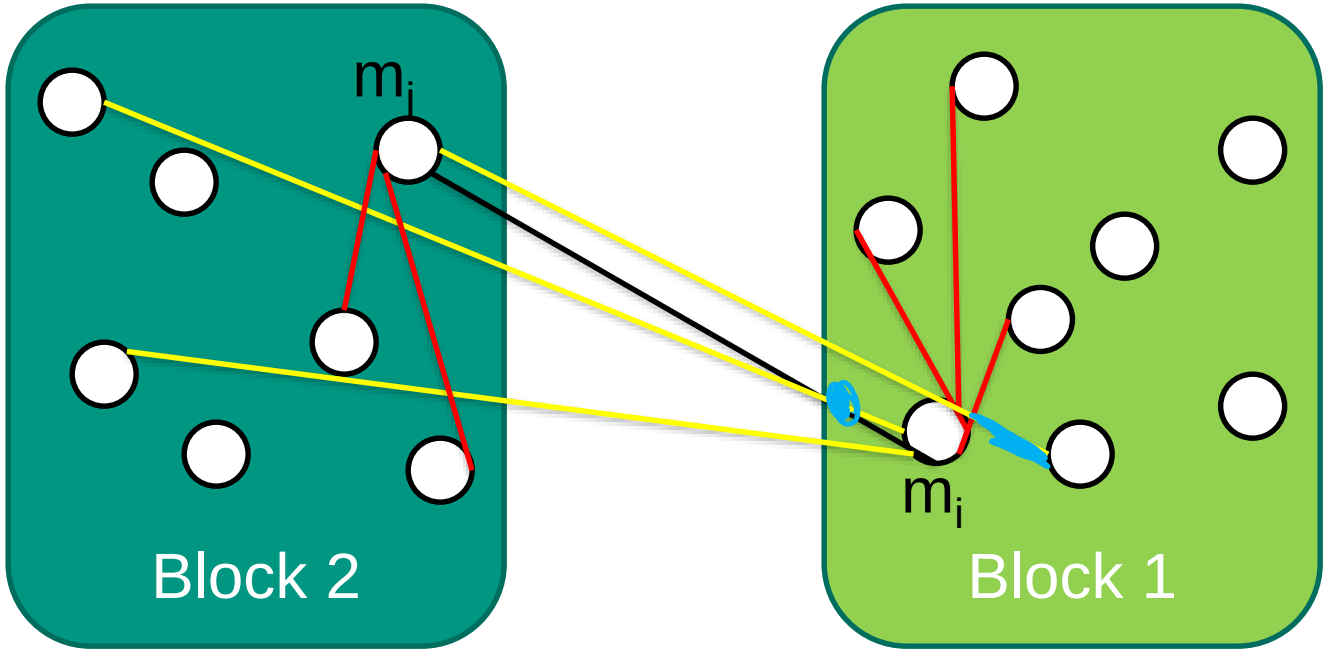


Beispiel: Random Interchange



- Knoten m_i: innere: 2 externe: 4 c_{ij}: 1
- Knoten m_j: innere: 1 externe: 3 c_{ij}: 1
- Bewertungsfunktion: $D = E(m_i) + E(m_j) - I(m_i) - I(m_j) - 2\,c_{ij} = 4+3-2-1-1-1$
- $D = 2 > 0$:
  - Verbesserung → also Austausch von m_i und m_j

