

14. Heuristiken und Lokale Suche

Verfahren:

- Konstruktionsverfahren
- Verbesserungsheuristiken – Lokale Suchverfahren
- Metaheuristiken
 - Simulated Annealing
 - Tabu Suche
 - Evolutionäre Algorithmen

Konstruktionsverfahren

Eigenschaften:

- Meist sehr **problemspezifisch** (d.h. sie sind oft nur für ein bestimmtes Problem oder eine bestimmte Art von Problemen konzipiert).
- Häufig **intuitiv gestaltet**.
- Basiert oft auf dem **Greedy-Prinzip**

Greedy-construction-heuristic:

Greedy-construction-heuristic:

$x \leftarrow$ leere Lösung

while x ist keine vollständige Lösung

$e \leftarrow$ die aktuell nützlichste Erweiterung von x

$x \leftarrow x \oplus e$

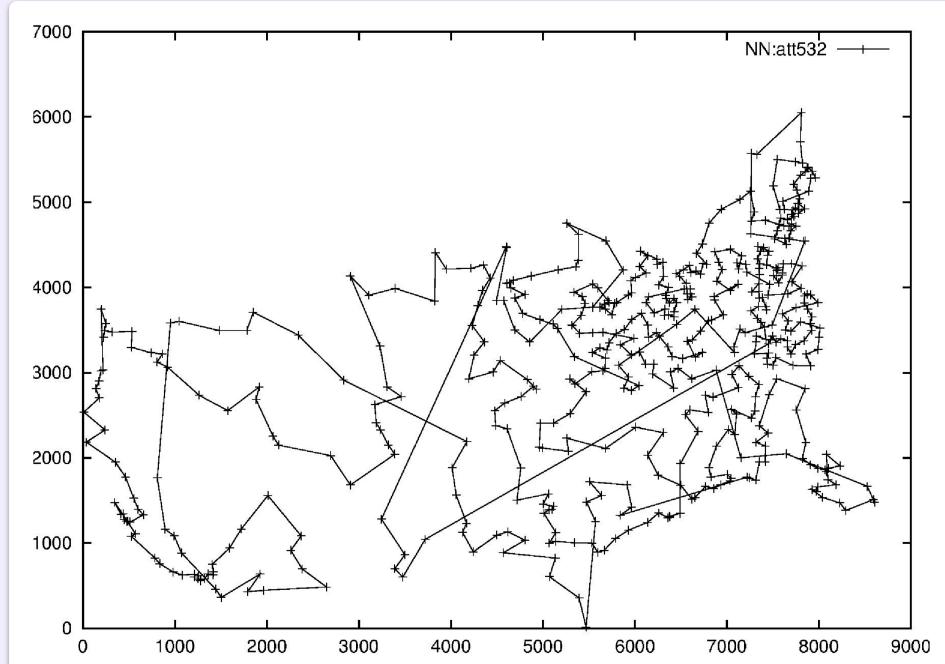
- Beginnt mit einer leeren Lösung x .
- In einer `while`-Schleife wird die Lösung schrittweise erweitert, solange sie nicht vollständig ist.
- In jedem Schritt wird die aktuell nützlichste Erweiterung e für die Lösung x bestimmt.
- Die Lösung x wird um diese Erweiterung e ergänzt ($x \leftarrow x \oplus e$).

☰ Beispiele für Konstruktionsverfahren

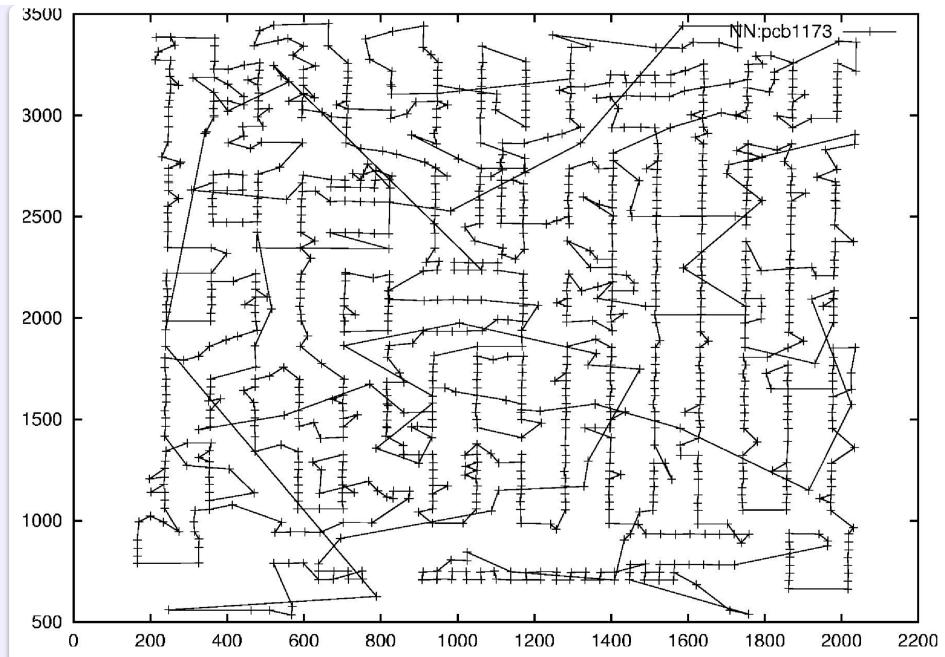
Wir haben bereits einige Konstruktionsverfahren kennengelernt:

- **Minimaler Spannbaum:**
 - Greedy Algorithmen von Prim und Kruskal

- Berechnen ein *globales Optimum* und sind daher *exakte Algorithmen*.
- **0/1-Rucksackproblem:**
 - *First-Fit-Heuristik*: Berücksichtigt alle Gegenstände in nicht aufsteigendem Verhältnis von Wert zu Gewicht (d.h., absteigend nach Wert/Gewicht sortiert) und packt jeden passenden Gegenstand ein.
- **Vertex Cover Problem**
- **Lastverteilung:**
 - *List Scheduling Algorithmus*
 - Gütegarantie: 2 (d.h., die Lösung ist maximal doppelt so schlecht wie die optimale Lösung).
- **TSP (Traveling Salesperson Problem):**
 - *Spanning-Tree-Heuristik*:
 - Gütegarantie: 2 für metrisches TSP (wenn die Dreiecksungleichung gilt, d.h. der direkte Weg ist nie länger als ein Umweg).
 - *Nearest-Neighbor-Heuristik*:
 - Starte bei einem Knoten und gehe immer zum nächsten nicht besuchten Nachbarn.



(Man nimmt immer die nächste Stadt die noch nicht genommen wurde)



(Printerplatte wo Chips draufgelötet werden. Bei allen Kreuzern müssen Löcher gemacht werden)

☰ Weiteres Beispiel: Insertion-Heuristiken für das TSP

Vorgehen:

- Starte mit einer Tour von 3 Knoten oder im Fall von Euklidischen Instanzen mit der konvexen Hülle.
- Füge die restlichen Knoten schrittweise zur Tour hinzu, bis alle Knoten eingebunden sind.

Unterschiedliche Strategien für die Auswahl des nächsten einzufügenden Knotens
(keine ist immer am besten):

- **Nearest Insertion:** Wähle den Knoten, der zu einem bereits in der Tour befindlichen Knoten am nächsten ist.
- **Cheapest Insertion:** Wähle den Knoten, dessen Einfügen die Tourlänge am wenigsten erhöht.
- **Farthest Insertion:** Wähle den Knoten, der zu einem bereits in der Tour befindlichen Knoten am weitesten entfernt ist.
- **Random Insertion:** Wähle Knoten zufällig.

Strategien für das Einfügen des neuen Knotens (sobald der nächste Knoten ausgewählt ist):

- Füge den neuen Knoten nach dem Knoten in der bestehenden Tour ein, der ihm am nächsten ist (oder nach dem nächstbesten Knoten).

- Füge den neuen Knoten so ein, dass eine minimale Zunahme der Tourlänge verursacht wird.

Praktische Ergebnisse:

- Oft *10-20% über dem Optimum*.
- *Keine konstante Gütegarantie* für das metrische TSP (d.h., die Abweichung vom Optimum kann beliebig groß werden, auch wenn die Dreiecksungleichung gilt).

Lokale Suche

Verbesserungsheuristiken

Konstruktionsheuristiken sind oft intuitiv und schnell, liefern aber häufig keine ausreichend guten Lösungen.

Idee der Lokalen Suche:

- Versuche eine Ausgangslösung durch kleine Änderungen iterativ zu verbessern.
- **i.A. (in aller Regel) keine Gütegarantien.**
- In der Praxis aber oft deutliche Verbesserung von Lösungen.
- Für die meisten Anwendungen akzeptable Laufzeiten.

Lokale Suche

Prinzip:

```

 $x \leftarrow$  Ausgangslösung
while Abbruchkriterium nicht erfüllt
    Wähle  $x' \in N(x)$ 
    if  $x'$  besser als  $x$ 
         $x \leftarrow x';$ 

```

- $N(x)$: Nachbarschaft von x (Menge aller Lösungen, die durch eine "kleine" Änderung aus x entstehen).

