Optimierung - Heuristische Verfahren

Algorithmen und Datenstrukturen
VU 186.866, 5.5h, 8 ECTS, 2023S
Letzte Änderung: 22. Juni 2023

Vorlesungsfolien

Optimierung: Roadmap

Branch-and-Bound

Dynamische Programmierung

Approximation(salgorithmen)

Heuristische Verfahren: Erzeuge in praktisch akzeptabler Zeit eine Näherungslösung. In der Praxis kann eine solche Lösung häufig sehr gut oder sogar optimal sein, es gibt aber keine Garantie dafür.

Heuristische Verfahren

Verfahren:

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

Konstruktionsverfahren

Eigenschaften:

- Meist sehr problemspezifisch.
- Häufig intuitiv gestaltet.
- Basiert oft auf dem *Greedy-Prinzip*.

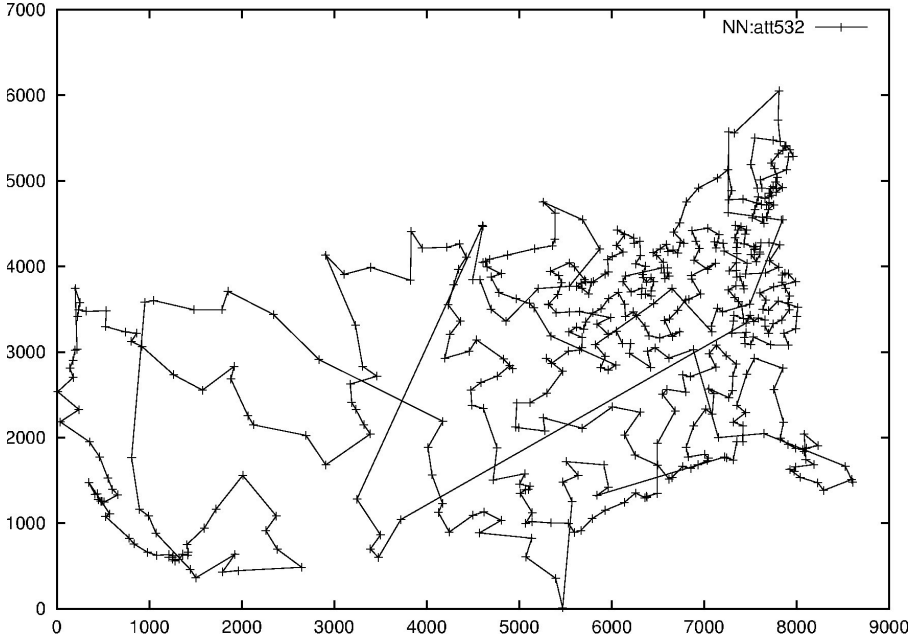
```
Greedy-construction-heuristic:
x ← leere Lösung
while x ist keine vollständige Lösung
    e ← die aktuell nützlichste Erweiterung von x
    x ← x ⊕ e
```

Beispiele für Konstruktionsverfahren

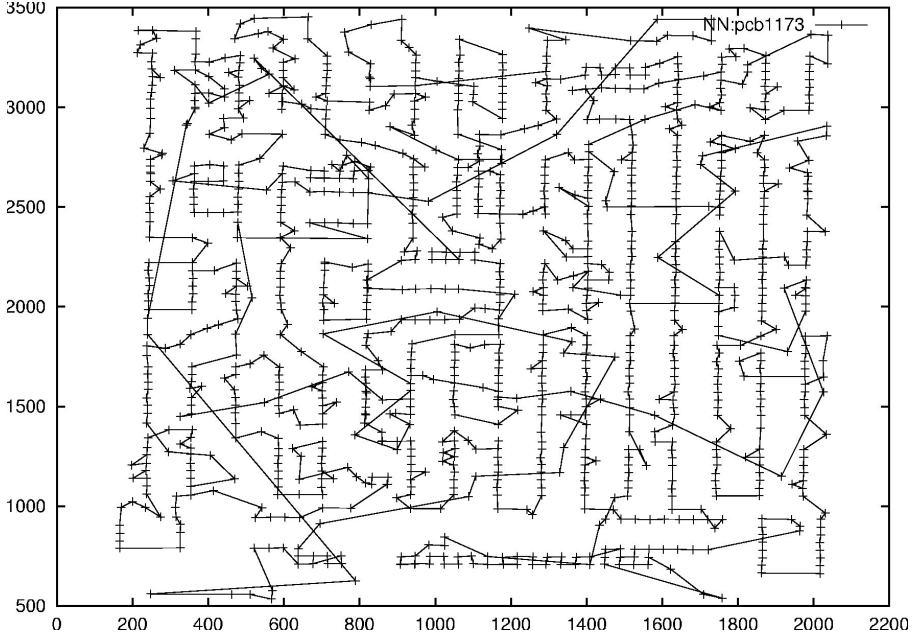
Wir haben in der VO bereits einige Konstruktionsverfahren kennengelernt.