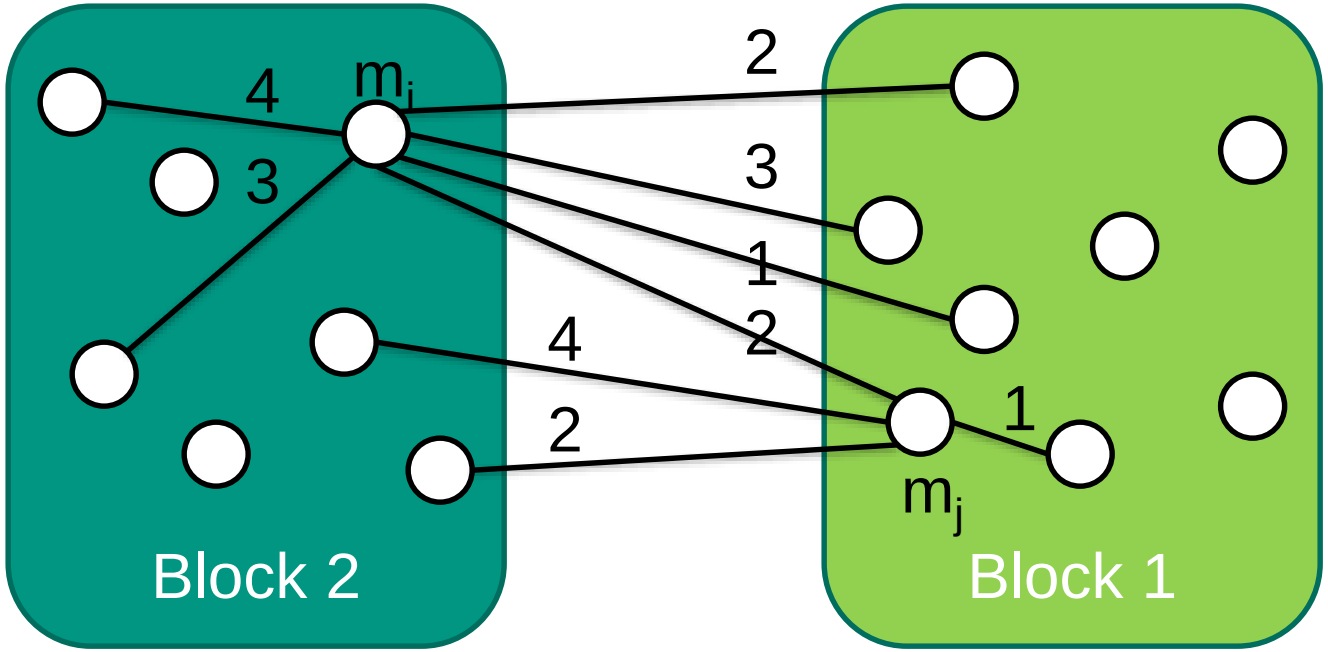
Beispiel: Random Interchange



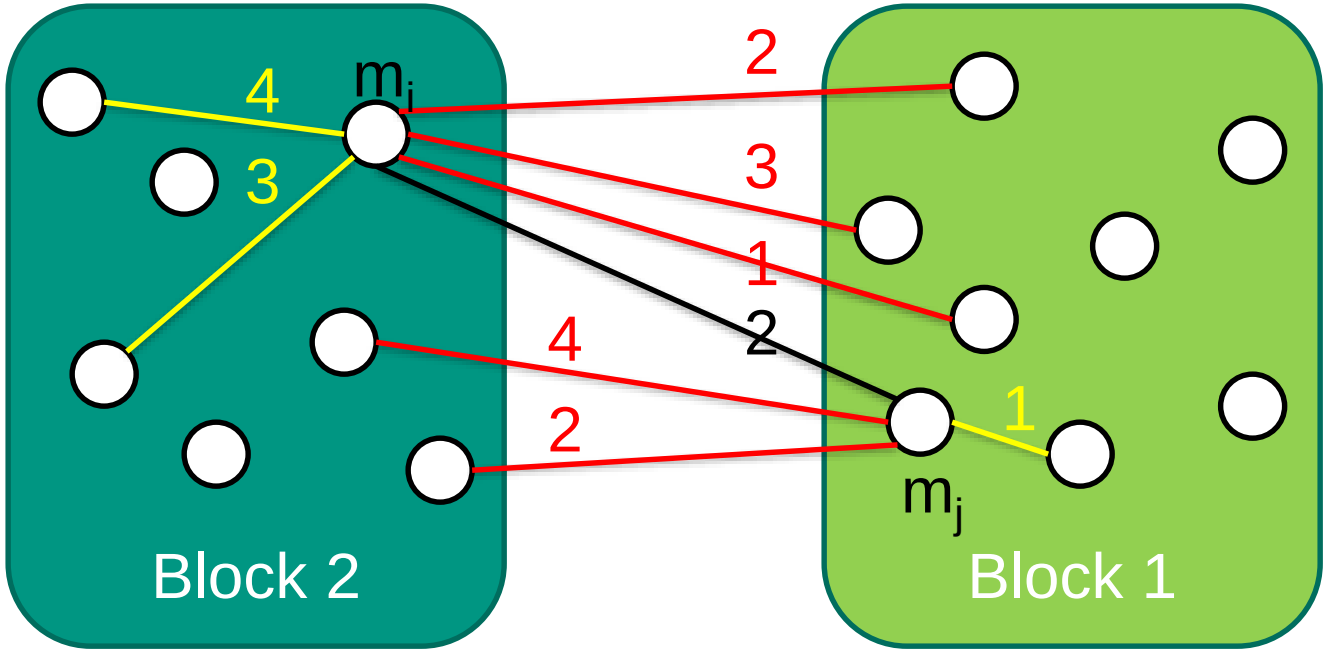
- Knoten m_i: innere: 2 externe: 4 c_{ij}: 1
- Knoten m_j: innere: 1 externe: 3 c_{ij}: 1
- Bewertungsfunktion: $D = E(m_i) + E(m_j) - I(m_i) - I(m_j) - 2\,c_{ij} = 4+3-2-1-1-1$
- $D = 2 > 0$:
  - Verbesserung → also Austausch von m_i und m_j

Beispiel: Random Interchange

Mit Kantengewichten



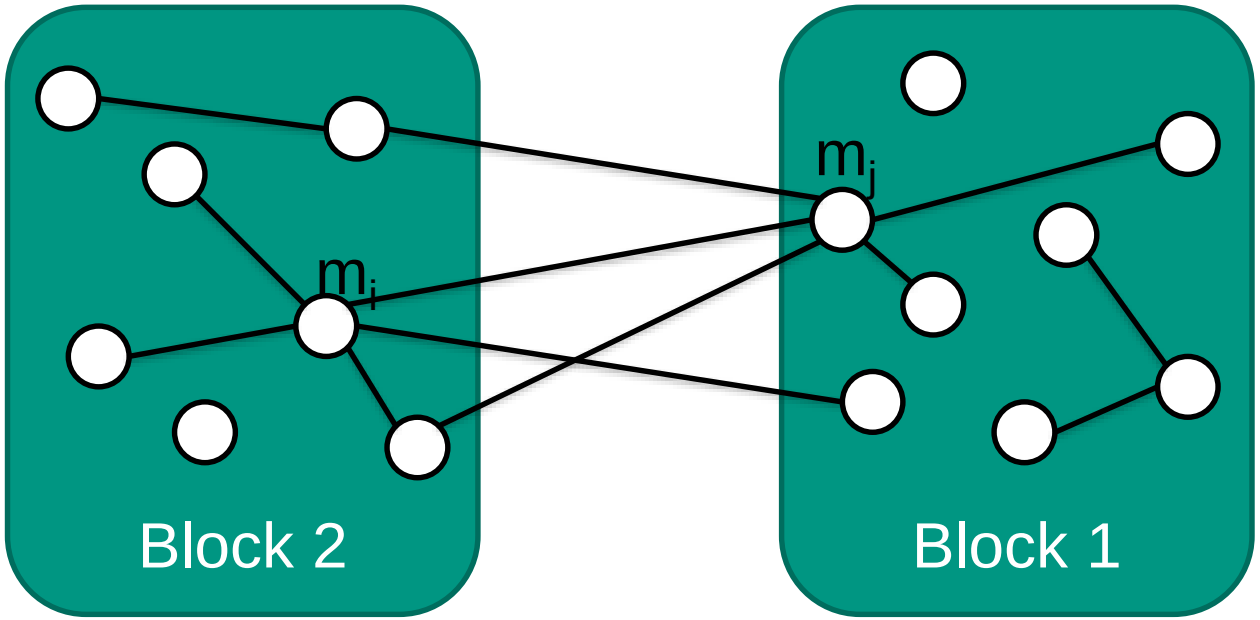
Beispiel: Random Interchange



- Knoten mi: innere: 4+3 externe: 2+3+1+2 cij: 2
- Knoten mj: innere: 1 externe: 2+4+2 cij: 2
- Bewertungsfunktion: D = E(mi) + E(mj) - I(mi) - I(mj) - 2 cij = 8+8-7-1-2-2
- D = 4 > 0 :
  - Verbesserung → also Austausch von mi und mj

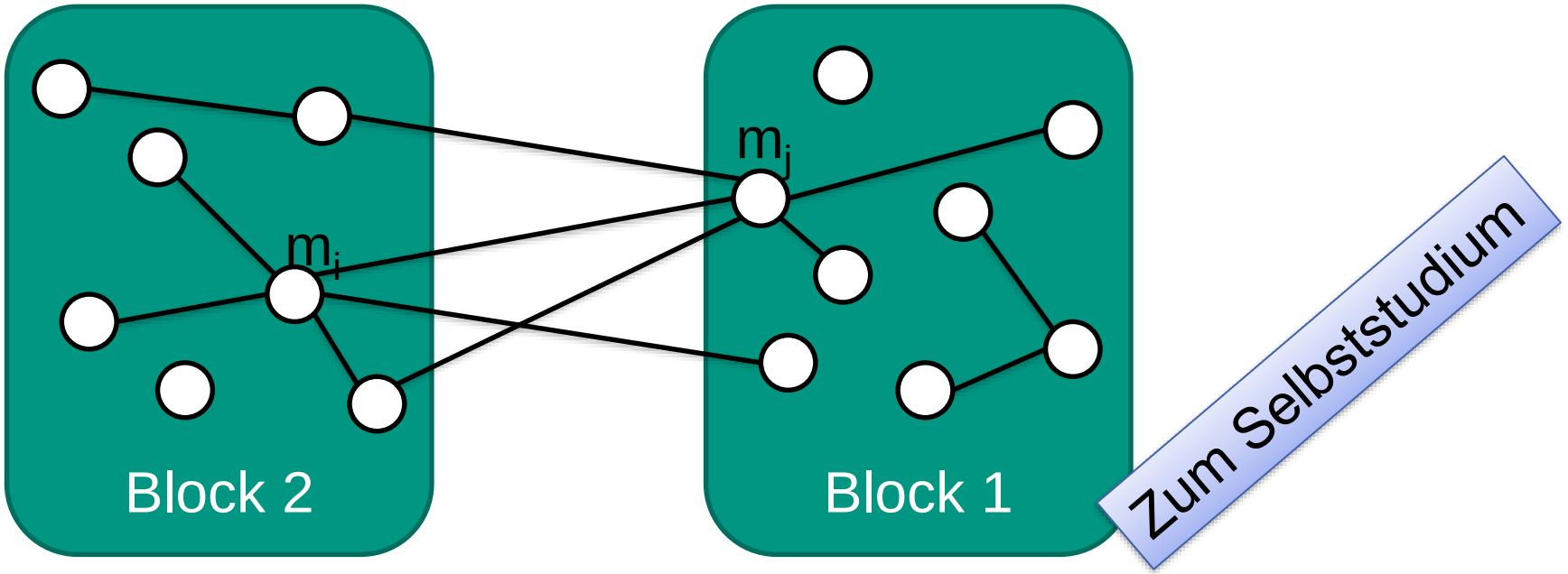
Zwischenübung 01: Random Interchange

- Bringt die Vertauschung von Knoten mi und mj einen Kostenvorteil?