Komponenten, die für eine lokale Suche wichtig sind

- **Lösungsrepräsentation:** Wie wird eine mögliche Lösung dargestellt?
- **Nachbarschaftsstruktur:** Welche Lösungen werden von einer aktuellen ausgehend unmittelbar in Erwägung gezogen? (Definition von $N(x)$)
- **Schrittfunktion:** Wie wird die Nachbarschaft durchsucht und welche Nachbarlösung wird als Nächstes gewählt?
- **Terminierungskriterium:** Wann wird die Suche beendet? (z.B. maximale Iterationen, keine Verbesserung über eine bestimmte Anzahl von Iterationen, Erreichen eines bestimmten Qualitätsniveaus).

Definition - Nachbarschaftsstruktur

Eine Nachbarschaftsstruktur ist eine Funktion $N : S \rightarrow 2^S$, die jeder gültigen Lösung $x \in S$ (wobei S der Lösungsraum ist) eine Menge von Nachbarn $N(x) \subseteq S$ zuweist.

- $N(x)$ wird auch **Nachbarschaft von x** genannt.

Eigenschaften:

- Die Nachbarschaft ist üblicherweise implizit durch *mögliche Veränderungen (Züge, Moves)* definiert. (Das heißt, man beschreibt, welche kleinen Änderungen man an einer Lösung vornehmen darf, um eine Nachbarlösung zu erhalten, anstatt explizit alle Nachbarn aufzulisten.)
- **Darstellung als Nachbarschaftsgraph möglich:**
 - Knoten entsprechen den Lösungen.
 - Eine Kante (u, v) existiert, wenn $v \in N(u)$ (d.h., v ist ein Nachbar von u).
- Es gilt, die **Größe der Nachbarschaft vs. Suchaufwand** abzuwägen.
 - Eine größere Nachbarschaft kann bessere Lösungen finden, erfordert aber mehr Rechenzeit pro Iteration.
 - Eine kleinere Nachbarschaft ist schneller zu durchsuchen, läuft aber Gefahr, in lokalen Optima stecken zu bleiben.

Lokale Suche: Vertex Cover

Vertex Cover (VC):

- Gegeben sei ein Graph $G = (V, E)$.
- Finde eine minimale Teilmenge $C \subseteq V$ von Knoten, sodass für jede Kante $(u, v) \in E$ entweder $u \in C$ oder $v \in C$ (oder beide) gilt. (Das heißt, jeder Kante muss mindestens ein Endpunkt im Vertex Cover haben.)

Nachbarschaftsstruktur für Vertex Cover:

- Eine Lösung $C' \in N(C)$ (Nachbar von C) existiert, wenn C' aus C durch Löschen eines einzigen Knotens erzeugt werden kann und C' immer noch ein gültiges Vertex Cover ist.
- Die Größe der Nachbarschaft $|N(C)| = O(|V|)$ (proportional zur Anzahl der Knoten im Graphen).

Lokale Suche für Vertex Cover (Beispiel):

- **Starte mit einer initialen Vertex Cover C , z.B. $C = V$** (die Menge aller Knoten ist immer ein gültiges Vertex Cover, wenn auch selten minimal).
- **Verbesserungsschritt:** Wenn es einen Nachbarn $C' \in N(C)$ gibt, der ein *gültiges Vertex Cover* ist (d.h., C' ist immer noch ein VC, obwohl ein Knoten entfernt wurde), dann ist dieser Nachbar immer eine *bessere Lösung*, da $|C'| = |C| - 1$ (die Kardinalität, also die Anzahl der Knoten, ist um 1 geringer).
- **Iteration:** Ersetze die aktuelle Lösung C durch solches C' und wiederhole den Prozess.

Diese lokale Suche terminiert nach $O(|V|)$ Schritten, da in jedem Schritt die Größe des Vertex Covers um 1 reduziert wird und die minimale Größe 0 ist.

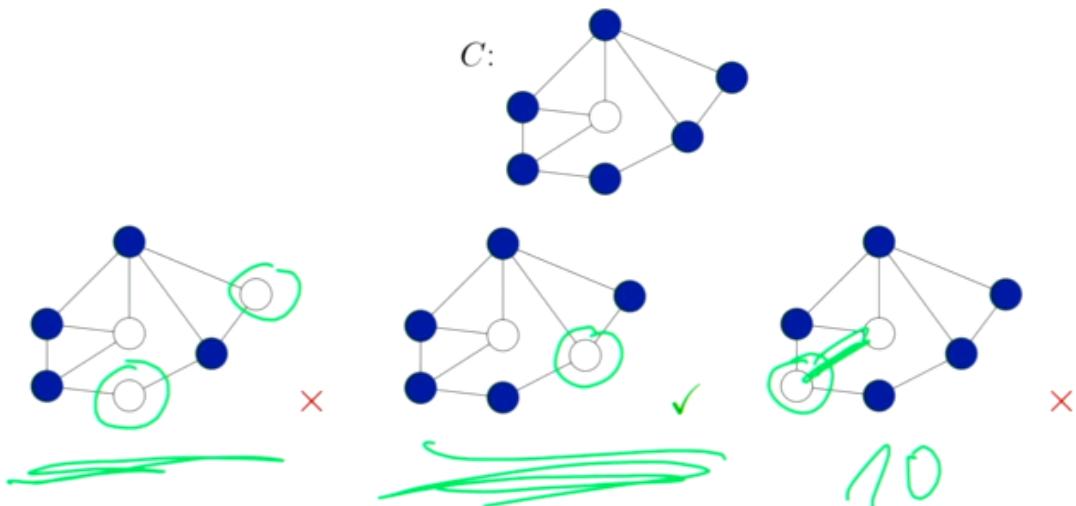
Die lokale Suche liefert nicht immer eine optimale Lösung.

Lokales Optimum:

- Ein *lokales Optimum* ist eine Lösung, bei der kein Nachbar (im Sinne der definierten Nachbarschaftsstruktur) strikt besser ist.
- Die lokale Suche kann die aktuelle Lösung daher nicht weiter verbessern, auch wenn es global bessere Lösungen geben könnte, die nicht direkt in der Nachbarschaft liegen.

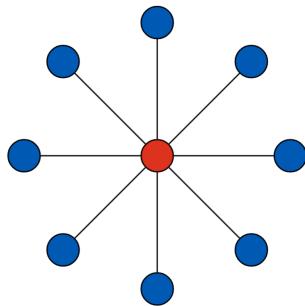
☰ Beispiel Slido

Frage 1: Welche blau markierten Mengen sind in der Nachbarschaft $N(C)$ enthalten?



Man darf in unserer Definition nur einen Knoten entfernen solange das Ergebnis stimmt. Beim 1. werden 2 entfernt und beim 3. Ist die eine Kante nicht mehr abgedeckt und dadurch kein Vertex Cover mehr. Daher stimmt nur 2..

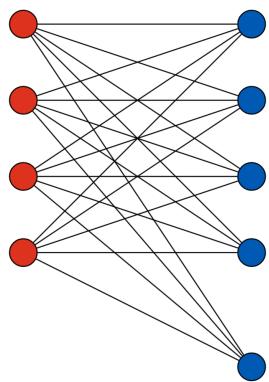
☰ Beispiel 1



(Globales) Optimum = roter Knoten in der Mitte
Lokales Optimum = alle blauen Knoten

Beispiel für eine Grenze von dieser Heuristik
Hier kann man keinen einzelnen Knoten entfernen.

☰ Beispiel 2



(Globales) Optimum = alle roten Knoten
Lokales Optimum = alle blauen Knoten

Hier kann man auch nichts entfernen ohne, dass es ungültig wird

☰ Beispiel 3

Optimum: Alle geraden Knoten



Lokales Optimum: Jeder dritte Knoten wird ausgelassen.



Hier auch nicht.

Die lokale Suche mit dieser Nachbarschaftsstruktur hat einfach ihre Grenzen

Lokale vs. globale Optima

Für die Maximierung einer Zielfunktion $f(x)$ gilt:

- Ein **lokales Maximum** in Bezug auf eine Nachbarschaftsstruktur \mathcal{N} ist eine Lösung x für die gilt: $f(x) \geq f(x')$ für alle $x' \in \mathcal{N}(x)$.
- **Intuitive Erklärung:** Ein lokales Maximum ist der höchste Punkt in einer bestimmten "Umgebung" (Nachbarschaft) der Lösung. Es ist nicht unbedingt der höchste Punkt der gesamten Funktion, sondern nur im unmittelbaren Umfeld.
- Die Nachbarschaftsstruktur bestimmt, welche Lösungen lokal optimal sind.

Verbesserungsmöglichkeiten (um bessere/globale Optima zu finden):

- Verwendung anderer/größerer Nachbarschaften.
- Iterierte lokale Suche: Wende lokale Suche wiederholt auf unterschiedliche Startlösungen an.
- Kombination unterschiedlicher lokaler Suchmethoden.

Lokale Suche: Vertex Cover

Alternative Nachbarschaft \mathcal{N}'

- Anstatt nur einen Knoten zu entfernen, können auch zwei Knoten entfernt und einer hinzugefügt werden.
- Formal beinhaltet $\mathcal{N}'(C)$ eine Knotenmengen $S \in \mathcal{N}(C)$ oder wenn S durch Hinzufügen eines Knotens von $V \setminus C$ und Entfernen von zwei Knoten aus C gebildet werden kann

und noch immer ein Vertex Cover ist.

