MAO - Zusammenfassung

Timo Bergerbusch

9. Februar 2018

Inhaltsverzeichnis

1 Typische Probleme

1.1 Matching

Paarbildung von adjazenten Knoten in einem ungerichteten bewerteten Graphen
Matching M: jeder Knoten hat max. einen Partner Knoten
perfektes Matching M^*: jeder Knoten hat genau einen Partner Knoten
Optional: Kosten, Profite, etc...

1.2 Set-Packing/Covering/Partitioning

Gegeben:

- m-elementige Menge $M = 1, \ldots, m$
- n Teilmengen $M_1, \ldots, M_n \subset M$
- Kosten/Profite der Teilmengen $c_1, \ldots, c_n$

Gesucht:

Auswahl von Teilmengen

1. mit max. Profit / min. Kosten
2. die Grundmenge M wird gepackt, überdeckt oder partitioniert

Eine Lösung L ist ein

- Set-Packing, wenn $M_j \cap M_k = \emptyset \forall j, k \in L, j \neq k$
- Set-Covering, wenn $\bigcup_{j \in L} M_j = M$
- Set-Partitioning, wenn es sowohl Set-Packing als auch Set-Covering ist

1.3 Traveling-Salesman-Problem (TSP)

Gegeben:

- vollständiger Graph $G = (V, E, d)$
- symmetrisch: $d_{i,j} = d_{j,i} \, \forall i, j \in V$
- asymmetrisch: $d_{i,j} \neq d_{j,i} \, \exists i, j \in V$

Gesucht:

kostenminimale Hamilton-Tour

LP:

$$\min z \quad = \quad \sum_{\{i,j\} \in E} d_{ij} x_{ij}$$

$$\begin{aligned}
\sum_{\{i,j\} \in \delta(i)} x_{ij} \; &= \; 2 && \forall i \in V, \\
\sum_{\{i,j\} \in E(S)} x_{ij} \; &\leq \; |S| - 1 && \forall S \subset V,\ 3 \leq |S| < |V| - 2, \\
x_{ij} \; &\in \; \{0,1\} && \forall \{i,j\} \in E.
\end{aligned}$$

1.3.1 Typische Distanzmatrixberechnung:

Euklidische Distanz: $d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$
Manhattan-Distanz: $d_{i,j} = |x_i - x_j| + |y_i + y_j|$

1.4 Bin-Packing-Problem (BPP)

sub

Gegeben:

- Kapazität $C > 0$
- Gewichte $w_1, \ldots, w_n$ mit $0 < w_i \leq C$

Gesucht:

Partitionierung $P_1 \cup \cdots \cup P_k = \{1, \ldots, n\}$ mit $P_i \cap P_j = \emptyset$ und $\sum_{j \in P_i} w_j \leq C$ für ein minimales k.

LP:

$$
\begin{aligned}
z_{BP} \quad &= \quad \min \sum_{j=1}^{n} y_j \\
\text{so dass} \quad & \sum_{i=1}^{n} w_i x_{ij} \leq C y_j \qquad \text{für alle } j \in \{1, \ldots, n\} \\
& \sum_{j=1}^{n} x_{ij} = 1 \qquad \text{für alle } i \in \{1, \ldots, n\} \\
& y_j \in \{0, 1\} \qquad \text{für alle } j \in \{1, \ldots, n\} \\
& x_{ij} \in \{0, 1\} \qquad \text{für alle } i, j \in \{1, \ldots, n\}
\end{aligned}
$$

Greedy-Algorithmus

Unterscheide die versch. Strategien

- Weight Decreasing: absteigend nach Gewicht sortiert
- Next-Fit: Zuordnung zum zuletzt geöffneten
- First-Fit: Zuordnung zumersten geöffneten Behälter mit kleinstem Index
- Best-Fit: Zuordnung zum Behälter mit kleinster noch ausreichender Kapazität

1.5 Rucksackproblem (KP)

Gegeben:

- Rucksack mit Kapazität C
- n Gegenstände mit Gewicht w_i und Profit p_i

Gesucht:

Profit-maximale Teilmenge von Gegenständen, welche nicht schwerer ist als C

LP:

$$z_{KP} = \max \sum_{j=1}^{n} p_j x_j$$

$$\text{so dass} \quad \sum_{j=1}^{n} w_j x_j \leq C$$

$$x_j \in \{0,1\} \quad \text{für alle } j = 1, \ldots, n$$

Greedy-Algorithmus

Algorithmus 1 : Greedy-Algorithmus für das Rucksack-Problem

```
// Input: absteigend sortierte Elemente i ∈ {1,...,n} nach
   relativem Profit p_i/w_i
SETZE C^Rest := C
for j = 1 bis n (gemäß Sortierung) do
    if w_j ≤ C^Rest then
        SETZE x̄_j := 1
        SETZE C^Rest := C^Rest − w_j
    else
        SETZE x̄_j := 0
// Output: (x̄_1, ..., x̄_n)
```