Zwischenübung 01: Random Interchange – Lsg.

- Bringt die Vertauschung von Knoten mi und mj einen Kostenvorteil?



- Knoten mi: innere: 3 externe: 2 cij: 1
- Knoten mj: innere: 2 externe: 3 cij: 1
- Bewertungsfunktion: D = E(mi) + E(mj) - I(mi) - I(mj) – 2 cij = -2
- D = -2 < 0
  - Verschlechterung → kein Austausch von mi und mj

- Random Interchange

Greedy

- Grundprinzip:
  - Finden einer Lösung, indem bei jedem Entscheidungsschritt die lokal optimale Entscheidung getroffen wird.
  - Dadurch findet man für viele Probleme aber nicht zwingend die global optimale Lösung
  - Trotzdem oft verwendet, da bei günstigem Sortierkriterium meist eine hinreichend gute Lösung mit sehr geringem Aufwand gefunden wird
- Eine Anwendung des Greedy-Algorithmus im täglichen Leben ist z.B. die Herausgabe von Wechselgeld.
  - Greedy: *Nimm jeweils immer die größte Münze unter dem Zielwert und ziehe sie von diesem ab.*
  - Verfahre derart bis Zielwert gleich null.
- Beispiel:
  - Zielwert ist 15.
  - Es stehen Münzen mit den Werten 1, 5 und 11 zu Verfügung.
  - 15 = 5 + 5 + 5 ist globales Optimum
  - 15 = 11 + 1 + 1 + 1 + 1 mit Greedy kein globales Optimum

Rucksackproblem

Kerzenständer
Gewicht: 5kg
Nutzen: 20

Radio
Gewicht: 2kg
Nutzen: 60

Taschenmesser
Gewicht: 0,3kg
Nutzen: 80

Wanderkarte
Gewicht: 0,2 kg
Nutzen: 100

Regenjacke:
Gewicht: 3kg
Nutzen: 90

Rucksack
Maximalgewicht: 10kg

Blumentopf
Gewicht: 4kg
Nutzen: 10

Rucksackproblem

- Eingabe:
  - Ein leerer Rucksack mit einer maximalen Kapazität (Maximalgewicht)
  - Eine Auswahl an möglichen Gegenständen, wobei jeder Gegenstand ein Gewicht und einen Nutzen hat.
- Zu lösendes Problem: packe den Rucksack so, …
  - … dass das Gesamtgewicht der eingepackten Gegenstände die Kapazität nicht übersteigt und/oder …
  - … der Nutzen der eingepackten Gegenstände optimal, d.h. maximal, ist.
- Klassisches Optimierungsproblem, auf welches zahlreiche andere Auswahlprobleme abgebildet werden können.
- Komplexitätsklasse $O(2^n)$ (n ist Anzahl der wählbaren Gegenstände)

Rucksackproblem: Lösung mit Greedy-Algorithmus

- Idee:
  - man packe den Rucksack, indem man jeweils den aktuell besten Gegenstand auswählt, z.B. nach folgenden Kriterien:
    - A) den mit dem höchsten Nutzen,
    - B) den mit dem geringsten Gewicht oder
    - C) den mit dem besten Verhältnis von Nutzen und Gewicht.
  - Wiederhole diese Auswahl, bis der Rucksack voll ist bzw. nur noch Gegenstände übrig sind, die nicht mehr in den Rucksack passen

- Greedy

Kernighan-Lin-Algorithmus

- Systematisches Vertauschen aller Knoten der beiden Partitionsblöcke und Erstellen einer Kostenfunktion:

1. – N. Schritt

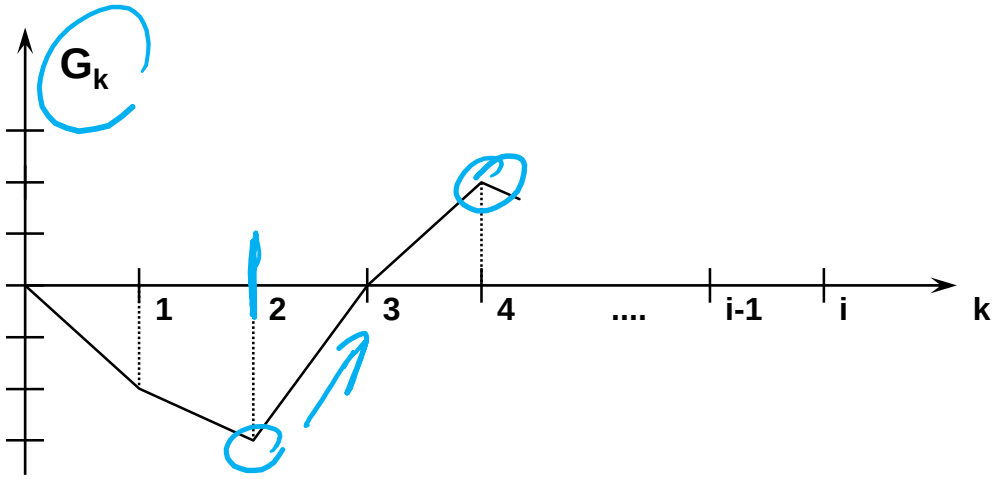
- Auswahl des Knotenpaars mit der besten Kostenfunktion, welches noch nicht getauscht wurde
- Vertauschen der beiden Knoten, trotz eventueller Verschlechterung der Gesamtkosten

Abbruch

- Wenn alle Knoten getauscht wurden, die Knoten also alle den Block gewechselt haben
- Auswahl der kostengünstigsten Konfiguration

Kernighan-Lin-Algorithmus

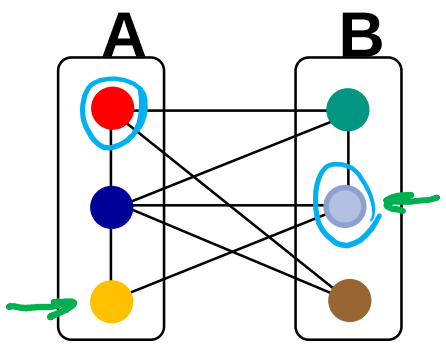
- Gleiche Bewertungsfunktion wie bei Random Interchange:
  - $D(m_i, m_j) = E(m_i) + E(m_j) - I(m_i) - I(m_j) - 2\, c_{ij}$
- Verlauf der Bewertungsfunktion über der Anzahl der Vertauschungen k, mit D_i = Gewinn beim i-ten Austausch
  - Hohe Kosten (positiver *D*-Wert) geben damit einen bestimmten Drang an, die Teilmenge zu wechseln, während niedrige Kosten (negativer *D*-Wert) ein gewisses „Bleiberecht“ verkörpern.
- Der ausgewählte Satz von Vertauschungen korrespondiert mit dem maximalen Gewinn [max G_k]
- Beispiel:



$$G_k = \sum_{i=1}^{k} D_i$$

Beispiel: Kernighan-Lin

1.