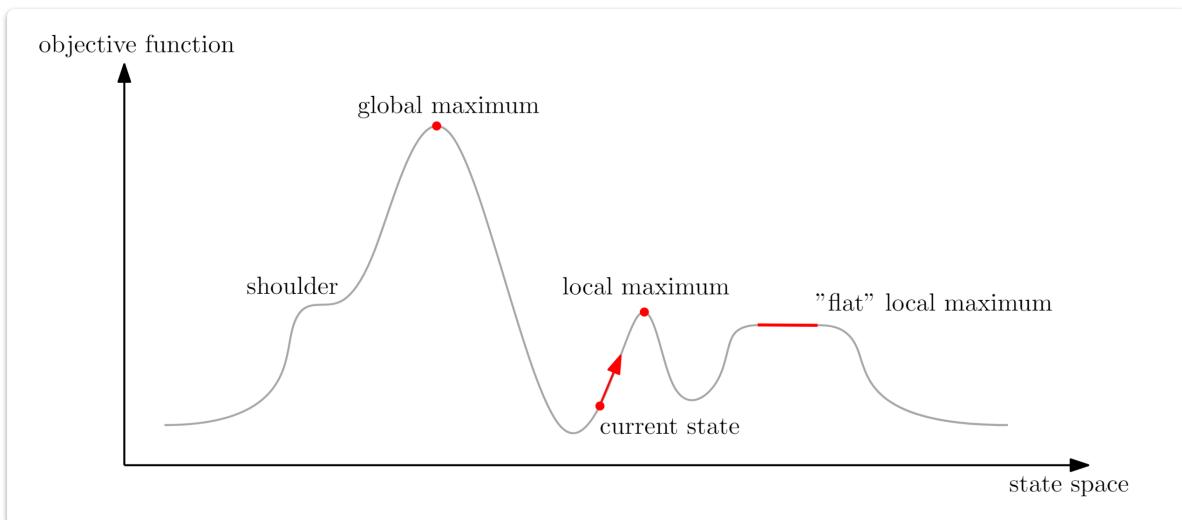
- $O(|\mathcal{V}|^3) \rightarrow$ lokale Suche wird aufwändiger.

- **Intuitive Erklärung:** Die Komplexität der lokalen Suche erhöht sich erheblich, wenn man eine komplexere Nachbarschaftsdefinition (z.B. Entfernen von zwei Knoten und Hinzufügen von einem) verwendet. Die Notation $O(|\mathcal{V}|^3)$ bedeutet, dass die Rechenzeit proportional zur dritten Potenz der Anzahl der Knoten im Graphen ($|\mathcal{V}|$) anwächst. Das ist deutlich mehr Aufwand als z.B. $O(|\mathcal{V}|)$, was für eine einfache Nachbarschaft der Fall wäre.

Lokale vs. globale Optima

Generelles Problem der lokalen Suche:

Es wird nur ein nächstes **lokales Optimum** gefunden, wir sind aber i.A. an einem globalen Optimum interessiert.



Lokale Suche: SAT

- **Gegeben:** Eine boolesche Formel in konjunktiver Normalform mit Variablen x_1, \dots, x_n .
 - **Beispiel:** $(\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$
- **Gesucht:** Variablenzuweisung, sodass alle Klauseln erfüllt werden.
- **Optimierungsvariante:** MAX-SAT - Maximiere die Anzahl der erfüllten Klauseln.

Hinweise:

- Repräsentation von Lösungen mit einem binären Vektor $x = \{0, 1\}^n$.
- **NP-vollständig** (Dies bedeutet, dass es wahrscheinlich keinen effizienten Algorithmus zur Lösung des Problems gibt, der in Polynomialzeit läuft).

k-flip Nachbarschaft für binäre Vektoren

- Nachbarlösungen haben eine *Hamming-Distanz* $\leq k$.
 - d.h., sie unterscheiden sich in bis zu k Bits (Positionen im binären Vektor).
- Größe der Nachbarschaft: $O(n^k)$ (Die Anzahl der Nachbarn wächst polynomiell mit n und exponentiell mit k).

Schrittfunktion – Wahl von $x' \in N(x)$ in der lokalen Suche

Auswahlmöglichkeiten:

- **Best Improvement:** Durchsuche $N(x)$ *vollständig* und nimm eine beste Nachbarlösung.
- **First Improvement:** Durchsuche $N(x)$ in einer bestimmten Reihenfolge, nimm *erste* Lösung, die besser als x ist.
- **Random Neighbor:** Wähle eine *zufällige* Lösung aus $N(x)$.

Hinweise:

- Wahl kann starken Einfluss auf Performance haben.
- Allgemein ist kein Verfahren immer besser als ein anderes (die Effektivität hängt oft vom spezifischen Problem und den Daten ab).
- Beispielsweise ist ein Durchlauf von *Random Neighbor* meist schneller, dafür benötigt *Best Improvement* oft erheblich weniger Iterationen (da es in jedem Schritt die optimale Verbesserung wählt).

Abbruchkriterium

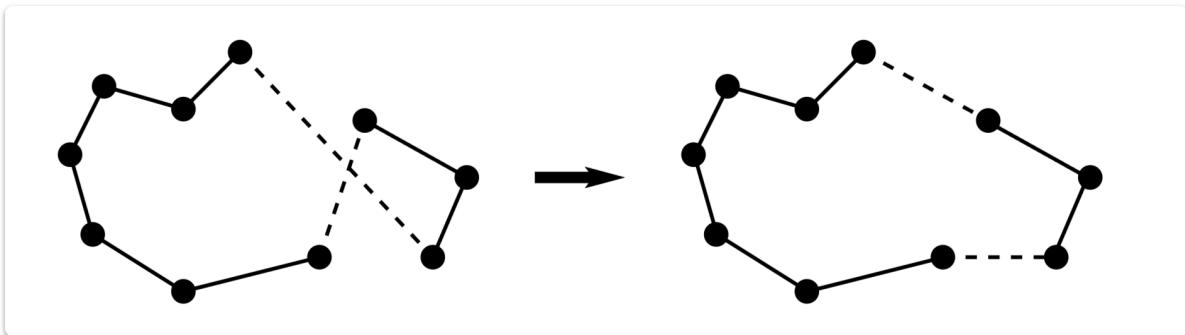
- Meist wird die lokale Suche beendet, wenn ein *lokales Optimum* erreicht wurde (ein Zustand, in dem keine der benachbarten Lösungen besser ist).
- Bei einer *Random Neighbor* Schrittfunktion kann ein solches jedoch nicht direkt erkannt werden (da man nicht systematisch alle Nachbarn prüft).
- Manchmal ist eine vollständige lokale Suche auch zu *zeitaufwendig*.

Alternativen für Abbruch:

- nach einer bestimmten Iterationsanzahl oder Zeit.
- wenn eine ausreichend gute Lösung gefunden wurde.
- wenn keine weitere Verbesserung über eine bestimmte Anzahl letzter Iterationen erreicht wurde (Konvergenz).

Lokale Suche für das symmetrische TSP

Nachbarschaftsstruktur: Austausch zweier Kanten ("2-exchange" oder "2-opt").



Größe der Nachbarschaft: $O(n^2)$ (Die Anzahl der möglichen 2-opt Operationen wächst quadratisch mit der Anzahl der Knoten n).

Inkrementelle Evaluierung:

- Berechne den Wert einer Nachbarlösung *effizient* aus dem Wert der aktuellen Lösung unter Berücksichtigung der wegfallenden und hinzukommenden Kanten.
- Benötigt hier nur konstante Zeit im Vergleich zu $O(n)$ Zeit für eine vollständige unabhängige Berechnung des Zielfunktionswerts (was eine Neuberechnung der gesamten Tourlänge bedeuten würde).

2-opt lokale Suche für das symmetrische TSP

Mit *First-Improvement* Schrittfunktion (die erste verbesserte Nachbarlösung wird gewählt).

Vorgehen

- (1) Sei $E(T) = \{(i_1, i_2), (i_2, i_3), \dots, (i_n, i_1)\}$ die Menge der Kanten der aktuellen Tour T und sei $i_{p+1} = i_1$.
 - (Dies stellt die Menge der Kanten in einer zyklischen Tour dar, wobei i_n mit i_1 verbunden ist).
- (2) Sei $Z = \{(i_p, i_{p+1}), (i_q, i_{q+1}) \subset T | 1 \leq p, q < n \wedge p + 1 < q\}$ sei die Menge aller Paare nicht nebeneinanderliegender Kanten. (Dies sind die Kantenpaare, die für einen 2-opt Austausch in Frage kommen, d.h., sie dürfen nicht benachbart sein).
- (3) Für alle Kantenpaare $\{(i_p, i_{p+1}), (i_q, i_{q+1})\}$ aus Z :
 - Falls $c_{i_p i_{q+1}} + c_{i_q i_{p+1}} < c_{i_p i_{p+1}} + c_{i_q i_{q+1}}$ (Wenn die Summe der Längen der neuen Kanten kleiner ist als die Summe der Längen der alten Kanten):
 - $T \leftarrow (T \setminus \{(i_p, i_{p+1}), (i_q, i_{q+1})\}) \cup \{(i_p, i_q), (i_{p+1}, i_{q+1})\}$ (Ersetze die alten Kanten durch die neuen).
 - Gehe zu (4) (Eine Verbesserung wurde gefunden und angewendet).

- (4) retourniere T . (Die verbesserte Tour wird zurückgegeben).