2 1. Kapitel - Einführung

- Abgrenzung Modell und Methode:

Modell: modellieren, darstellen, abbilden

Methode: rechnen, Algorithmus

- Beispielanwendungen:
  - Demand Planning: Welche Dienstleistungen auf welchen Flugstrecken
  - Umlaufplanung: Welches Flugzeug für welchen Flug?

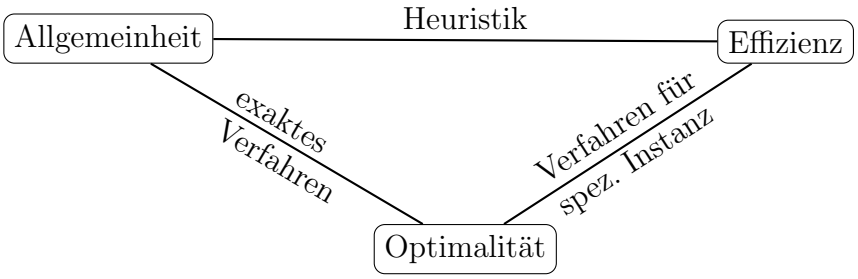
  - Crew Scheduling: Wer erledigt wann welche Aufgaben?
  - Disruption Management: Wie reagieren auf Störungen, Abweichungen und Ausfälle?
  - Revenue Management: Pricing, Kapazitätssteuerung
  - Netzwerk-Design: Welche Standorte und welche Aufgaben dort?
  - Transportplanung
  - Standort Planung
  - "Letzte Meile": Bezirkseinteilung und Postbotenrouten

- **optimal** : zulässig und bestmöglich
- **Simulation** : Durchführung von Experimenten anhand von Modellen
- Optimierung:
  - exakt vs. heuristisch
  - kontinuierlich vs. diskret
  - linear vs. nicht-linear
- Simulation vs. Optimierung:
  - Simulation zeigt nur Konsequenzen von Entscheidungen auf
  - Simulation macht keinen Entscheidungsvorschlag
  - Simulation gestattet nur den Vergleich von Alternativen
- **Heuristik** :

Def.: Algorithmen die ein geg. Optimierungsproblem mit akzeptablem Aufwand in möglichst gut zu lösen versuchen

Kennzeichen: Ausschluss potentieller Lösungen, fehlende Lösungsgarantie, nicht-willkürliche Lösungssuche(, künstliche Stoppregel)

Prinzipien Allgemeinheit, Effizienz, Optimalität



Klassifikation nach Stopp-Regel:

Eröffnungsv. generieren einer (mögl. guten) Lösung. Terminierung sobald Lösung gefunden

Verbesserungsv. Start mit zulässiger Lösung und verbessern, bis Stopp-Krit. erfüllt ist

Zsm.-gesezte V. Kombination von Eröffnungs- und Verbesserungsv.

- **Algorithmus** :

  Def.: Ein Algorithmus ist eine genau definierte Verarbeitungsvorschrift zur Lösung eines Problems oder einer bestimmten Art von Problemen. Typischerweise wird ein Algorithmus durch eine endliche Folge von Anweisungen beschrieben, die nacheinander ausgeführt und oft in festgelegter Weise wiederholt werden

  Eig.: eindeutig, allgemein, ausführbar, endlich

- **exakter Algorithmus** : ein Algorithmus, welcher für jede Instanz eines Optimierungsproblems ein Optimum bestimmt

- **Laufzeitkomplexität**: Alg. A ist in $\mathcal{O}(f(n))$, gdw $\exists k_1, k_2$ konst., sodass $\text{time}_A(P) \leq k_1 + k_2 f(n)$ für alle Instanzen P mit $|P| = n$

- **effizienter Algorithmus** : A ist effizient gdw $A \in \mathcal{O}(f(n))$

- $\mathcal{P}$: alle Probleme mit einem deterministischen effizienten Algorithmus.

- $\mathcal{NP}$: alle Probleme mit einem nicht-deterministischen effizienten Algorithmus.