- Minimaler Spannbaum: Greedy Algorithmen von Prim und Kruskal berechnen globales Optimum und sind daher exakte Algorithmen
- 0/1-Rucksackproblem: First-Fit-Heuristik berücksichtigt alle Gegenstände in nicht absteigendem Verhältnis von Wert zu Gewicht und packt jeden passenden Gegenstand ein.
- Vertex Cover Problem
- Lastverteilung: List Scheduling Algorithmus (Gütegarantie 2)
- TSP:
  - Spanning-Tree-Heuristik (Gütegarantie 2 für metrisches TSP)
  - Nearest-Neighbor-Heuristik: Starte bei einem Knoten und gehe immer zum nächsten nicht besuchten Nachbarn.

Beispiel: Nearest-Neighbor-Heuristik für das TSP (1)



Beispiel für Nearest-Neighbor-Heuristik (2)



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 Knoten, keine ist aber immer am besten, z.B.:

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

Strategien für das Einfügen des neuen Knoten:

- Nach dem nächsten Knoten in der Tour.
- Eine minimale Zunahme der Tourlänge verursachend.

Praktische Ergebnisse oft 10–20% über dem Optimum, aber auch für das metrische TSP keine konstante Gütegarantie!

Lokale Suche

Verbesserungsheuristiken: Lokale Suche

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

Idee: Versuche eine Ausgangslösung durch kleine Änderungen iterativ zu verbessern
$\rightarrow$ Lokale Suche

- i.A. keine Gütegarantien
- in der Praxis aber oft deutliche Verbesserung von Lösungen
- für die meisten Anwendungen akzeptable Laufzeiten

Lokale Suche

Prinzip:

```
x ← Ausgangslösung
while Abbruchkriterium nicht erfüllt
    Wähle x' ∈ N(x)
    if x' besser als x
        x ← x';
```

- $N(x)$: *Nachbarschaft von* x

Design einer Lokalen Suche

Komponenten, die für eine lokale Suchen wichtig sind:

- Lösungsrepräsentation
- Nachbarschaftsstruktur: Welche Lösungen werden von einer aktuellen ausgehend unmittelbar in Erwägung gezogen?
- Schrittfunktion: Wie wird die Nachbarschaft durchsucht?
- Terminierungskriterium

Nachbarschaftsstruktur

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

Nachbarschaft: $N(x)$ wird auch Nachbarschaft von x genannt.

Eigenschaften:

- Die Nachbarschaft ist üblicherweise implizit durch mögliche Veränderungen (Züge, Moves) definiert.
- Darstellung als Nachbarschaftsgraph möglich: Knoten entsprechen den Lösungen, eine Kante (u, v) existiert wenn $v \in N(u)$.
- Es gilt Größe der Nachbarschaft vs. Suchaufwand abzuwägen.

Lokale Suche: Vertex Cover

VERTEX COVER: 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).

Nachbarschaftsstruktur: $C' \in N(C)$ wenn C' aus C durch Löschen eines einzigen Knotens erzeugt werden kann und noch immer ein Vertex Cover ist. $N(C) = O(|V|)$.