Laufzeitkomplexität:

- Da $|N(x)| = O(n^2)$ (Größe der Nachbarschaft) und jede Nachbarlösung in konstanter Zeit inkrementell evaluiert wird, benötigt eine Iteration $O(n^2)$ Zeit.
- **Frage:** Wieviele Iterationen sind notwendig bis ein lokales Optimum erreicht ist?
 - **Worst-Case:** Bis zu $O(n!)$ Iterationen (ohne Beweis)! (Dies entspricht der Anzahl aller möglichen Permutationen der Knoten, was extrem hoch ist).
 - Die Worst-Case-Laufzeit dieser lokalen Suche ist daher *exponentiell*!
 - **Praxis:** Dennoch ist das Verfahren auch auf großen Instanzen meist *schnell*. Startet man mit einer sinnvollen Ausgangslösung (z.B. einer Heuristik), sind in der Regel nur wenige Iterationen erforderlich, um ein lokales Optimum zu erreichen.

r-opt Nachbarschaft für das Symmetrische TSP

- **Verallgemeinerung:** Die Idee der 2-opt Nachbarschaft kann verallgemeinert werden. Es werden $r \geq 2$ Kanten durch neue ersetzt.

Prinzip der *r*-opt lokalen Suche

- (1) Wähle eine beliebige Anfangstour $T = \{(i_1, i_2), (i_2, i_3), \dots, (i_n, i_1)\}$.
- (2) Sei Z die Menge aller r -elementigen Teilmengen von T . (Dies sind alle möglichen Gruppen von r Kanten, die aus der aktuellen Tour entfernt werden könnten).
- (3) Für alle $R \in Z$: Setze $S = T \setminus R$ und konstruiere alle Touren, die S enthalten. Ist ein S' besser als T , setze $T = S'$ und gehe zu (2). (Finde die beste Art und Weise, die verbleibenden Pfade in S mit r neuen Kanten zu einer vollständigen Tour zu verbinden, die eine Verbesserung darstellt).
- (4) T ist das Ergebnis.

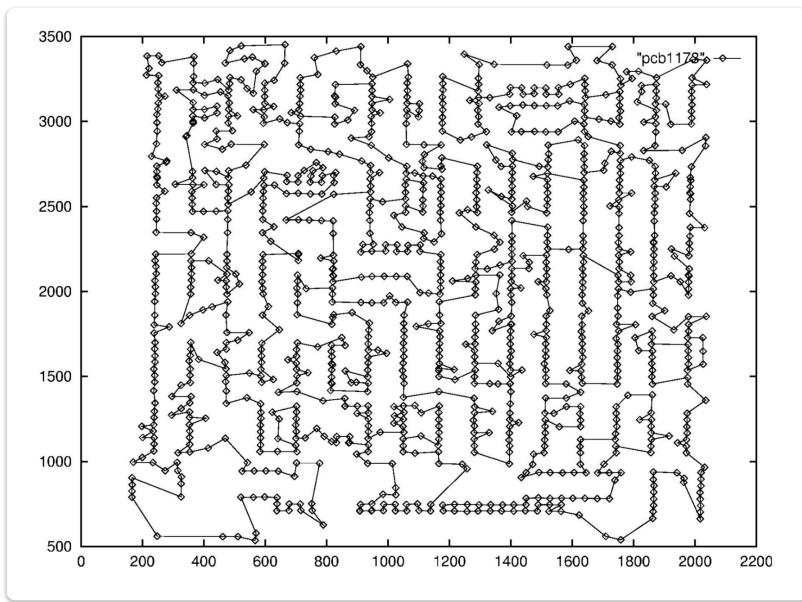
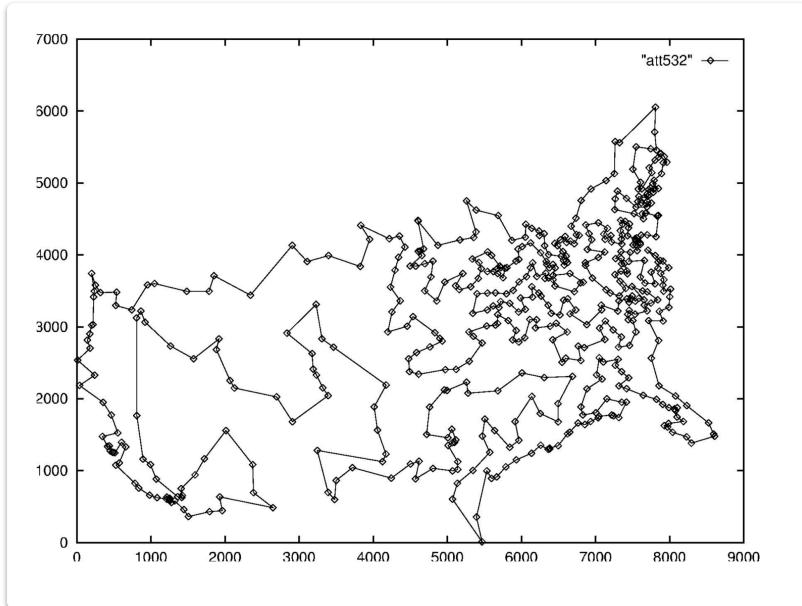
Laufzeitkomplexität der *r*-opt lokalen Suche

Größe einer *r*-opt Nachbarschaft:

- Es gibt $\binom{n}{r} = O(n^r)$ Möglichkeiten r unterschiedliche Kanten aus einer aktuellen Tour zu entfernen. (Dies ist die Anzahl der Kombinationen, n über r).
- Das Entfernen führt zu r nicht zusammenhängenden Pfaden.
- Diese können auf $O(r!)$ Möglichkeiten zu neuen Touren zusammengefügt werden.
- Somit ist $|N(x)| = O(n^r \cdot r!) = O(n^r)$. (Die $r!$ Faktoren sind dominant für kleine r , aber für festes r ist die Abhängigkeit von n entscheidend).

Hinweis: Wie bereits bei 2-opt ist auch hier im allgemeinen Fall die Worst-Case-Anzahl der möglichen Iterationen *nicht polynomiell* beschränkt.

2-opt Lösung für TSP



Hinweis: Die Lösung wurde durch eine lokale Suche mit Random-Neighbor Schrittfunktion gefunden. Es gibt 2 Kanten, die sich kreuzen, daher ist diese Lösung kein lokales Optimum

Zusammenfassung zur lokalen Suche für das TSP

Anwendung:

- **2-opt** wird sehr häufig eingesetzt.
 - Kommt meist auf ca. *6–8% an die optimale Lösung* heran.
- **3-opt** wird manchmal verwendet (deutlich zeitaufwändiger).
 - Kommt meist auf *3–4% an die optimale Lösung* heran.

- **4-opt** ist in der Praxis i.A. bereits zu *zeitaufwändig*.

Hinweis: Die Prozentangaben beziehen sich auf bestimmte Instanzen, die zum Testen der Algorithmen verwendet werden und sollen hier nur einen *groben Richtwert* vermitteln.

Weitere Nachbarschaftsstrukturen:

- **Verschieben eines Knotens an eine andere Position.**
 - Für das *asymmetrische TSP* (wo die Kosten von A nach B nicht unbedingt gleich den Kosten von B nach A sind) häufig besser geeignet.
- **Verschieben einer Teilsequenz an eine andere Position.**
- **Lin-Kernighan Heuristik (1973):**
 - Eine der *führenden, schnellen Heuristiken* für große TSPs.
 - Kommt meist auf *1–2% an das Optimum* heran.
 - *Variable Tiefensuche*: Die Anzahl der ausgetauschten Kanten ist nicht grundsätzlich beschränkt.
 - Es werden jedoch nur „vielversprechende“ Kantenaustausche durchprobiert (dies reduziert den Suchraum erheblich, indem nur vielversprechende Pfade verfolgt werden).

Maximaler Schnitt (Maximal Cut)

- **MAX-CUT:** Gegeben sei ein ungerichteter Graph $G = (V, E)$ mit positiven ganzzahligen Kantengewichten w_{uv} für alle Kanten $(u, v) \in E$. Finde eine Partition der Knoten (A, B) , sodass das Gesamtgewicht von Kanten, die Knoten in den unterschiedlichen Partitionen verbinden, maximiert wird.