- **polynomiale Reduzierbarkeit**: Geg.: Probleme Π_1, Π_2, Funktionen $f : \Pi_2 \to \Pi_1$ und $g : L(\Pi_1) \to L(\Pi_2)$
  Π_2 ist polynomial reduzierbar auf Π_1, gdw. f und g polynomial berechenbar sind
- $\mathcal{NP}$**-schwer** : falls jedes $\Pi' \in \mathcal{NP}$ pol. red. auf Π ist
- $\mathcal{NP}$**-Vollständig** : falls Π $\mathcal{NP}$-schwer und $\Pi \in \mathcal{NP}$

3 2. Kapitel - Greedy Algorithmen

- **Greedy-Algorithmus** : Greedy = gierig; Greedy-Prinzip: fixiere Variable die jetzt die größte Verbesserung darstellt

---

SETZE $J^{fix} := \varnothing$ und $J^{frei} := J$
repeat
  Löse das Hilfsproblem P_j für alle $j \in J^{frei}$.
  Es sei $j^* \in J^{frei}$ der Index der nächsten zu fixierenden Variablen
  SETZE x_j^* auf den im Hilfsproblem P_{j^*} ermittelten Wert $x_{j^*} := \bar{x}_{j^*}$
  SETZE $J^{fix} := J^{fix} \cup \{j^*\}$ und $J^{frei} := J^{frei} \setminus \{j^*\}$
  if *zusätzliche Variablen aus* J^{frei} *können fixiert werden* **then**
    Fixiere diese und aktualisiere J^{frei} und J^{fix}
until $J^{fix} = J$

- Eröffnungsverfahren
- BPP-Greedy und KP-Greedy
- MST-Greedy Algorithmus:

Algorithmus 2 : Algorithmus von Kruskal

```
// Input: sortierte Kanten E nach steigenden Kosten
SETZE T = ∅
repeat
    Wähle nächste Kante {i,j} ∈ E gemäß der Sortierung
    if Hinzufügen von {i,j} zu T erzeugt keinen Kreis then
        Füge {i,j} zu T hinzu.
until alle Kanten wurden durchlaufen
// Output: MST (V,T)
```

- meinst nicht optimal
- Nutzenmöglichkeit:
  1. unzulässige Lösungen zulassen und mit Strafkosten versehen
  2. Wiederholen mit mod. Kosten/Koeffizienten/(Ersatz-)Problem

4 3. Kapitel - Lösungsqualität und Approximation

- **Performance-Verhältnis** : Probleminstanz $P \in \Pi$, Heuristik $H \Rightarrow R_H(P) = \frac{z_H(P)}{z_{opt}(P)}$
- wenn Performance Verhältnis = 1 für alle Instanzen von $P \Rightarrow$ exakter Opt.-Algo.
  - Minimierungsproblem: $R_H(P) \geq 1$
  - Maximierungsproblem: $R_H(P) \leq 1$
- **Performance Analysen**:

worst-case: schlechtestes Performance Verhältnis

average-case: durchschnittliches Performance Verhältnis über alle Instanzen, via Wahrscheinlichkeitsverteilung, Erwartungswert und Varianz

empirisch: durchschnittliches Performance Verhältnis, maximale Abweichung und empirische Varianz über relevante Instanzen

- **ϵ-Approximationsalgorithmus**: Für $\epsilon \geq 0$, Heuristik H. H ist ϵ-Approximationsalgorithmus gdw.
  - $R_H \leq 1 + \epsilon$, für Minimierungsp.
  - $R_H \geq 1 - \epsilon$, für Maximierungsp.

5 4. Kapitel - Lokale Suche

- Grundidee: geg. zulässige Lösung in eine andere bessere zulässige Lösung transformieren und rekursiv wiederholen
- **Nachbarschaft** : Abbildung $\mathcal{N} : X \to Pot(X)$
- **verbessernder Nachbar** : zu min. Ziel-fkt z: $x' \in \mathcal{N}(X)$ mit $z(x') < z(x)$ heißt verbessernder Nachbar
- Nachbarschaften, Nachbarn
- Pseudocode:

Algorithmus 1 : Lokale Suche

```
// Input: Lösung x und Nachbarschaft N
SETZE t := 0 und x^0 := x
repeat
    Durchsuche die Nachbarschaft N(x^t) nach einem verbessernden
      Nachbarn x' ∈ N(x^t)
    if verbessernder Nachbar x' gefunden then
        SETZE x^{t+1} := x' und t := t + 1
until kein verbessernder Nachbar gefunden
// Output: lokal optimale Lösung x^t
```

- Suchstrategien:

Erstensuche: suche ersten verbessernden Nachbarn

Bestensuche: suche besten Nachbarn

l-Erstensuche: suche ersten l verbessernde Nachbarn

- 2-Opt Nachbarschaft:
  - Für (S)TSP: Entfernen von 2 nicht benachbarten Kanten und hinzufügen von zwei anderen
  - n-Städte: $|\mathcal{N}_{2Opt}(X)| = \frac{n(n-3)}{2}$
- **Nachbarschaftsgraph** : $(X, A_{\mathcal{N}})$ mit X alle zulässigen Lösungen, $(x, x') \in A_{\mathcal{N}}$, falls $x' \in \mathcal{N}(x)$
- **stark Zusammenhängend** : Falls von jeder Lösung $x, y \in X$ ein Weg π ex., sodass $\pi = x \dots y$
- **Transitionsgraph** : $(X, A_{\mathcal{N}}^{trans})$ mit X alle zulässigen Lösungen, $(x, x') \in A_{\mathcal{N}}$, falls x' ein verbessernder Nachbar von x ist
- **exakte Nachbarschaft**: Wenn jedes lokale Optimum auch ein globales Optimum ist
- **Durchmesser einer Nachbarschaft**: max. Länge eines kürzesten Weges
- Wünschenswerte Eigenschaften:
  1. starker Zusammenhang
  2. Symmetrie: $x \in \mathcal{N}(x') \Leftrightarrow x \in \mathcal{N}(x')$
  3. Effiziente Berechnung des Zielfunktionswertes.
  4. Effiziente Konstruktion von $\mathcal{N}$
  5. Effiziente Zulässigkeitsprüfung
  6. Effiziente Suche nach verbessernden Nachbarn
- **Sequentielle Suche**:

  Idee: Lösche $(x, x') \rightarrow$ Füge (x', x'') hinzu $\rightarrow$ Lösche $(x'', x''') \rightarrow \dots$

  In 2-Opt: t_1, t_2, t_3, t_4, wobei (t_1, t_2) und (t_3, t_4) gelöscht und (t_2, t_3) und (t_1, t_4) hinzu gefügt wurden

  $\Rightarrow$ für $g_1 = c_{t_1,t_2} - c_{t_2,t_3}$ und $g_2 = c_{t_3,t_4} - c_{t_4,t_1}$ will man $g_1 + g_2 > 0$ für Verbesserung

$\Rightarrow$ man genötigt $g_1 > 0$ und $g_1 + g_2 > 0$ (einer muss > 0 sein und wenn $g_1 + g_2 > 0$ und $g_1 \leq 0$ dann stimmt die Aussage für $g_1' = g_2$ und $g_2' = g_1$)

– Algorithmus:

Algorithmus 2 : 2-Opt-Bestensuche

```
// Input: Tour x = (x_1, x_2 ..., x_n, x_1)
SETZE G* := 0
for i_1 = 1, ..., n und s ∈ {−1, +1} do
    SETZE t_1 := x_{i_1}, t_2 := x_{i_1+s}
    SETZE Bound := c_{t_1,t_2} − G*/2
    for t_3 ∈ N(t_2) mit c_{t_2,t_3} < Bound do
        SETZE i_2 := Position(t_3) und t_4 := x_{i_2−s}
        SETZE G := c_{t1,t2} − c_{t2,t3} + c_{t3,t4} − c_{t4,t1}
        if G > G* then
            SETZE G* := G
            SETZE t* := (t_1, t_2, t_3, t_4)
// Output: Gewinn G* und für G* > 0 die
   Knotenkombination t*
```

(Wichtig: `bound` fängt den symmetrischen Fall ab und bedeutet soviel wie: wir müssen bereits mit der ersten Löschung min. die Hälfte der Einsparung machen)

6 5. Kapitel - Metaheuristiken

6.1 Einführung

- Anwendungsbereich: exakte Verfahren nicht mehr anwendbar
- Eigenschaften: keine Optimalitätsgarantie, keine Aussage über Abweichung, akzeptable Lösungsgüte bei vernünftiger Laufzeit
- vom lokalen zum globalen Optima, via:
  1. Multi-Start
  2. Gedächtnis (Tabusuche)
  3. Randomisierung (Simulated Annealing)