2.

A \ B	1	2	3
1	0	2	1
2	0	0	1
3	1	-1	2

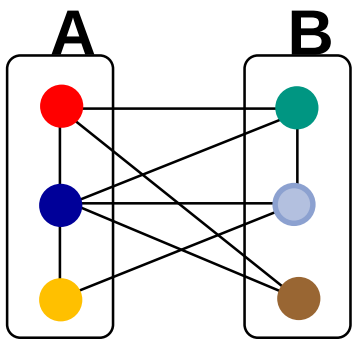
- $D(m_i, m_j) = E(m_i) + E(m_j) - I(m_i) - I(m_j) - 2\,c_{ij}$

$= 2 + 2 - 1 - 1 - 0 = 2$

$= 1 + 2 - 1 - 1 - 2 = -1$

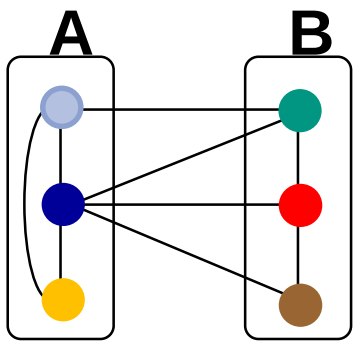
Beispiel: Kernighan-Lin

1.



A \ B	1	2	3
1	0	**2**	1
2	0	0	1
3	1	-1	2

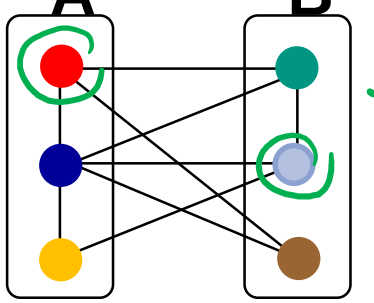
2.



A \ B	1	2	3
1	-	-	-
2	**0**	-	-1
3	-1	-	-2

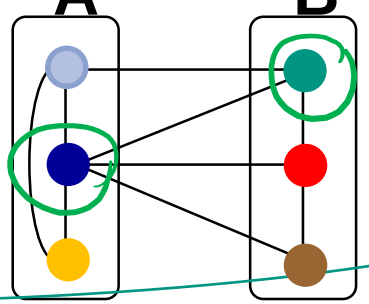
Beispiel: Kernighan-Lin

1.



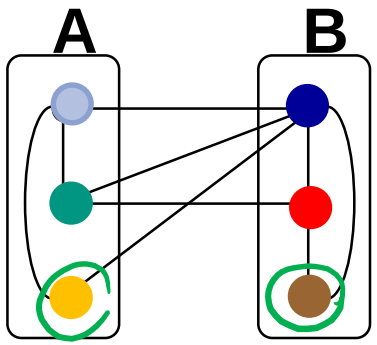
A \ B	1	2	3
1	0	2	1
2	0	0	1
3	1	-1	2

2.



A \ B	1	2	3
1	-	-	-
2	0	-	-1
3	-1	-	-2

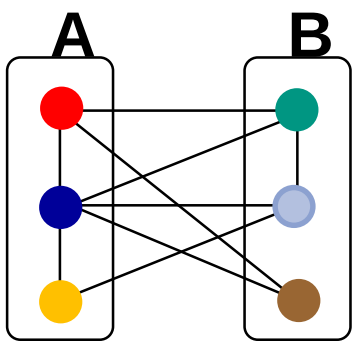
3.



A \ B	1	2	3
1	-	-	-
2	-	-	-
3	-	-	-2

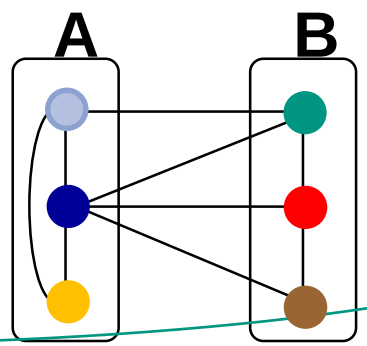
Beispiel: Kernighan-Lin

1.



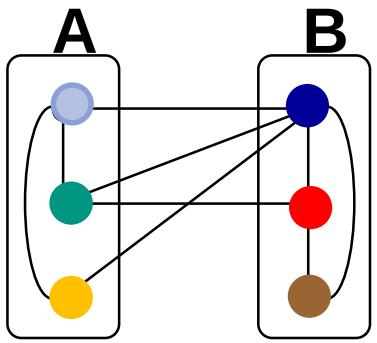
A \ B	1	2	3
1	0	**2**	1
2	0	0	1
3	1	-1	2

2.



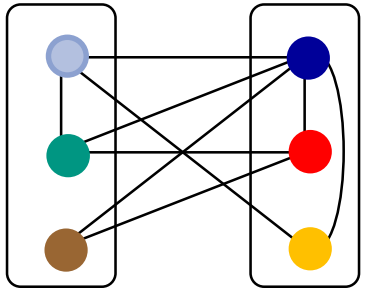
A \ B	1	2	3
1	-	-	-
2	**0**	-	-1
3	-1	-	-2

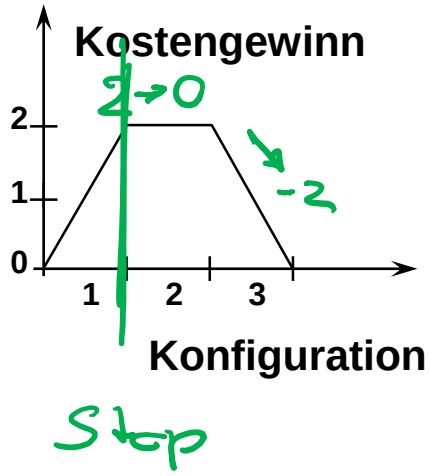
3.



A \ B	1	2	3
1	-	-	-
2	-	-	-
3	-	-	**-2**

4.





- Kernighan-Lin-Algorithmus

Simulated Annealing (*simulierte Abkühlung*)