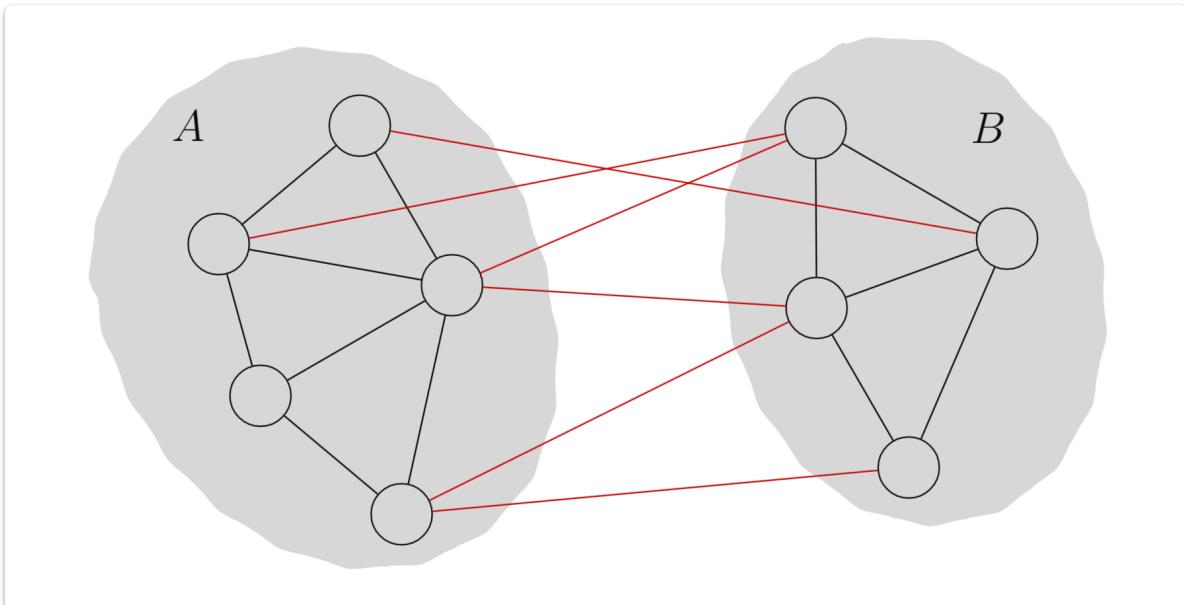
- Formel für das Gewicht der Kanten zwischen Partitionen A und B :

$$w(A, B) := \sum_{u \in A, v \in B} w_{uv}$$

Beispielanwendung:

- n Aktivitäten, m Personen.
- Jede Person möchte an zwei Aktivitäten teilnehmen.
- Plane die Aktivitäten am Morgen und am Nachmittag so, dass eine maximale Anzahl an Personen daran teilnehmen kann (dies entspricht der Maximierung der "Schnittkanten", d.h., der Personen, die an Aktivitäten in beiden Zeiträumen teilnehmen können).

Hinweis: MAX-CUT ist *NP-vollständig*.

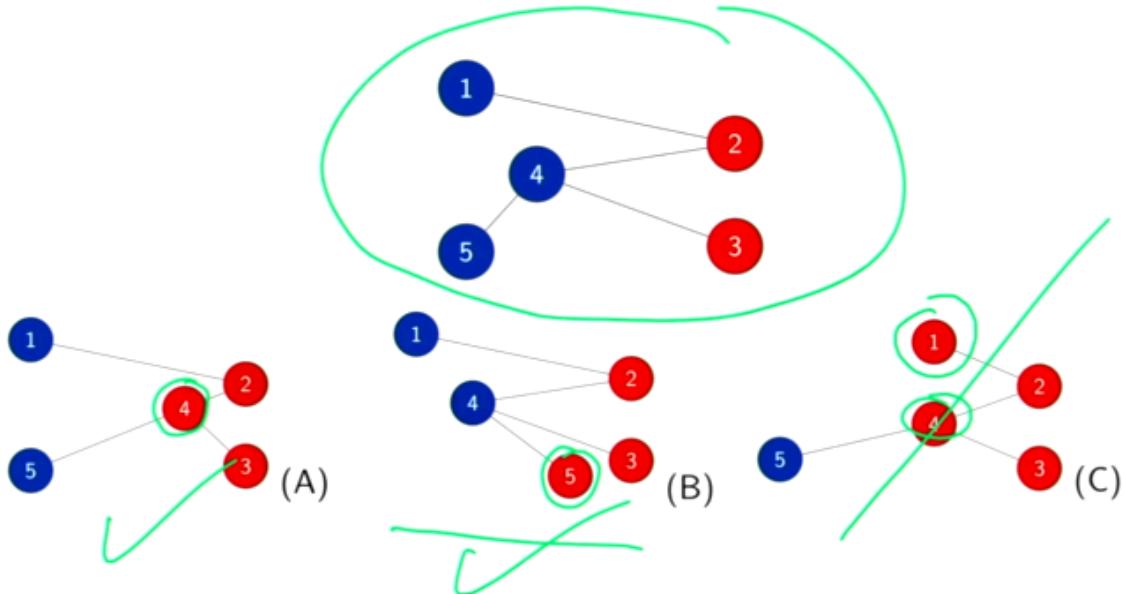


1-Flip-Nachbarschaft: Gegeben sei eine Partition (A, B) . Verschiebe einen Knoten von A nach B oder von B nach A . (Dies ist die einfachste Form der Nachbarschaft bei Partitions-Problemen, bei der nur ein Element seine Gruppenzugehörigkeit ändert).

Algorithmus: Ausgehend von einer gültigen Initiativlösung ist auf die 1-Flip-Nachbarschaft aufbauend unmittelbar eine einfache lokale Suche möglich. (Man beginnt mit einer beliebigen Aufteilung der Knoten und versucht dann, durch das Verschieben einzelner Knoten eine bessere Schnittgröße zu erreichen).

☰ Beispiel

Frage 3: Welche Lösungen sind Teil der Flip-Nachbarschaft?



Analyse der lokalen Suche

Wir können zeigen: Wird die lokale Suche ausgeführt bis eine lokal optimale Lösung erreicht wurde, so gilt eine *Approximationsgüte von 1/2*.

ⓘ Theorem

Theorem: Sei (A, B) eine lokal optimale Partition und sei (A^*, B^*) eine optimale Partition.

Dann ist

$$w(A, B) \geq \frac{1}{2} \sum_{e \in E} w_e \geq \frac{1}{2} w(A^*, B^*)$$

(Gewichte sind *nicht negativ*)

✓ Beweis

Beweis:

- Lokale Optimalität bedeutet, dass für alle

$$u \in A : \sum_{v \in A} w_{uv} \leq \sum_{v \in B} w_{uv}$$

Das Aufsummieren aller Ungleichungen ergibt:

$$2 \sum_{\{u,v\} \subseteq A} w_{uv} \leq \sum_{u \in A, v \in B} w_{uv} = w(A, B)$$

- Ähnlich ist $2 \sum_{\{u,v\} \subseteq B} w_{uv} \leq \sum_{u \in A, v \in B} w_{uv} = w(A, B)$
- Nun gilt,

$$\sum_{e \in E} w_e = \underbrace{\sum_{\{u,v\} \subseteq A} w_{uv}}_{\leq \frac{1}{2}w(A, B)} + \underbrace{\sum_{u \in A, v \in B} w_{uv}}_{w(A, B)} + \underbrace{\sum_{\{u,v\} \subseteq B} w_{uv}}_{\leq \frac{1}{2}w(A, B)} \leq 2w(A, B) \quad \square$$

- Jede Kante wird einmal gezählt.

Metaheuristiken

🔥 Definition

- **Metaheuristiken:** Sind problemunabhängig formulierte Algorithmen zur heuristischen Lösung schwieriger Optimierungsaufgaben.
 - **Anpassung:** Teile dieser Algorithmen müssen an das jeweilige Problem angepasst werden, wie beispielsweise die Nachbarschaftsstruktur auch in der lokalen Suche.

Betrachtete Metaheuristiken:

- Simulated Annealing
- Tabu-Suche
- Evolutionäre Algorithmen

Simulated Annealing (SA)

- Inspiriert durch den physikalischen Prozess der langsamen Abkühlung von Materialien zur Erreichung einer stabilen Kristallstruktur, z.B. nach dem Glühen eines Metalls.
- **Grundlegende Idee:** Auch *schlechtere Nachbarlösungen werden mit einer bestimmten Wahrscheinlichkeit akzeptiert*.
- **Schrittfunktion:** I.A. Random Neighbor (es wird ein zufälliger Nachbar als nächste Lösung gewählt).

Variablen:

- Z : (Pseudo-)Zufallszahl $\in [0, 1]$
- T : „Temperatur“

```

Simulated-Annealing():
t ← 0
T ← Tinit
x ← Ausgangslösung
while Abbruchkriterium nicht erfüllt
    Wähle  $x' \in N(x)$  zufällig
    if  $x'$  besser als  $x$ 
         $x \leftarrow x'$ 
    elseif  $Z < e^{-|f(x') - f(x)|/T}$ 
         $x \leftarrow x'$ 
     $T \leftarrow g(T, t)$ 
     $t \leftarrow t + 1$ 

```

Metropolis-Kriterium

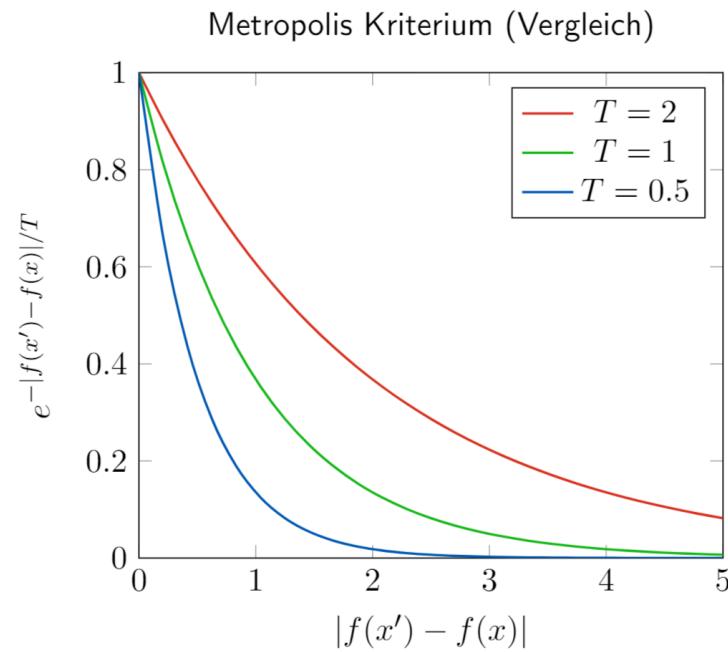
(Das grün markierte ist das, was durch SA dazugekommen ist.) Mit der Wahrscheinlichkeit die hier ausgedrückt wird, akzeptieren wir auch schlechtere Versuche

Metropolis Kriterium

Das Metropolis-Kriterium ist die Akzeptanzbedingung für schlechtere Lösungen in Simulated Annealing:

$$Z < e^{-|f(x') - f(x)|/T}$$

- Z : Zufallszahl $\in [0, 1]$
- $f(x')$: Zielfunktionswert der Nachbarlösung
- $f(x)$: Zielfunktionswert der aktuellen Lösung
- T : Aktuelle Temperatur



Das Diagramm zeigt, wie die Akzeptanzwahrscheinlichkeit (y-Achse) in Abhängigkeit von der Verschlechterung ($|f(x') - f(x)|$, x-Achse) für verschiedene Temperaturen (T) verläuft.