4. Pertubation (ILS, VND)

- "Def.": Strategien zum Steuern des Suchprozesses; unabhängig vom Problem; Konkretisierung ⇒ Heuristik für spez. Optimierungsproblem
- Klassifikation ähnlich wie hier mit Erweiterung:
  - Zufall: determinitistisch vs. zufallsgesteuert
  - Gedächtnis: mit vs. ohne
  - natur-analog: der Natur nachempfunden vs. künstlich
  - Anzahl Lösungen: Nutzung einer Lösung vs. Population von Lösungen
- Alternieren zwischen: Intensivierung und Diversifikation
- Effiziente Datenstrukturen/Subalgorithmen wichtiger Bestandteil
- Charakteristika:

Geschwindigkeit:
- ∗ hängt von Planungsebene ab: strategisch vs. taktisch vs. operativ
- ∗ Echtzeitumgebung, dynamische Planung
- ∗ Interaktion

Genauigkeit:
- ∗ Abweichung von Optimallösung
- ∗ Konsistenz: lieber immer (nur) gut als mal sehr gut mal sehr schlecht

Problem: Lösung durch genaues hinsehen schon verbesserbar
- ∗ Lieber gute Lösung + stetige Verbesserung sichtbar, als einfach finale Lösung

Einfachheit:
- ∗ Verständlichkeit von der Grundidee
- ∗ robust
- ∗ vernünftige Anzahl aussagekräftiger Parameter

Flexibilität:
- ∗ Anpassung durch weitere NB verringern nicht/kaum die Güte

6.2 Iterierte Lokale Suche (ILS)

6.2.1 Idee

Suche auf der Menge der lokalen Minima.

6.2.2 Algorithmus

Algorithmus 1 : Iterierte Lokale Suche (ILS) für ein Minimierungsproblem

```
// Input: Startlösung x
SETZE x* := LokaleSuche(x)
repeat
    SETZE x' := Perturbation(x*, Historie)
    SETZE x'* := LokaleSuche(x')
    SETZE x* := AkzeptanzKriterium(x*, x'*, Historie)
until Terminierungsbedingung(Historie)
// Output: beste gefundene Lösung
```

1. `LokaleSuche(`x`)`: Verbesserung einer Lösung x zu einem lokalen Optimum
2. `Pertubation(`x`,historie)`: verwirbelt eine Lösung
3. `AkzeptanzKriterium(`x`,`x'`,historie)`: akzeptiert x' oder lässt x

6.2.3 Stärken

- Gute Lösungsqualität mit einfacher Impl. (besser als zufällige Neustarts)
- konzeptuell einfach und modularer Aufbau
- Geschwindigkeit: besser als zufällige Neustarts

6.2.4 Design

- **Initiallösung**: Einfluss nimmt mit Länge des Suchverlaufs ab
- **Lokale Suche**: gründliche LS liefert meist bessere Ergebnisse
- **Pertubation**:

  zu schwach: wenig Diversifikation

  zu stark: quasi zufällige Neustarts

– Abstimmung mit LS: Güte und Geschwindigkeit

- **Akzeptanzkriterium** Oft basierend auf SA

6.3 Variable Neighborhood Descent (VND)

6.3.1 Idee

Det. LS mit einer Folge von Nachbarschaften. Nur verbessernde Schritte und ende in lokalem Optimum bzgl. aller Nachbarschaften

6.3.2 Algorithmus

Algorithmus 2 : VND (für ein Minimierungsproblem)

```
// Input: Zulässige Lösung x^0
SETZE t := 0 (Iterationszähler) und k := 1 (Nachbarschaftszähler)
repeat
    repeat
        Durchsuche Nachbarschaft N_k(x^t) nach (verbesserndem) Nachbarn
          x' ∈ N_k(x^t) mit c(x') < c(x^t)
        if verbessernder Nachbar x' gefunden then
            SETZE x^{t+1} := x'
            SETZE t := t + 1 und k := 1
    until keine Verbesserung mehr gefunden
    SETZE k := k + 1
until k > k_max
// Output: lokales Optimum x^t bzgl. aller Nachbarschaften
```

$k_{max} = 1$ einfache ITS

- Bei Maximierung ersetze ≪"durch ≫"
- Wenn bessere Lösung x'' gefunden wurde starte wieder bei $\mathcal{N}_1$
- Auswahl $x \in \mathcal{N}_k(x^t)$ sollte aufgrund einer Gleichverteilung erfolgen

6.4 Variable Neighborhood Search (VNS)

6.4.1 Idee

Det. Wahl einer Nachbarschaft $\mathcal{N}_1, \ldots, \mathcal{N}_k$ und zufällige Wahl einer Nachbarlösung in dieser. Anschließende LS und dann Akzeptanzkriterium.