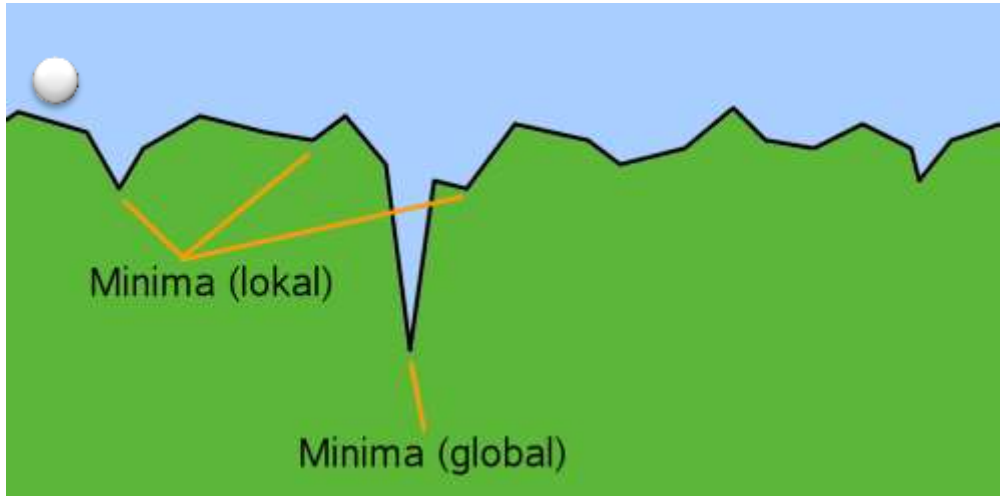
- Das Verfahren wird zum Auffinden einer approximativen Lösung von Optimierungsproblemen eingesetzt, die durch ihre hohe Komplexität das vollständige Ausprobieren aller Möglichkeiten und einfache mathematische Verfahren ausschließen.
- Benutzt wird dieses Verfahren zum Beispiel beim *Floorplanning* im Laufe eines Chipentwurfs.
- Grundidee ist die Nachbildung eines Abkühlungsprozesses, etwa beim Glühen in der Werkstoffkunde.
  - Nach Erhitzen eines Metalls sorgt die langsame Abkühlung dafür, dass die Atome ausreichend Zeit haben, sich zu ordnen und stabile Kristalle zu bilden.
- Übertragen auf das Optimierungsverfahren entspricht die Temperatur einer Wahrscheinlichkeit, mit der sich ein Zwischenergebnis der Optimierung auch verschlechtern darf.
  - Bzw. mit abnehmender Temperatur verringert sich der Lösungsraum.

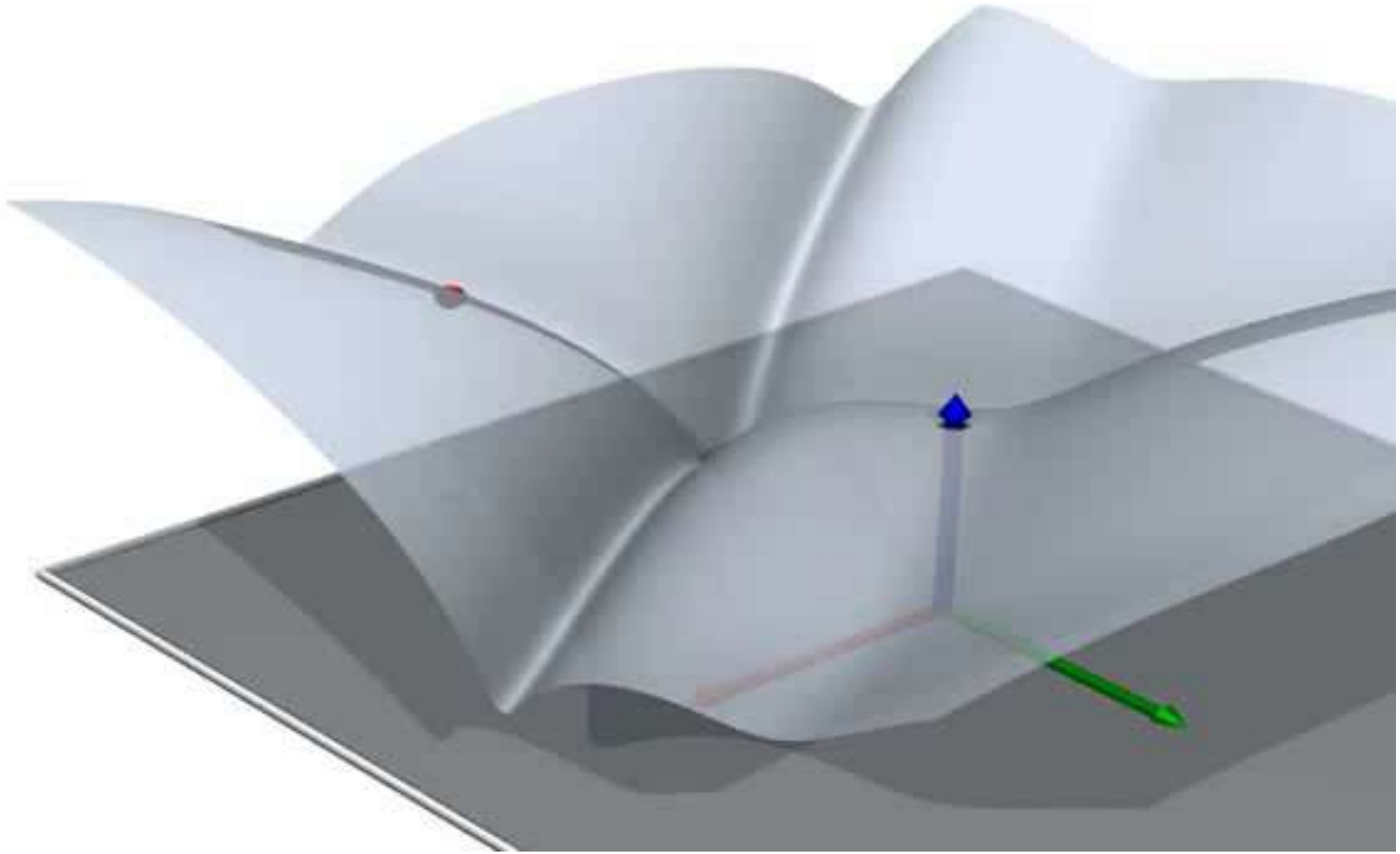
Analogie

Das simulierte Ausglühen kann man sich wie folgt verdeutlichen.

- Man sucht in einer Landschaft den (global) tiefsten Punkt.
- Die Landschaft selbst besteht aus vielen unterschiedlich tiefen Dellen.
- Die einfache Suchstrategie entspricht dem Verhalten einer Kugel, welche in dieser Landschaft ausgesetzt wird.
  - Sie rollt zum nächsten lokalen Minimum und bleibt dort.
- Bei der simulierten Abkühlung wird der Kugel immer wieder ein Stoß versetzt, der mit zunehmender „Abkühlung“ schwächer wird.