Lokale Suche: Starte mit Vertex Cover C, z.B. $C = V$.
Wenn es ein $C' \in N(C)$ gibt, das ein gültiges Vertex Cover ist, so ist dieses immer eine bessere Lösung, da $|C'| = |C| - 1$.
Ersetze C durch ein solches C'.

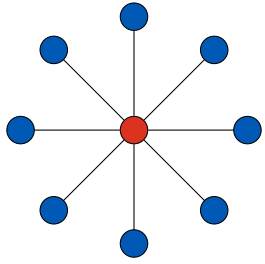
Diese Lokale Suche terminiert nach $O(|V|)$ Schritten.

Lokale Suche: Vertex Cover

Die lokale Suche liefert nicht immer eine optimale Lösung:

Lokales Optimum:
Kein Nachbar ist strikt besser und die lokale Suche kann die aktuelle Lösung daher nicht weiter verbessern.

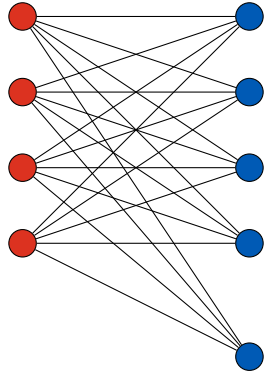
Beispiel 1:



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

Lokale Suche: Vertex Cover

Beispiel 2:



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

Lokale Suche: Vertex Cover

Beispiel 3:

Optimum: Alle geraden Knoten

Lokales Optimum: Jeder dritte Knoten wird ausgelassen.

Lokale vs. globale Optima

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

- Ein lokales Maximum in Bezug auf eine Nachbarschaftsstruktur N ist eine Lösung x für die gilt: $f(x) \geq f(x')$ für alle $x' \in N(x)$.
- Nachbarschaftsstruktur bestimmt welche Lösungen lokal optimal sind.

Verbesserungsmöglichkeiten:

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

Lokale Suche: Vertex Cover

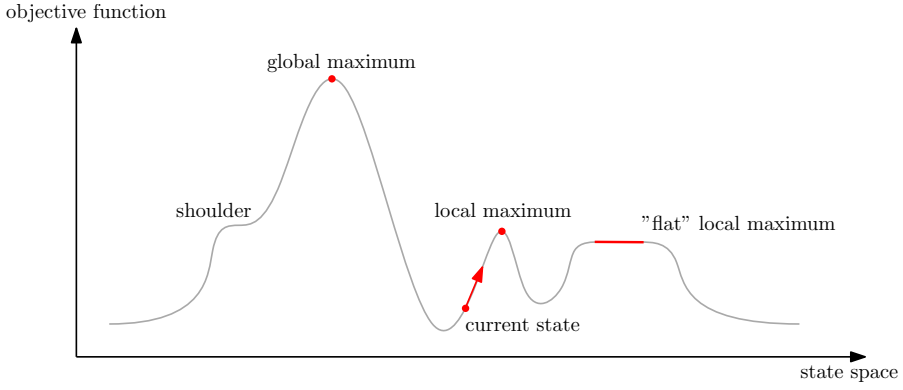
Alternative Nachbarschaft z.B.

- $N'(C)$ beinhaltet auch alle Knotenmengen, die durch Hinzufügen eines Knotens von $V \setminus C$ und Entfernen von zwei Knoten aus C gebildet werden können.
- $O(N'(C)) = O(|V|^3) \rightarrow$ Lokale Suche wird aufwändiger

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.

Abstrahierte, eindimensionale Vorstellung des Suchraums:



Lokale Suche: SAT

Gegeben: Eine boolesche Formel in konjunktiver Normalform mit Variablen $x_1, \ldots, x_n$, z.B.

$$(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$$

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.

k-flip Nachbarschaft für binäre Vektoren

- Nachbarlösungen haben eine Hamming-Distanz $\leq k$, d.h., sie unterscheiden sich in bis zu k Bits.
- Größe der Nachbarschaft: $O(n^k)$

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.
- Beispielsweise ist ein Durchlauf von Random Neighbor meist schneller, dafür benötigt Best Improvement oft erheblich weniger Iterationen.