Eigenschaften des Metropolis-Kriteriums

- Nur geringfügig schlechtere Lösungen werden mit höherer Wahrscheinlichkeit akzeptiert als viel schlechtere.
 - Dies ist im Graphen ersichtlich: Für einen gegebenen T -Wert sinkt die Akzeptanzwahrscheinlichkeit drastisch, je größer die Verschlechterung $|f(x') - f(x)|$ wird.
- Anfangs, bei hoher Temperatur T , werden schlechtere Lösungen mit größerer Wahrscheinlichkeit akzeptiert als im späteren Verlauf bei niedrigerer Temperatur.
 - Im Graphen verschiebt sich die Kurve bei höherem T (z.B. $T = 2$) nach oben und ist flacher, was bedeutet, dass für dieselbe Verschlechterung eine höhere Akzeptanzwahrscheinlichkeit besteht als bei niedrigerem T (z.B. $T = 0.5$).
 - Dies ist ein Kernpunkt von Simulated Annealing: Am Anfang kann der Algorithmus "sprunghafter" sein und auch schlechtere Lösungen erkunden, um lokale Optima zu verlassen. Mit sinkender Temperatur wird der Algorithmus "konservativer" und konzentriert sich auf die Verfeinerung guter Lösungen.

Abkühlungsplan

Der Abkühlungsplan definiert, wie die Temperatur T im Laufe des Simulated Annealing Algorithmus reduziert wird. Dies ist entscheidend für das Konvergenzverhalten des Algorithmus.

Geometrisches Abkühlen

Dies ist eine gängige und einfache Methode zur Temperaturreduktion.

- **Faustregel für T_{init} (Anfangstemperatur):**
 - $T_{init} : f_{max} - f_{min}$, wobei f_{max} bzw. f_{min} eine obere bzw. untere Schranke oder Schätzung für den maximalen/minimalen Zielfunktionswert sind.
- **Abkühlfunktion $g(T, t)$:**
 - $g(T, t) = T \cdot \alpha$, mit $\alpha < 1$ (z.B. 0,999).
 - **Erklärung:** Die Temperatur wird in jeder Iteration um einen konstanten Faktor α multipliziert und sinkt somit exponentiell. Ein Wert von $\alpha = 0,999$ bedeutet eine sehr langsame Abkühlung, was mehr Zeit zum Erkunden des Lösungsraums lässt, aber auch länger dauert.
- **Häufige Praxis:**
 - Häufig wird die Temperatur auch über einige (z.B. $|N(x)|$) Iterationen *gleich gelassen* und dann jeweils *etwas stärker reduziert*.

Adaptives Abkühlen

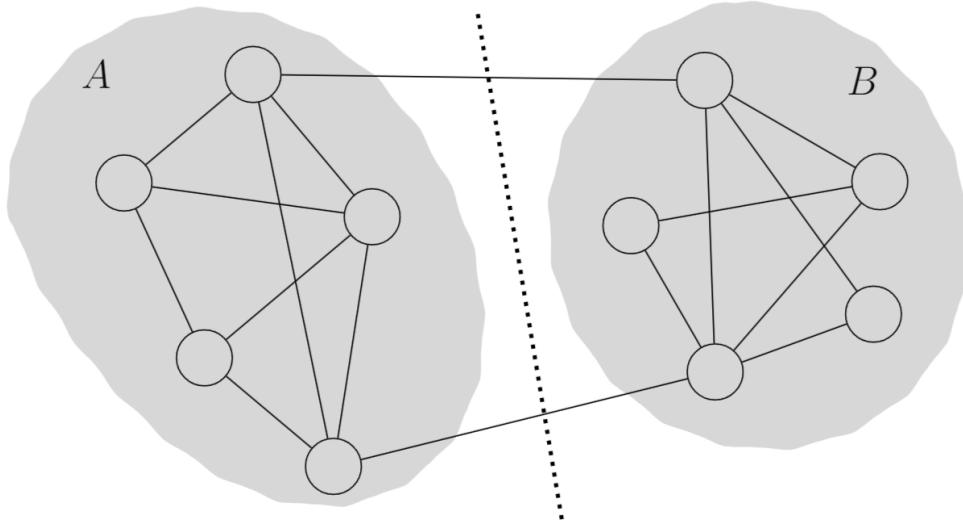
- Es wird der **Anteil der Verbesserungen an den letzten erzeugten Lösungen gemessen** und auf Grund dessen T stärker oder schwächer reduziert.

Beispiel: Simulated Annealing (SA) für Graph-Bipartitionierung

SA ist eine der ersten Anwendungen für die Graph-Bipartitionierung.

Definition für Graph-Bipartitionierung

- **Gegeben:** Ein Graph $G = (V, E)$, wobei V die Menge der Knoten und E die Menge der Kanten ist.
- **Gesucht:** Eine Partitionierung (Aufteilung) des Graphen G in zwei Knotenmengen A und B , sodass folgende Bedingungen erfüllt sind:
 - $|A| = |B|$: Die Anzahl der Knoten in A und B ist gleich.
 - $A \cap B = \emptyset$: Die Mengen A und B haben keine gemeinsamen Knoten (sie sind disjunkt).
 - $A \cup B = V$: Die Vereinigung von A und B ergibt alle Knoten des Graphen.
- **Minimiere:** Die Anzahl der Kanten zwischen A und B . Formal: $|\{(u, v) \in E \mid u \in A, v \in B\}|$. Das Ziel ist es, möglichst wenig Kanten zu haben, die die beiden Mengen verbinden.



Repräsentation der Lösung: Charakteristischer Vektor

Eine mögliche Lösung (also eine spezifische Aufteilung in A und B) wird durch einen charakteristischen Vektor $x = (x_1, \dots, x_n)$ repräsentiert, wobei $n = |V|$ die Gesamtzahl der Knoten ist.

- Jeder Eintrag x_i im Vektor entspricht einem Knoten i des Graphen.
- Wenn $x_i = 0$, wird Knoten i der Menge A zugewiesen.
- Wenn $x_i = 1$ (oder ein anderer Wert außer 0), wird Knoten i der Menge B zugewiesen.

Simulated Annealing (SA) für Graph-Bipartitionierung

Die Anwendung von Simulated Annealing auf dieses Problem beinhaltet folgende Schritte und Konzepte:

- **Nachbarschaft:** Um eine Nachbarlösung zu erzeugen (d.h. eine leicht veränderte Aufteilung), tauscht man jeweils einen Knoten von Menge A mit einem Knoten von Menge B aus. Dies erzeugt eine neue Partitionierung, die bewertet werden kann.
- **Zufällige Anfangslösung:** Der Algorithmus beginnt mit einer zufälligen Initialisierung, d.h., die Knoten werden zunächst zufällig auf die Mengen A und B aufgeteilt (wobei die Bedingung $|A| = |B|$ beachtet werden muss).
- **Geometrisches Abkühlen, $\alpha = 0.95$**
- **Iterationen auf jeder Temperaturstufe:** Auf jeder Temperaturstufe werden $n \cdot (n - 1)$ Iterationen durchgeführt.
- **Eine der ersten Anwendungen von SA:** Dies unterstreicht die historische Bedeutung der Graph-Bipartitionierung als frühes Anwendungsgebiet für Simulated Annealing.

Verbesserungen

Einschränkung der Nachbarschaft und Erlauben ungültiger Lösungen

- **Flip-Nachbarschaft (vgl. MAX-CUT):**

- Verschiebe einen einzelnen Knoten in andere Menge.
 - $|N(x)| = n$ anstatt $n^2/4$.
 - **Modifizierte Zielfunktion:**
 - $f(A, B) = |\{(u, v) \in E \mid u \in A, v \in B\}| + \gamma(|A| - |B|)^2$
 - γ : Faktor für das Ungleichgewicht (Strafterm für $|A| \neq |B|$).
-

Fazit zu Simulated Annealing

- Typischerweise einfach zu implementieren.
- Parametertuning notwendig, aber meist nicht so schwer.
- Für viele Probleme gute Resultate, aber häufig können ausgefeilte Methoden noch bessere Ergebnisse liefern.
- Viele Varianten/Erweiterungen:
 - Dynamische Strategien für das Abkühlen (Wiedererwärmen)
 - Parallelisierung
 - Kombination mit anderen **Methoden**

Tabu-Suche (TS)

⌚ Definition

- Basiert auf einem **Gedächtnis (History)** über den bisherigen Optimierungsverlauf.
 - Nutzt dieses Gedächtnis, um über lokale Optima hinwegzukommen.
- **Vermeidung von Zyklen** durch Verbieten des Wiederbesuchens früherer Lösungen.
- **I.A. (Im Allgemeinen) Best Improvement Schrittfunktion:**
 - In jedem Schritt wird die **beste erlaubte Nachbarlösung** angenommen.
 - Dies gilt auch, wenn diese schlechter ist als die aktuelle Lösung.