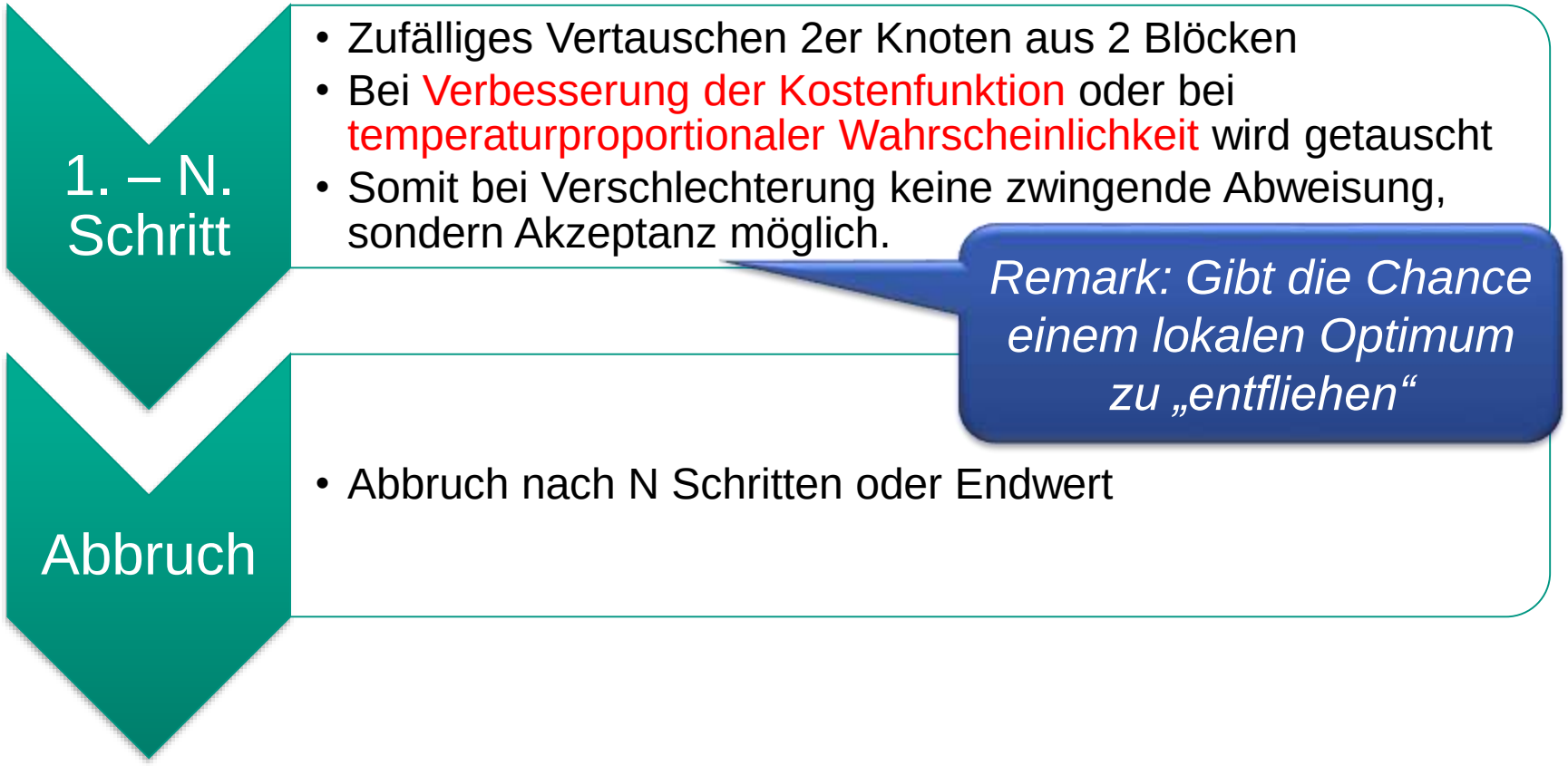
- Dieser ist idealerweise stark genug, um die Kugel aus einer flachen Delle (lokales Minimum) zu entfernen, reicht aber nicht aus, um aus dem globalen Minimum zu fliehen.





Simulated Annealing (*simulierte Abkühlung*)

- Verbesserung des Random Interchange-Algorithmus



Ende, VL 22.05.2020

Pseudo Code: Simulated Annealing

```
simulate_annealing() {
  T = T0;                                          //Anfangswert der Temperatur
  X = j0;                                          //Anfangskonfiguration
  while( !stop_criterion ) {                       //Endtemperatur, oder weniger als Min Austauschungen

    while( !loop_criterion ) {                     //Mindestanzahl Versuche bei einer Temperatur

      j = generate( X );                           //Generiere neue Konfiguration, also Austausch zweier Module

      if( accept( cost( j ), cost( X ), T ) )      //spezielle Akzeptanzfunktion
        X = j;                                     //akzeptierte neue Konfiguration abspeichern
    }
    T = update( T );                               //langsames Senken der Temp. (z.B. T = 0.9T)
   }
}
```

Pseudo Code: Simulated Annealing

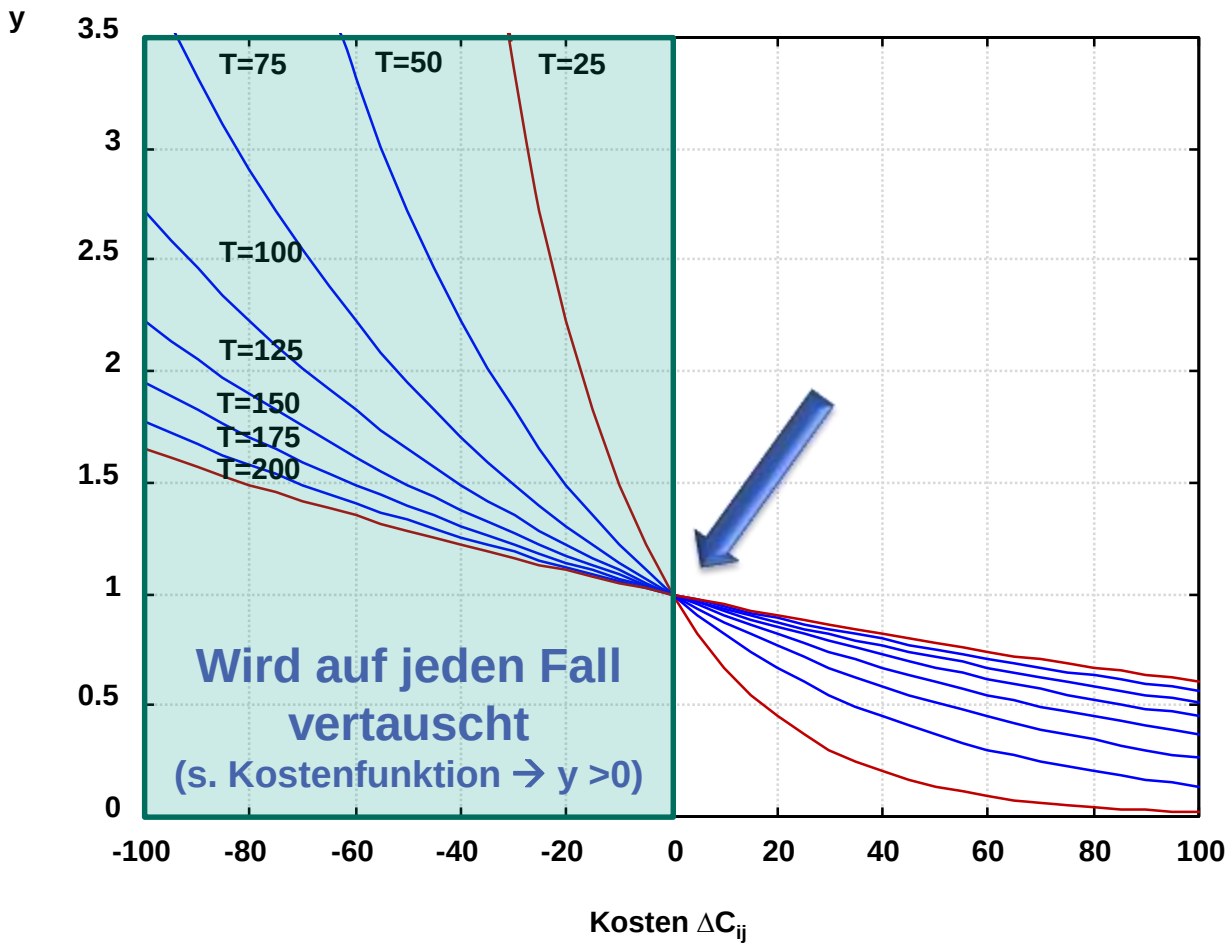
```
simulate_annealing() {
  T = T0;                                          //Anfangswert der Temperatur
  X = j0;                                          //Anfangskonfiguration
  while( !stop_criterion ) {                       //Endtemperatur, oder weniger als Min Austauschungen

    while( !loop_criterion ) {                     //Mindestanzahl Versuche bei einer Temperatur

      j = generate( X );                           //Generiere neue Konfiguration, also Austausch zweier Module

      if( accept( cost( j ), cost( X ), T ) )      //spezielle Akzeptanzfunktion
        X = j;                                     //akzeptierte neue Konfiguration abspeichern
    }
    T = update( T );                               //langsames Senken der Temp. (z.B. T = 0.9T)
   }
}
accept( cost( j ), cost( i ), T ) {                //liefert eine 1, wenn ein Austausch akzeptiert wird,
                                                   //wenn nicht, eine 0
  diff_cost = cost( j ) - cost( i );               //Differenz der Kosten berechnen
  y = exp( -diff_cost / T );                       //Wahrscheinlichkeitsfunktion nach Boltzmannstatistik
                                                   //je höher die Temperatur, desto größer wird y für diff_cost>0
  r = random;                                      //Zufallszahl zwischen 0 und 1
                                                   // → den lokalen Optima entfliehen
if( r < y )                                        //bei höheren Temperaturen mit höherer Wahrscheinlichkeit
    return 1;                                      //akzeptiert,
else
    return 0;                                      //oder nicht
}
```