Abbruchkriterium

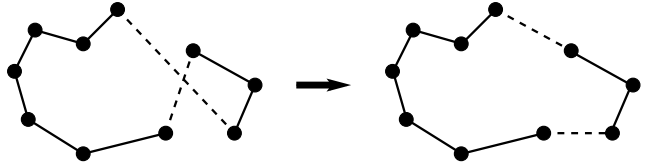
- Meist wird die Lokale Suche beendet, wenn ein lokales Optimum erreicht wurde.
- Bei einer Random Neighbor Schrittfunktion kann ein solches jedoch nicht direkt erkannt werden.
- Manchmal ist eine vollständige lokale Suche auch zu zeitaufwändig.

Alternativen:
Abbruch erfolgt

- 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.

Lokale Suche für das symmetrisches TSP

Nachbarschaftsstruktur: Austausch zweier Kanten
(„2-exchange“ oder „2-opt“)



Größe der Nachbarschaft: $O(n^2)$

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.

2-opt Lokale Suche für das symmetrische TSP

mit First-Improvement Schrittfunktion

Vorgehen:

(1) Sei $E(T) = \{(i_1, i_2), (i_2, i_3), \ldots, (i_n, i_1)\}$ die Menge der Kanten der aktuellen Tour T und sei $i_{n+1} = i_1$.

(2) Sei $Z = \{\{(i_p, i_{p+1}), (i_q, i_{q+1})\} \subset T \mid 1 \leq p, q \leq n \wedge p + 1 < q\}$
sei die Menge aller Paare nicht nebeneinanderliegender Kanten

(3) Für alle Kantenpaare $\{(i_p, i_{p+1}), (i_q, i_{q+1})\}$ aus Z:
Falls $c_{i_p i_{p+1}} + c_{i_q i_{q+1}} > c_{i_p i_q} + c_{i_{p+1}, i_{q+1}}$:
- $T = T \backslash \{(i_p, i_{p+1}), (i_q, i_{q+1})\} \cup \{(i_p, i_q), (i_{p+1}, i_{q+1})\}$
- gehe zu (2)

(4) retourniere T.

2-opt Lokale Suche für das symmetrische TSP

Laufzeitkomplexität: Da $|N(x)| = O(n^2)$ und jede Nachbarlösung in konstanter Zeit inkrementell evaluiert werden kann, 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)!
Die Worst-Case-Laufzeit dieser Lokalen Suche ist daher exponentiell!

Praxis: Dennoch ist das Verfahren auch auf großen Instanzen in meist sehr schnell. Starten wir mit einer sinnvollen Ausgangslösung 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 $T = \{(i_1, i_2), (i_2, i_3), \ldots, (i_n, i_1)\}$.

(2) Sei Z die Menge aller r-elementigen Teilmengen von T.

(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).

(4) T ist das Ergebnis.

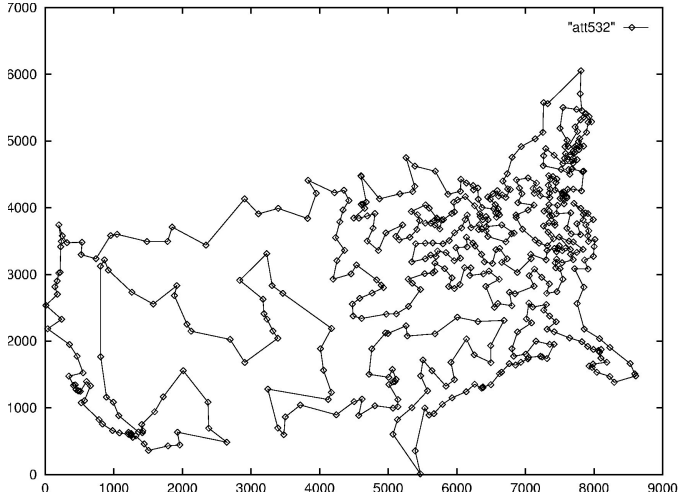
Laufzeitkomplexität der r-opt Lokalen Suche