6.4.2 Algorithmus

Algorithmus 3 : VNS (für ein Minimierungsproblem)

```
// Input: Zulässige Lösung x^0
SETZE t := 0 (Iterationszähler) und k := 1 (Nachbarschaftszähler)
repeat
    (Schütteln) Wähle zufällig eine Nachbarlösung x' ∈ N_k(x^t)
    (Lokale Suche, optional) x'' = LokaleSuche(x') mittels Nachbarschaft N
    (Akzeptanzkriterium)
    if c(x'') < c(x^t) then
        SETZE x^{t+1} = x'', t := t + 1 und k := 1
    else
        SETZE k := { k + 1   k < k_max
                   { 1       k = k_max
until Stoppkriterium erfüllt
// Output: beste gefundene Lösung x^t
```

- Zyklisches durchlaufen der Nachbarschaften
- Sobald besseres x'' gefunden starte wieder bei $\mathcal{N}_1$
- Auswahl von $x \in \mathcal{N}_k(x^t)$ gleich verteilt zufällig

6.5 Tabu Search

6.5.1 Idee

Gehe zur zur besten Nachbarlösung, welche mit den Tabu-Restriktionen vereinbar ist.

6.5.2 attributives vs. explizites Gedächtnis

Name	attributiv	explizit
Speichern von:	einzelne Attribute einer Lösung(-sübergangs)	komplette Lösungen
Typische Elemente:	• Vorhanden sein eines Obj. in einer Lösung • Anzahl von Objekten	Verwaltung von Elite-Lös.
	• Austausch von Objekten	test
Vorteil:	geringer Speicheraufwand	kein Tabu für nicht besuchte Lösungen
Nachteil:	mögl. tabu für nicht besuchte Lösung	Speicheraufwendig

6.5.3 Tabu Restriktionen

Nr.	Basis-Attribut	Tabu-Restriktion
1	`Inc(i)`	Verbietet Verringern von x_i auf 0
2	`Dec(j)`	Verbiete Erhöhen von x_j auf 1
3	`Inc(i)` und `Dec(j)`	Verbietet Nr. 1 und/oder 2
4	`Swap(+i,-j)`	Verbietet Umkehrung `Swap(+j,-i)`
5	`c(`x'`)-c(`x`)`	Verbietet mit Ziel-fkt Änderung `c(`x`)-c(`x'`)`
6	`g(`x'`)-g(`x`)`	Verbietet Schritte mit Fkt Änderung `g(`x`)-g(`x'`)`

- restriktivere Tabu-Restriktionen tabuisieren mehr Nachbarlösungen
- meist Versucht Inversion vom aktuellen zum vorherigen zu vermeiden
- Gedächtnis genutzt um Zyklen zu vermeiden

6.5.4 Kurzzeitgedächtnis

- 1. Füge neue Restriktionen in eine Liste der Länge k ein
  2. Sind bereits alle k Elemente genutzt lösche das "ältesteünd füge die neuen hinzu
- zu kleines $k \rightarrow$ höheres Zyklen-Risiko
- zu großes $k \rightarrow$ zu starke Einschränkung

6.5.5 Anspruchskriterium

Wenn alle Nachbarlösungen tabu $\rightarrow$ wähle eine, welche tabu ist

	Anspruchskriterium	Bedeutung
•	Anspruch bei Ermangelung	falls alle tabu, wähle die, die "am wenigsten"tabu ist
•	Anspruch durch Ziel-funktionswert	falls der Zielfunktionswert besser ist als der bisher beste Wert ist die Lösung nicht mehr tabu.
•	Anspruch durch Suchrichtung	falls Restriktion gesetzt während Verbesserung und tabuisierte erneut Verbesserung → hebe Restriktion auf

6.5.6 Algorithmus

Algorithmus 1 : Tabu-Search mit Kurzzeitgedächtnis

```
// Input: Zulässige Lösung x^0
SETZE den Iterationszähler t := 0.
repeat
    // (Bestensuche)
    Durchsuche die gesamte Nachbarschaft N^t(x^t) und wähle eine
      bezüglich einer Bewertungsfunktion beste Lösung x' ∈ N^t(x^t), die
      nicht tabu ist oder tabu ist und ein Anspruchskriterium erfüllt.
    // Iteration
    SETZE x^{t+1} := x' und t := t + 1
    Aktualisiere das Kurzzeitgedächtnis.
until ein Stoppkriterium ist erfüllt
// Output: Lösung x^t bzw. beste gefunden Lösung x^best
```

- als Lokale Suche meist Bestensuche → aggressive Suche
- Nachbarschaft kann im Laufe variiert werden
- beste Lösung immer explizit speichern
- Bewertungsfunktion kann von Zfkt. abweichen (Strafkosten, Einfluss auf Struktur)

6.6 Large Neighboorhood Search (LNS)

Alternative Namen: Ruin and Recreate, Fix and Optimize

6.6.1 Idee

Graduelle Verbesserung einer Lösung durch Destroy- (entfernen eines Teils) und Repair- (bauen entfernte anders wieder auf) Operatoren.