temperaturproportionale Wahrscheinlichkeit

Simulated Annealing Akzeptanz

```
simulate_annealing() {
  T = T0;
  X = j0;
  while( !stop_criterion ) {

    while( !loop_criterion ) {

      j = generate( X );

      if( accept( cost( j ), cost( X ), T ) )
        X = j;
    }
    T = update( T );
   }
}
accept( cost( j ), cost( i ), T ) {

  diff_cost = cost( j ) - cost( i );
  y = exp( -diff_cost / T );

  r = random;

if( r < y )
    return 1;
else
    return 0;
}
```

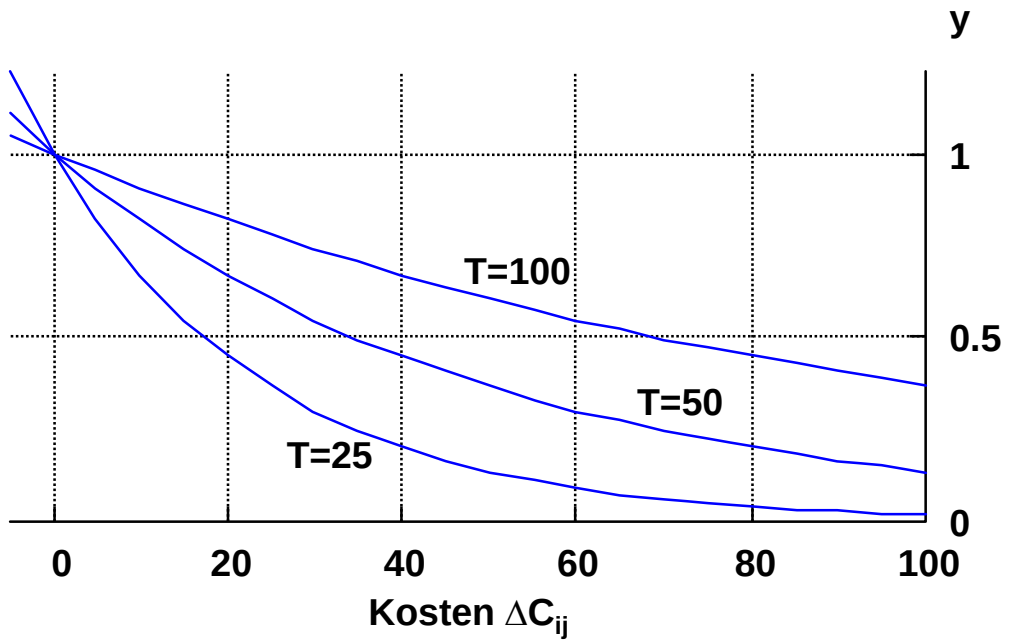
temperaturproportionale Wahrscheinlichkeit




```
y = exp( -diff_cost / T );
```

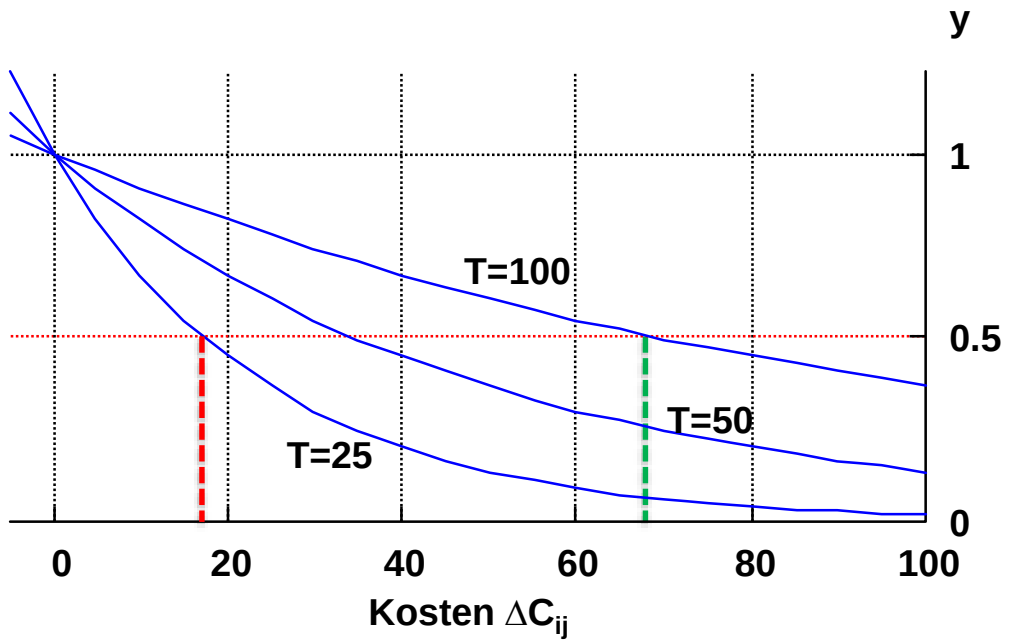
Übung: Simulated Annealing

- Sie sehen die grafische Darstellung für den Fall einer Kostenverschlechterung bei verschiedenen Temperaturen. Beantworten Sie die folgenden Fragen:
- Welche Kostendifferenz wird für T=25 mit einer Wahrscheinlichkeit von 50% akzeptiert?
- Wie groß darf die Kostendifferenz für dieselbe Wahrscheinlichkeit bei T=100 maximal sein?