Größe einer r-opt Nachbarschaft:

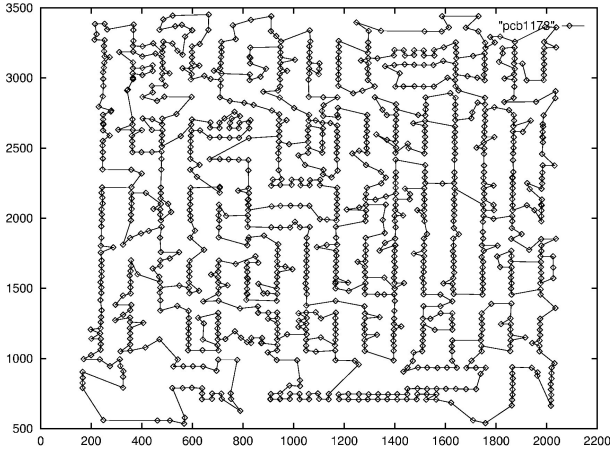
- Es gibt $\binom{n}{r} = O(n^r)$ Möglichkeiten r unterschiedliche Kanten aus einer aktuellen Tour zu entfernen.
- Das Entfernen führt zu r nicht zusammenhängenden Pfaden.
- Diese können auf $O(r!)$ Möglichkeiten zu neuen Touren zusammengefügt werden.
- $|N(x)| = O(n^r \cdot r!) = O(n^r)$

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 (1)



2-opt Lösung für TSP (2)



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 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.

Zusammenfassung zur Lokalen Suche für das TSP

Weitere Nachbarschaftsstrukturen:

- Verschieben eines Knotens an eine andere Position.
  - Für das asymmetrische TSP 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: Anzahl der ausgetauschten Kanten nicht grundsätzlich beschränkt, es werden jedoch nur „vielversprechende“ Kantenaustausche durchprobiert.

Maximaler Schnitt (*Maximal Cut*)

Maximaler Schnitt (MAX-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.

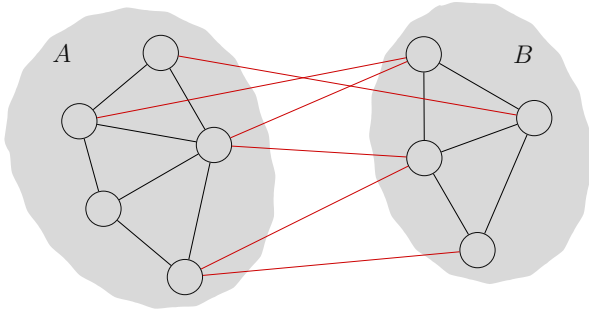
$$w(A, B) := \sum_{u \in A,\ v \in B} w_{uv}$$

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.

Hinweis: MAX-CUT ist NP-vollständig.

Maximaler Schnitt



1-Flip-Nachbarschaft: Gegeben sei eine Partition (A, B). Verschiebe einen Knoten von A nach B oder von B nach A.

Algorithmus: Ausgehend von einer gültigen Initiallösung ist auf die 1-Flip-Nachbarschaft aufbauend unmittelbar eine einfache Lokale Suche möglich.

Maximaler Schnitt: 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: 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*

Maximaler Schnitt: Analyse der lokalen Suche

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,

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

  $\square$ *Jede Kante wird einmal gezählt.*

Metaheuristiken

Metaheuristiken

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.

Wir betrachten hier folgende Metaheuristiken:

- Simulated Annealing
- Tabu-Suche
- Evolutionäre Algorithmen

Simulated Annealing (SA)

S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, *Science* 220(4598), pp. 671–680, 1983.