6.6.2 Algorithmus

Algorithmus 1 : Large Neighborhood Search

```
// Input: Zulässige Lösung x
SETZE x^b = x;
repeat
    SETZE x^t = r(d(x))
    if akzeptiert (x^t, x) then
        SETZE x = x^t
    if c(x^t) < c(x^b) then
        SETZE x^b = x^t
until Abbruchkriterium erfüllt ist
return x^b
// Output: Beste gefundene Lösung
```

- Zufall: versch. Teile der Lösung zerstören → Diversifikation
- Zerstörungsgrad:
  - zu gering → keine Exploration des Lösungsraums
  - zu hoch → zeitaufwendig und wenig erfolgversprechend

6.6.3 Destroy-Operatoren

Name	Beschreibung	Effekt
Random Destroy	Entferen zufällige Elemente	Diversifikation
Worst/Critial Destroy	Entferne teuersten Elemente	Intensivierung
Related Destroy	Entferne leicht austauschbare Elemente	Intensivierung
Small Removal	Entferne Variablen mit geringen Koeff.	Intensivierung
History-Based Destroy	Entferne Elemente die Teil einer schlechter Lösungen waren	Diversifikation

6.6.4 Repair-Operatoren

- Greedy Heuristiken: iterativ bestmögliche Wahl
  Problem: Wahlmöglichkeiten in späteren Iterationen
- Regret Heuristiken: maximale Differenz zwischen bester Einfügemöglichkeit und k-bester
- Relaxierte exakte Verfahren: Beschleunigung
- Exakte Verfahren: schnell gute Lösungen, keine Diversifikation
- Lokale Suche

6.7 Adaptive Neighborhood Search (ALNS)

6.7.1 Idee

Anwenden von mehreren gewichteten Destroy- und Repair-Operatoren, mit dynamischer Gewichtsanpassung basierend auf vorherigen Iterationen

6.7.2 Algorithmus

Algorithmus 2 : Adaptive Large Neighborhood Search

```
// Input: Zulässige Lösung x
SETZE x^b = x; ρ^- = (1, ..., 1); ρ^+ = (1, ..., 1);
repeat
    WÄHLE Destroy- und Repair-Operatoren d ∈ Ω^- und r ∈ Ω^+ unter
      Verwendung von ρ^- und ρ^+
    SETZE x^t = r(d(x))
    if akzeptiert (x^t, x) then
        SETZE x = x^t
    if c(x^t) < c(x^b) then
        SETZE x^b = x^t
    AKTUALISIERE ρ^- und ρ^+
until Abbruchkriterium erfüllt ist
return x^b
// Output: Beste gefundene Lösung
```

- robust durch adaptiven Mechanismus

- keine explizite Nachbarschafts Auswahl nötig

6.8 Genetische Algorithmen

6.8.1 Ablauf

1. **Initialisierung**: Erzeugung einer Population: zufällig oder random-greedy
2. **Evaluation**: Auswertung der Fitness, bestimmt die Fortpflanzungs-wsk
3. **Rekombination**: Crossover-Operator erzeugt aus zwei Chromosomen zwei neue
4. **Reproduktion**: Klonen
5. **Mutation**: verhindert vorzeitige Konvergenz, stellt Vielfalt sicher

6.8.2 Schlüsselfaktoren

- Repräsentation: Binär Codierung, min. Alphabet, Repair vs. Strafkosten
- Selektion: proportional zu Fitness, nach Rang
- Populationsmodelle: diskrete vs überlappende Generation
- Fitte Eltern erzeugen noch fitteres Kind

6.8.3 Repräsentation für das TSP

- Adjazenz-Repräsentation:
  - speicher Stadt j an Stelle i, wenn $(i, j) \in E$
  - **Beispiel:** 3 5 7 6 4 8 2 1 $\rightarrow$ 1-3-7-2-5-4-6-8-1
  - **Problem:** Zyklen, zB. 3 5 7 6 2 4 1 8
- Pfad-Repräsentation: 1 2 3 4 5 6 7 8