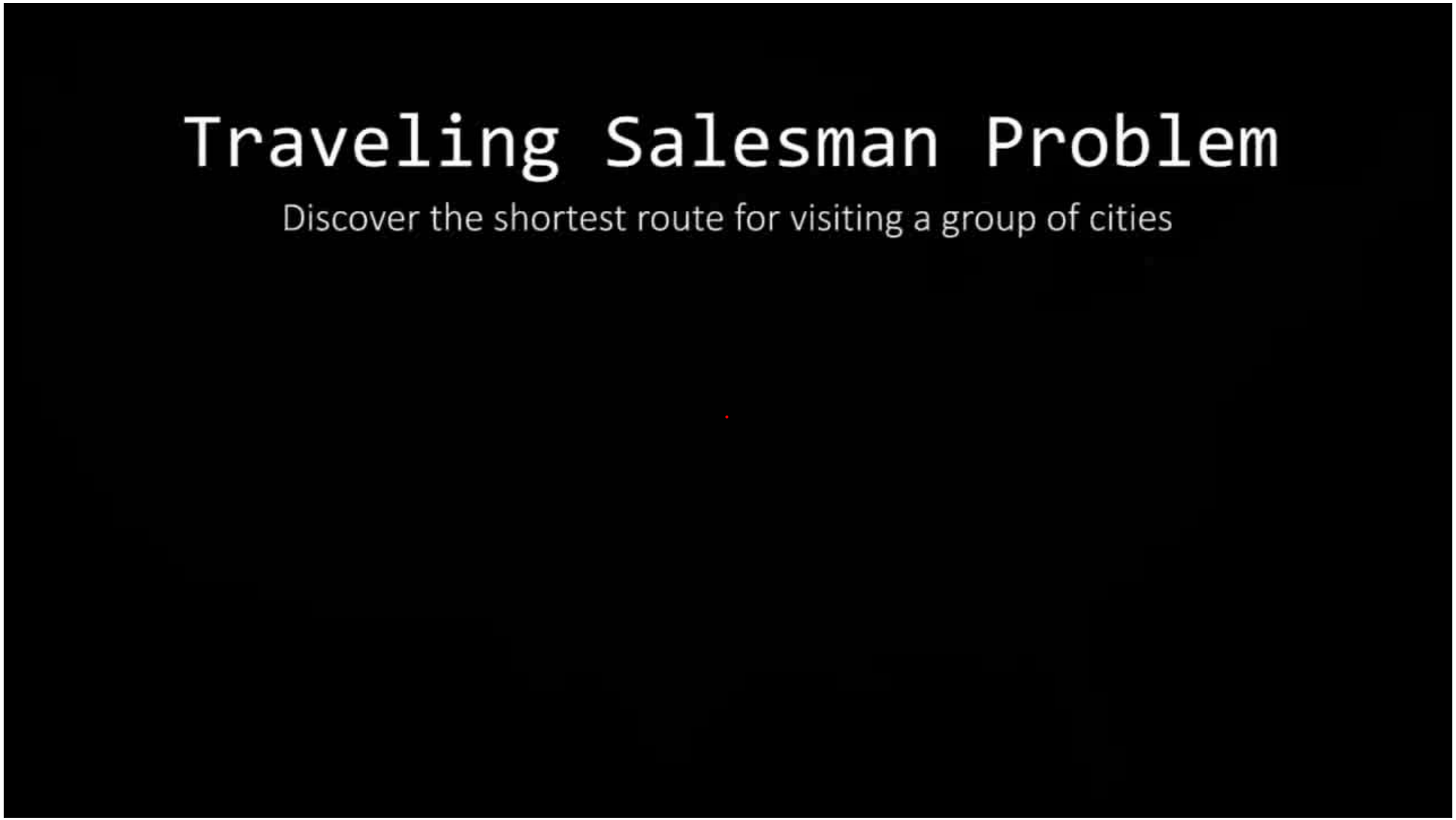
Übung: Simulated Annealing – Lsg.

- Sie sehen die grafische Darstellung für den Fall einer Kostenverschlechterung bei verschiedenen Temperaturen. Beantworten Sie die folgenden Fragen:
- Welche Kostendifferenz wird für T=25 mit einer Wahrscheinlichkeit von 50% akzeptiert?
  Abgelesen: $\Delta C_{ij} = 18$
- Wie groß darf die Kostendifferenz für dieselbe Wahrscheinlichkeit bei T=100 maximal sein?
  Abgelesen: $\Delta C_{ij} = 64$



- Simulated Annealing

TSP


Traveling Salesman Problem
Discover the shortest route for visiting a group of cities

- Optimierungsalgorithmen


Backup
Shift

C++ Lösung

```
class Gegenstand {

   public :
      Gegenstand() {
         gewicht = 1+rand()%MAX_GEWICHT;
         nutzen = 1+rand()%MAX_NUTZEN;
      };
      int gewicht;
      int nutzen;
};
```

```
class Rucksack {

   public :
      int gesamtnutzen();
      int gesamtgewicht();
      bool einpacken(Gegenstand*);
      void ausgabe();
   private :
      set<Gegenstand*> inhalt;
};
```

Pseudo Code

- Sortierung der Gegenstände über Vergleichsfunktion
- 1. Alternative: Sortierung absteigend nach Nutzen

```
bool vergleichsfunktion(Gegenstand* g1, Gegenstand* g2) {
    if (g1->nutzen > g2->nutzen) return true;
    if (g1->nutzen == g2->nutzen && g1->gewicht < g2->gewicht)
        return true;
    return false;
}
```

- 2. Alternative: Sortierung nach Verhältnis Nutzen/Gewicht liefert bessere Ergebnisse

```
bool vergleichsfunktion2(Gegenstand* g1, Gegenstand* g2) {
    if ((float)g1->nutzen/g1->gewicht >
        (float)g2->nutzen/g2->gewicht) return true;
    return false;
}
```

Greedy Algorithmus

- Sortieren der Liste nach Greedy-Präferenz (Vergleichsfunktion)
- Dann entsprechend dieser Reihenfolge „in den Rucksack stopfen“

```
Rucksack packenGreedy(list<Gegenstand*> auswahl) {

   Rucksack rs;
   auswahl.sort(vergleichsfunktion);
   list<Gegenstand*>::const_iterator pos;
   for (pos = auswahl.begin();
         rs.gesamtgewicht() < KAPAZITAET
         && pos != auswahl.end(); pos++) {
      Gegenstand* g = *pos;
      rs.einpacken(g);
   }
   return rs;
}
```

Auswahl nach Gewicht

```
static Rucksack packeGierigNachGewicht() {
        Rucksack r = new Rucksack();
        while (true) {
                int pos=-1; int bestesGewicht = MAX_GEWICHT+1;
                for (int i=0; i<auswahlObjekte.length; i++)
                        if (r.auswahl[i] == false &&
                                auswahlObjekte[i].gewicht < bestesGewicht &&
                                r.gewicht() + auswahlObjekte[i].gewicht <=
                                        kapazitaet) {
                           bestesGewicht = auswahlObjekte[i].gewicht;
                           pos = i;
                        }
                if (pos == -1) break;
                else r.auswahl[pos] = true;
        }
        return r;
}
```

Auswahl nach Nutzen

```
static Rucksack packeGierigNachNutzen() {
        Rucksack r = new Rucksack();
        while (true) {
                int pos=-1; int besterNutzen = 0;
                for (int i=0; i<auswahlObjekte.length; i++)
                        if (r.auswahl[i] == false &&
                                auswahlObjekte[i].nutzen > besterNutzen &&
                                r.gewicht() + auswahlObjekte[i].gewicht <=
                                        kapazitaet) {
                           besterNutzen = auswahlObjekte[i].nutzen;
                           pos = i;
                        }
                if (pos == -1) break;
                else r.auswahl[pos] = true;
        }
        return r;
}
```