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

Simulated Annealing

Variablen:

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

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

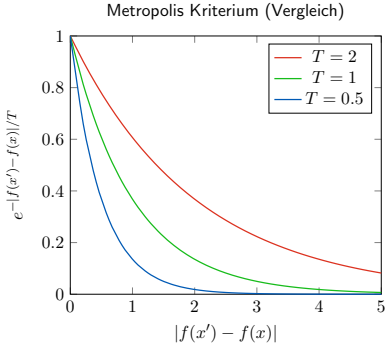
- *Metropolis-Kriterium*

Metropolis Kriterium

Metropolis Kriterium:

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

ist die Akzeptanzbedingung für schlechtere Lösungen.



Eigenschaften:

- Nur geringfügig schlechtere Lösungen werden mit höherer Wahrscheinlichkeit akzeptiert als viel schlechtere.
- Anfangs, bei hoher Temperatur T, werden schlechtere Lösungen mit größerer Wahrscheinlichkeit akzeptiert als im späteren Verlauf bei niedrigerer Temperatur.

Abkühlungsplan

Geometrisches Abkühlen:

- Faustregel für 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.
- $g(T, t) = T \cdot \alpha, \quad \alpha < 1$ (z.B. 0,999)
- Häufig wird die Temperatur auch über einige (z.B. $|N(x)|$) Iterationen gleich belassen und dann jeweils etwas stärker reduziert.

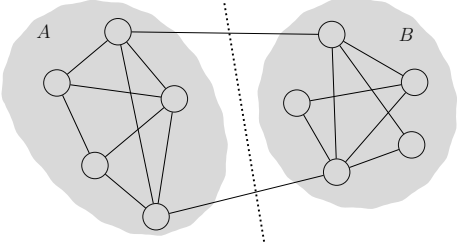
Adaptives Abkühlen: Es wird der Anteil der Verbesserungen an den letzten erzeugten Lösungen gemessen und auf Grund dessen T stärker oder schwächer reduziert.

Beispiel: SA für Graph-Bipartitionierung

Eine der ersten Anwendungen von SA

Definition für Graph-Bipartitionierung:

- Gegeben: Graph $G = (V, E)$
- Gesucht: Partitionierung von G in zwei Knotenmengen A, B, mit $|A| = |B|$, $A \cap B = \emptyset$, und $A \cup B = V$
- Minimiere $|\{(u, v) \in E \mid u \in A, v \in B\}|$



Beispiel: SA für Graph-Bipartitionierung

Repräsentation der Lösung: Charakteristischer Vektor

- $x = (x_1, \ldots, x_n)$, $n = |V|$
- Knoten i wird der Menge A zugewiesen, wenn $x_i = 0$ und B sonst.

Simulated Annealing:

- Nachbarschaft: Tausche jeweils einen Knoten von A und B aus.
- Zufällige Anfangslösung
- Geometrisches Abkühlen, $\alpha = 0.95$
- Iterationen auf jeder Temperaturstufe: $n \cdot (n - 1)$
- Eine der erste Anwendungen von SA

Beispiel: SA für Graph-Bipartitionierung

Verbesserung:

- Einschränkung der Nachbarschaft und Erlauben ungültiger Lösungen.
- Flip-Nachbarschaft (vgl. $\mathrm{MAX\text{-}CUT}$)
  - 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

Fazit zu Simulated Annealing

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

Tabu-Suche (TS)

- Basiert auf einem Gedächtnis (History) über den bisherigen Optimierungsverlauf und nutzt dieses um über lokale Optima hinweg zu kommen.
- Vermeidung von Zyklen durch Verbieten des Wiederbesuchens früherer Lösungen.
- I.A. Best Improvement Schrittfunktion: In jedem Schritt wird die beste erlaubte Nachbarlösung angenommen, auch wenn diese schlechter ist als die aktuelle.

Tabu-Suche

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

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

Das Gedächtnis: Tabuliste

Eigenschaften:

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

Parameter: Tabulistenlänge

Tabulistenlänge:

- Wahl von t_L häufig sehr kritisch!
- Zu kurze Tabulisten können zu Zyklen führen.
- Zu lange Tabulisten verbieten viele mögliche Lösungen und beschränken die Suche stark.
- Geeignete Länge i.A. problemspezifisch.
- Muss experimentell bestimmt werden, oder
  - immer zufällig neu wählen.
  - adaptiv anpassen ($\rightarrow$ *Reactive Tabu Search*).