Variablen: Bisher beste gefundene Lösung x_{best} , Aktuelle Lösung x , Nachbarlösung x' , Tabu-Liste TL , Menge erlaubter Nachbarlösungen X' .

```
Tabu-Suche():
   $x_{\text{best}} \leftarrow x \leftarrow$  Ausgangslösung
   $TL \leftarrow \{x\}$ 
  while Abbruchkriterium nicht erfüllt
     $X' \leftarrow$  Teilmenge von  $N(x)$  unter Berücksichtigung von  $TL$ 
     $x' \leftarrow$  beste Lösung von  $X'$ 
    Füge  $x'$  zu  $TL$  hinzu
    Lösche Elemente aus  $TL$ , welche älter als  $t_L$  Iterationen sind
     $x \leftarrow x'$ 
    if  $x$  besser als  $x_{\text{best}}$ 
       $x_{\text{best}} \leftarrow x$ 
```

Das Gedächtnis: Tabuliste

📋 Eigenschaften

- **Explizites Speichern von vollständigen Lösungen:**
 - Nachteil: Speicher- und zeitaufwändig.
- **Meist bessere Alternative: Speichern von Tabuattributen.**
 - D.h., nur einzelne Aspekte von besuchten Lösungen werden gespeichert.
- Lösungen sind **tabu (verboten)**, falls sie Tabuattribute enthalten.
- Als Tabuattribute werden meist Variablenwerte benutzt, die von durchgeföhrten Zügen gesetzt wurden.
 - Die Umkehrung der Züge ist dann für t_L Iterationen verboten.
- **Wichtiger Parameter:** Tabulistenlänge t_L .

Parameter - Tabulistenlänge t_L

- Wahl von t_L ist häufig sehr kritisch!
 - Zu kurze Tabulisten können zu Zyklen führen (d.h., man kehrt zu schon besuchten Lösungen zurück).
 - Zu lange Tabulisten verbieten viele mögliche Lösungen und beschränken die Suche stark (dadurch wird der Suchraum zu stark eingeschränkt).
 - Geeignete Länge ist i.A. problemspezifisch.
 - Muss experimentell bestimmt werden, oder:
 - Immer zufällig neu wählen.
 - Adaptiv anpassen (\rightarrow Reactive Tabu Search).
-

Aspirationskriterien

- Manchmal wird eine Lösung verboten (d.h. ihre Attribute sind in der Tabuliste), obwohl sie sehr gut ist.
 - Aspirationskriterium: Überschreibt den Tabu-Status einer „interessanten“ Lösung.
 - D.h. die Lösung darf gewählt werden, obwohl sie eigentlich tabu wäre.
 - Beispiel eines oft benutzten Aspirationskriteriums:
 - Eine verbotene Lösung ist besser als die bisher beste gefundene Lösung.
-

Beispiel: Tabu-Suche für das Graphenfärbeproblem

Graphenfärbeproblem:

- Gegeben: Graph $G = (V, E)$.
- Gesucht: Weise jedem Knoten $v \in V$ eine Farbe $x_v \in \{1, \dots, k\}$ zu.
 - Sodass für alle Kanten $(u, v) \in E$ gilt: $x_u \neq x_v$ (benachbarte Knoten dürfen nicht die gleiche Farbe haben).
- Hinweis: Ist ein NP-vollständiges Problem.
- Optimierungsvariante: Minimiere Anzahl der „verletzten“ Kanten (Kanten, bei denen benachbarte Knoten dieselbe Farbe haben).

Aspekte der Tabu-Suche für das Graphenfärbeproblem:

- Evaluierungskriterium: Minimiere die Anzahl der „verletzten“ Kanten (Kanten, deren Endknoten dieselbe Farbe haben).
- Nachbarschaft: Eine Nachbarlösung ist eine Färbung, die sich genau in der Farbe eines Knotens unterscheidet (d.h., nur ein Knoten ändert seine Farbe).
- Tabuattribute: Paare (v, j) mit $v \in V, j \in \{1, \dots, k\}$.

- Dies bedeutet: Bestimmte Farbzueweisungen (j) zu bestimmten Knoten (v).
 - **Tabukriterium:** Wird ein Zug $(v, j) \rightarrow (v, j')$ (Knoten v ändert Farbe von j nach j') durchgeführt, ist das Attribut (v, j) für t_L Iterationen verboten.
 - Das bedeutet, dass es für t_L Iterationen verboten ist, dem Knoten v wieder die Farbe j zuzuweisen, die er *vor* dem aktuellen Zug hatte.
 - **Aspirationskriterium:** Falls ein Zug zu einer besseren Lösung als der bisher gefundenen führt, ignoriere den Tabu-Status und akzeptiere diese Lösung.
 - **Einschränkung der Nachbarschaft:** Betrachte nur Zuweisungen für Knoten, die in eine Kantenverletzung involviert sind.
 - D.h., es werden nur Knoten umgefärbt, die aktuell eine "verletzte" Kante verursachen, um die Effizienz der Suche zu erhöhen.
-

Fazit zur Tabu-Suche

- Viele weitere unterschiedliche Strategien für:
 - Gedächtnis (z.B. unterschiedliche Arten von Tabulisten oder Gedächtnisstrukturen).
 - Diversifizierung der Suche (Strategien, um den Suchraum breiter zu erkunden und nicht in lokalen Optima stecken zu bleiben).
 - Intensivierung in der Nähe gefundener Elitelösungen (Strategien, um gute gefundene Lösungen genauer zu untersuchen und zu verbessern).
- Oft exzellente Ergebnisse und vergleichsweise schnell.
- Meist relativ aufwändiges Fine-Tuning (Feinabstimmung der Parameter) notwendig.

Evolutionäre Algorithmen

Idee: Grundprinzipien der natürlichen **Evolution** werden primitiv nachgeahmt, um schwierige Optimierungsaufgaben zu lösen.

- **Population:** Eine Menge von aktuellen Kandidatenlösungen wird verwendet.
- **Selektion:** Durch natürliche Auslese ("survival of the fittest") bleiben bessere Lösungen mit höherer Wahrscheinlichkeit erhalten und erzeugen neue Lösungen.
- **Rekombination:** Neue Lösungen werden durch zufallsgesteuerte Kreuzung oder Vererbung von Lösungsmerkmalen aus den "Eltern"-Lösungen abgeleitet.
- **Mutation:** Kleine zufällige Änderungen fügen neue Lösungsmerkmale hinzu, die nicht in den Elternlösungen vorkamen. Dies ermöglicht eine Variation der Elternlösungen.

Prinzip eines evolutionären Algorithmus:

- Selektierte Eltern Q_s
- Zwischenlösungen Q_r

```

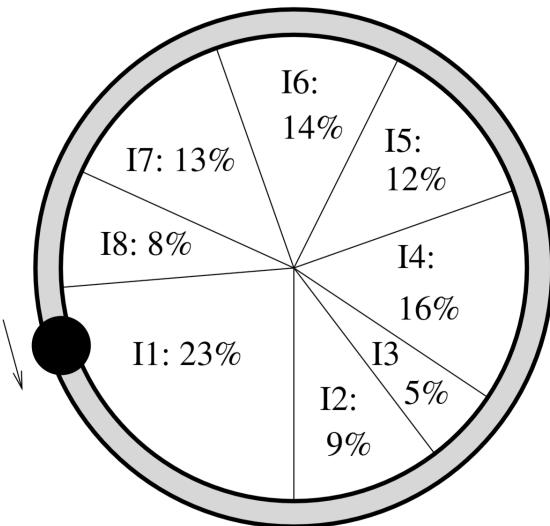
Evolutionär():
 $P \leftarrow$  Menge von Ausgangslösungen
Bewerte( $P$ )
while Abbruchkriterium nicht erfüllt
     $Q_s \leftarrow$  Selektion( $P$ )
     $Q_r \leftarrow$  Rekombination( $Q_s$ )
     $P \leftarrow$  Mutation( $Q_r$ )
    Bewerte( $P$ )

```

Fitness Proportional Selection

Roulette-Wheel Selection

- Sei $f(x_i) > 0$ der Zielfunktionswert (Fitness) jeder Lösung $x_i \in P$.
- $P = \{x_1, \dots, x_{|P|}\}$ repräsentiert die Population der Lösungen.
- Wir gehen von einem Maximierungsproblem aus.



Selektionswahrscheinlichkeit für Lösung x_i :

$$p_s(x_i) = \frac{f(x_i)}{\sum_{j=1}^n f(x_j)}$$

Selektionsdruck

Es ist wichtig, auf die Verhältnisse zwischen den Selektionswahrscheinlichkeiten der Lösungen in P zu achten.

Selektionsdruck:

Sei $p_s^{\max} = \max\{p_s(x_1), \dots, p_s(x_{|P|})\}$ die maximale Selektionswahrscheinlichkeit einer Lösung in der Population.

Sei $\bar{p}_s = 1/|P|$ die durchschnittliche Selektionswahrscheinlichkeit, wenn alle Lösungen gleich wahrscheinlich ausgewählt würden.