Aspirationskriterien

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.
- Beispiel eines oft benutzten Aspirationskriteriums:
  - Eine verbotene Lösung ist besser als die bisher beste.

Beispiel: Tabu-Suche für das Graphenfärbeproblem

Graphenfärbeproblem:

- Gegeben: Graph $G = (V, E)$.
- Gesucht: Weise jedem Knoten $v \in V$ eine Farbe $x_v \in \{1, \ldots, k\}$ zu, sodass für alle Kanten $(u, v) \in E$ gilt $x_u \neq x_v$.

Hinweis: Ist ein NP-vollständiges Problem.

Optimierungsvariante: Minimiere Anzahl der „verletzten“ Kanten.

Beispiel: Tabu-Suche für das Graphenfärbeproblem

Aspekte:

- Evaluierungskriterium: Minimiere Anzahl der „verletzten“ Kanten
- Nachbarschaft: Färbung, die sich genau in der Farbe eines Knoten unterscheidet.
- Tabuattribute: Paare (v, j) mit $v \in V$, $j \in \{1, \ldots, k\}$, d.h. bestimmte Farbzuweisungen (j) zu bestimmten Knoten (v).
- Tabukriterium: Wird Zug $(v, j) \to (v, j')$ durchgeführt, ist Attribut (v, j) für t_L Iterationen verboten.
- Aspirationskriterium: Falls Zug zu besserer Lösung als bisher gefunden führt, ignoriere Tabu-Status und akzeptiere diese Lösung.
- Einschränkung der Nachbarschaft: Betrachte nur Zuweisungen für Knoten, die in eine Kantenverletzung involviert sind.

Fazit zur Tabu-Suche

- Viele weitere unterschiedliche Strategien für
  - Gedächtnis
  - Diversifizierung der Suche
  - Intensivierung in der Nähe gefundener Elitelösungen
- Oft exzellente Ergebnisse und vergleichsweise schnell.
- Meist relativ aufwändiges Fine-Tuning notwendig.

Evolutionäre Algorithmen

Idee: Grundprinzipien der natürlichen Evolution werden auf primitive Weise nachgeahmt, um schwierige Optimierungsaufgaben zu lösen.

- Population: Es wird mit einer Menge von aktuellen Kandidatenlösungen gearbeitet.
- Selektion: Natürliche Auslese („survival of the fittest“); bessere Lösungen bleiben mit größerer Wahrscheinlichkeit erhalten und erzeugen neue Lösungen.
- Rekombination: Neue Lösungen werden durch zufallsgesteuerte Kreuzung bzw. Vererbung von in Eltern vorkommenden Lösungsmerkmalen abgeleitet.
- Mutation: Kleine zufällige Änderung bringt nicht in Eltern vorkommende Lösungsmerkmale ein und dadurch ist eine Variation von Elternlösungen möglich.

Evolutionäre Algorithmen

Prinzip eines evolutionären Algorithmus:

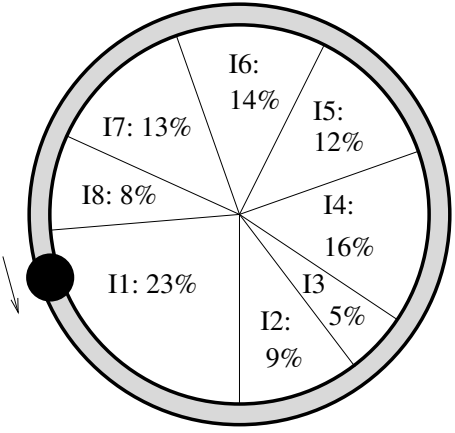
- Selektierte Eltern Q_s
- Zwischenlösungen Q_r

```
Evolutionär():
P ← Menge von Ausgangslösungen
Bewerte(P)
while Abbruchkriterium nicht erfüllt
    Q_s ← Selektion(P)
    Q_r ← Rekombination(Q_s)
    P ← Mutation(Q_r)
    Bewerte(P)
```

Fitness Proportional Selection

Roulette-Wheel Selection:

- Sei $f(x_i) > 0$ der Zielfunktionswert (Fitness) jeder Lösung $x_i \in P$.
- $P = \{x_1, \ldots, x_{|P|}\}$.
- 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

Zu achten ist auf die Verhältnisse zwischen den Selektionswahrscheinlichkeiten der Lösungen in P.

Selektionsdruck: Sei $p_s^{\max} = \max\{p_s(x_1), \ldots, p_s(x_{|P|})\}$ und $\overline{p_s} = 1/|P|$. Dann ist der Selektionsdruck:

$$S = p_s^{\max} / \overline{p_s}$$

- S zu niedrig: Ineffiziente Suche ähnelt einer Zufallssuche.
- S zu hoch: Rascher Verlust der Vielfalt da einzelne Lösungen zu häufig ausgewählt werden, rasche Konvergenz zu lokalem 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
- Wenn $f(x_i) < 0$

Alternative: Tournament Selektion

Alternative:

(1) Wähle aus der Population k Lösungen gleichverteilt zufällig.

(2) Die beste der k Lösungen ist die selektierte.

Eigenschaften:

- Relative Unterschiede in der Bewertung spielen keine Rolle.
- Skalierung deshalb nicht erforderlich.
- Selektionsdruck wird über Gruppengröße k gesteuert.

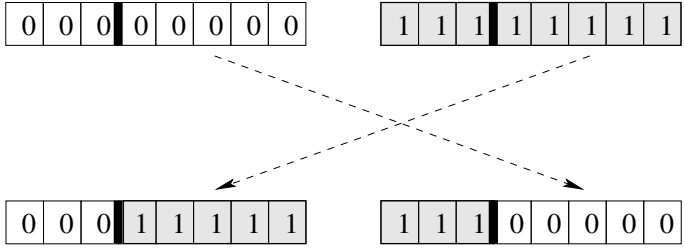
Rekombination

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.

Meist zufallsbasierte, einfach gehaltene und schnelle Operation.

Beispiel für Bitstrings: 1-point crossover



Mutation

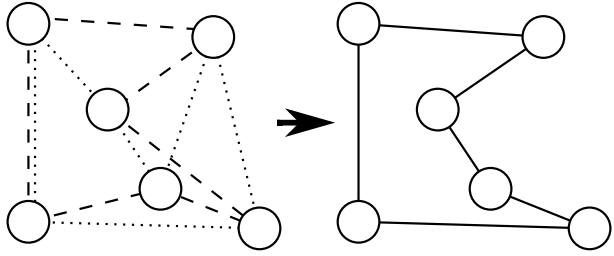
Möglichkeiten:

- Für Bitstrings z.B. Flip eines jeden Bits mit kleiner Wahrscheinlichkeit.
- Allgemein meist ein oder mehrere zufällige Moves in einer sinnvollen Nachbarschaft.

Hinweis: Vergleiche Random-Neighbor-Schrittfunktion der lokalen Suche.

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 ← v_0, T ← {}
while es existieren noch unbesuchte Knoten
    Sei W Menge noch unbesuchten Knoten, welche in
    T^1 ∪ T^2 adjazent zu v sind
    if W ≠ {}
        Wähle einen Nachfolgeknoten w ∈ W zufällig aus
    else
        Wähle einen zufälligen noch nicht
        besuchten Nachfolgeknoten w
    T ← T ∪ {(v, w)}, v ← w
Schließe die Tour: T ← T ∪ {(v, v_0)}
```

Fazit zu evolutionären Algorithmen

- Grundprinzip leicht umsetzbar.
- Häufig 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.
- Lösungsgüte und Laufzeit hängt sehr von den konkreten Operatoren ab.
- Parallelisierung ist gut möglich.