Dann ist der Selektionsdruck S definiert als:

$$S = p_s^{\max} / \bar{p}_s$$

- S zu niedrig: Ineffiziente Suche, da sie einer Zufallssuche ähnelt.
- S zu hoch: Rascher Verlust der Vielfalt in der Population, da einzelne (meist die besten) Lösungen zu häufig ausgewählt werden. Dies führt zu einer raschen Konvergenz zu einem lokalen Optimum.

Skalierung der Bewertungsfunktion

Skalierung: Um den Selektionsdruck S zu steuern, wird die Bewertungsfunktion meist skaliert, z.B. über eine lineare Funktion:

$$g(x_i) = a \cdot f(x_i) + b$$

mit geeigneten Werten a und b .

Hinweis: Skalierung ist auch notwendig für:

- Minimierungsprobleme (da die Roulette-Wheel Selection für Maximierungsprobleme ausgelegt ist und positive Fitnesswerte benötigt)
- Wenn $f(x_i) < 0$ (da die Fitness proportional Selection positive Fitnesswerte voraussetzt)

Alternative: Tournament Selektion

Alternative Vorgehensweise:

- (1) Wähle aus der Population k Lösungen gleichverteilt zufällig.
- (2) Die beste der k Lösungen ist die selektierte Lösung.

Eigenschaften:

- Relative Unterschiede in der Bewertung spielen keine Rolle. Es zählt nur, welche der k ausgewählten Lösungen die höchste Fitness hat.
 - Skalierung ist deshalb nicht erforderlich. (Im Gegensatz zur Fitness Proportional Selection)
 - Der Selektionsdruck wird über die Gruppengröße k gesteuert. Ein größeres k führt zu einem höheren Selektionsdruck, da die Wahrscheinlichkeit steigt, dass die "wirklich" beste Lösung der k zufällig ausgewählten auch tatsächlich die beste der gesamten Population ist und somit der Selektionsdruck steigt.
-

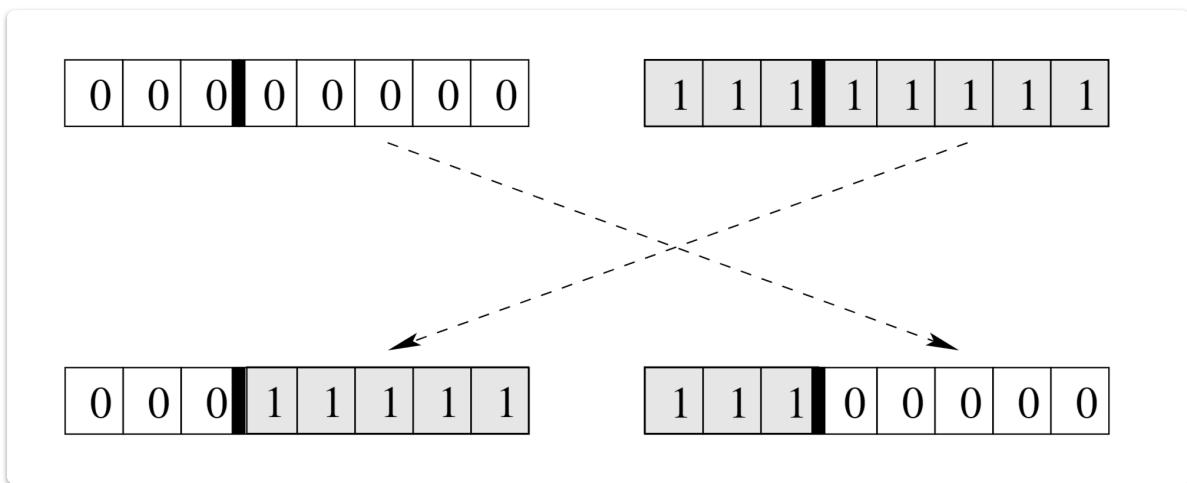
Rekombination

Rekombination: Aus zwei (oder mehreren) Elternlösungen wird eine neue Lösung abgeleitet.

Vererbung: Die neue Lösung sollte möglichst ausschließlich aus den Eigenschaften (Bestandteilen) der Eltern aufgebaut werden.

Die Rekombination ist meist eine zufallsbasierte, einfach gehaltene und schnelle Operation.

Beispiel für Bitstrings: 1-point crossover



Erklärung des 1-point crossover:

Beim 1-point crossover wird ein zufälliger "Schnittpunkt" innerhalb der Bitstrings der beiden Elternlösungen festgelegt. Alle Bits vor diesem Schnittpunkt stammen vom ersten Elternteil, und alle Bits nach dem Schnittpunkt stammen vom zweiten Elternteil, oder umgekehrt. Dies führt zur Erzeugung von zwei neuen Kindlösungen.

Mutation

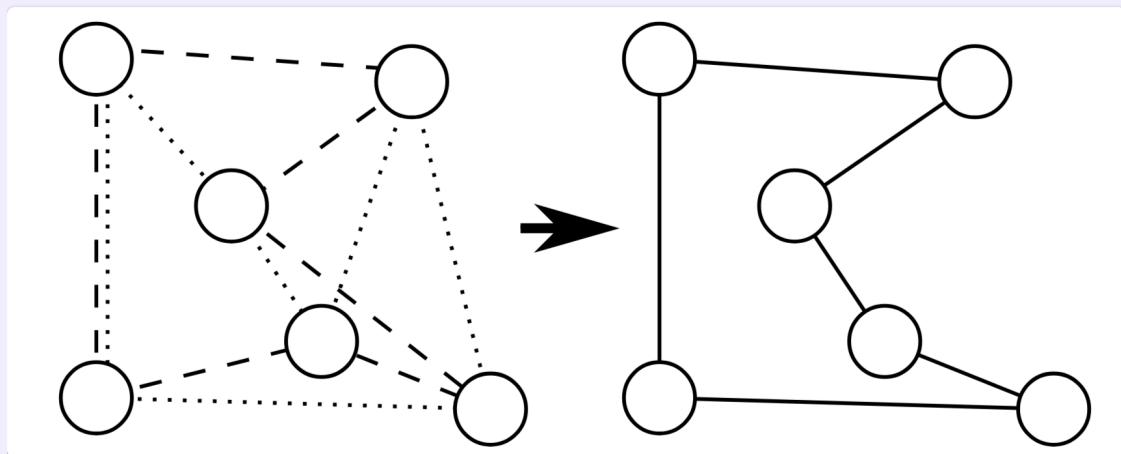
Möglichkeiten der Mutation:

- Für Bitstrings (eine Reihe von 0en und 1en) z.B. **Flip eines jeden Bits** mit kleiner Wahrscheinlichkeit. Das bedeutet, eine 0 wird zu einer 1 und eine 1 zu einer 0.
- Allgemein meist ein oder mehrere **zufällige Moves in einer sinnvollen Nachbarschaft**. Das bedeutet, dass eine kleine, zufällige Änderung an der Lösung vorgenommen wird, die aber immer noch im Bereich "sinnvoller" Lösungen liegt.

Hinweis: Vergleiche Random-Neighbor-Schrittfunktion der lokalen Suche. (Die Mutation ähnelt dem Prozess, bei dem in der lokalen Suche zufällig ein Nachbar einer aktuellen Lösung als nächste potenzielle Lösung ausgewählt wird.)

☰ Beispiel: Evolutionärer Algorithmus für TSP

Edge-Recombination: Eine neue TSP-Tour wird zufallsgesteuert möglichst nur aus Kanten aufgebaut, die bereits in zwei Elternlösungen vorkommen.



Mutation: Typischerweise ein zufälliger Move in einer klassischen Nachbarschaft wie z.B. 2-opt oder Verschieben eines Knotens an eine andere Position.

☰ Beispiel: Edge-Recombination für das TSP

Eingabe: Zwei gültige Touren T^1 und T^2 .

Ausgabe: Neue abgeleitete Tour T .

Variablen: Aktueller Knoten v , Nachfolgeknoten w , Kandidatenmenge für Nachfolgeknoten W .

Edge-Recombination(T^1, T^2):

Beginne bei einem beliebigen Startknoten $v \leftarrow v_0$, $T \leftarrow \{\}$

while es existieren noch unbesuchte Knoten

Sei W Menge noch unbesuchten Knoten, welche in

$T^1 \cup T^2$ adjazent zu v sind

if $W \neq \{\}$

Wähle einen Nachfolgeknoten $w \in W$ zufällig aus

else

Wähle einen zufälligen noch nicht besuchten Nachfolgeknoten w

$T \leftarrow T \cup \{(v, w)\}$, $v \leftarrow w$

Schließe die Tour: $T \leftarrow T \cup \{(v, v_0)\}$

Fazit zu evolutionären Algorithmen

- **Grundprinzip leicht umsetzbar.**
- **Häufig ist eine Kombination mit anderen Methoden sinnvoll:**
 - Ausgangslösungen mit Konstruktionsheuristiken erzeugen.
 - Problemspezifisches Wissen in Rekombination und Mutation ausnutzen.
 - Neue Kandidatenlösungen mit lokaler Suche etc. versuchen zu verbessern.
- Die **Lösungsgüte** und die **Laufzeit** hängen sehr stark von den konkreten Operatoren (z.B. der Art der Selektion, Rekombination und Mutation) ab.
- **Parallelisierung ist gut möglich**, da viele Operationen (z.B. Fitnessbewertung der Population, Generierung neuer Lösungen) unabhängig voneinander durchgeführt werden können.