6.8.4 Crossover

- Partially Matched Crossover (PMX):
  - Zufällige Wahl von 2 Crossover-Punkten und Austausch der dazwischen liegenden

    Beispiel:

Elternteil 1:	a	b	c	d	e	f	g	h	i	j
Elternteil 2:	c	f	g	a	j	b	d	i	e	h
Kind 1:	c	b	d	a	j	f	g	h	e	i
Kind 2:	a	f	c	g	e	b	d	i	h	j

  - absolute Pos. werden vererbt
  - geringer Rechenaufwand
  - mäßig fürs TSP
- Order Crossover-Operator (OX)
  - Zufällige Wahl von 2 Crossover-Punkten und Austausch der dazwischen liegenden
  - Ordnung der Städte der Elternteile wird bestmöglich beibehalten

    Beispiel:

Elternteil 1:	a	b	c	d	e	f	g	h	i	j
Elternteil 2:	c	f	g	a	h	b	d	i	e	j
Kind 1:	c	a	b	d	i	f	g	h	e	j
Kind 2:	c	e	f	g	h	b	d	i	j	a

  - Relativer Ordnung wird vererbt
  - geringer Rechenaufwand
  - Gute Ergebnisse für das TSP
- Cyclic Crossover (CX)
  - Absolute Position der Gene bleibt erhalten

Beispiel:

Elternteil 1:	*a*	b	*c*	*d*	e	f	*g*	h	i	j
Elternteil 2:	c	f	g	a	h	b	d	i	e	j
Kind 1:	a	f	c	d	h	b	g	i	e	j
Kind 2:	c	b	g	a	e	f	d	h	i	j

- etwas größerer Rechenaufwand
- weniger gute Ergebnisse für das TSP

6.9 Simulated Annealing (SA)

6.9.1 Idee

Basierend auf dem Abkühlen von Metallen.
Versuche ein höheres Energieniveau abzugeben und akzeptiere ein höheres Niveau mit vor schreitender Zeit immer weniger Wahrscheinlich.

6.9.2 Metropolis

Algorithmus:

Algorithmus 1 : Metropolis-Algorithmus

```
// Input: Zustandsraum X; Startzustand i ∈ X; Temperatur T
repeat
    WÄHLE zufälligen Nachbarzustand j von i
    BESTIMME Energiedifferenz Δ := E_j − E_i
    if Δ < 0 then
        // akzeptiere niedrigeres Energieniveau immer
        SETZE i := j
    else
        // bedingte Akzeptanz höherer Energieniveaus
        Mit Wahrscheinlichkeit exp(−Δ/kT) SETZE i := j
until (ohne Ende)
```

- Akzeptanzwahrscheinlichkeit: $P(i \to j) = exp(\frac{-\Delta}{kT})$ mit $\Delta = E_j - E_i$

6.9.3 Simmulated Annealing

- Besteht aus den folg. Bausteinen:

1. Nachbarschaft $\mathcal{N}$ und Startlösung x
2. Abkühlschema mit: Starttemperatur T_0, Anzahl Schritte bei konst. Temp L_t, Update-Funktion $u(T_t, t)$
3. Stoppkriterium

Algorithmus:

Algorithmus 2 : Simulated-Annealing (für Minimierungsproblem)

```
// Input: Startlösung x, Abkühlschema (T_t, L_t)
SETZE t := 0, L := L_0, T := T_0 und x_best := x
repeat
    // Innere Schleife = L Schritte im Metropolis-Alg.
    repeat L mal
        // zufällige Wahl
        WÄHLE zufällig ein x' ∈ N(x)
        SETZE Δ := c(x') − c(x)
        // zufällige Akzeptanz bei Verschlechterung
        if Δ ≤ 0 oder (Δ > 0 und mit Wahrscheinlichkeit exp(−Δ/T_t)) then
            SETZE x := x'
        if c(x) < c(x_best) then SETZE x_best := x
    until
    SETZE t := t + 1
    UPDATE T := T_t und L := L_t
until Stoppkriterium erfüllt
// Output: beste Lösung x_best
```

- Konvergenz zum Optimum, falls: $\mathcal{N}$ zusammenhängend und symmetrisch, $|\mathcal{N}(x)| = |\mathcal{N}(x')|$, Nachbar Auswahl gleichverteilt
- Mögliche **Stoppkriterien**:
  - letzten N Iterationen keine Verbesserung
  - $T < C$, wobei C vorgegebener Wert
- **Starttemperatur**: hoch genug um "gut durchzumischen"
- Intervalle L_t: möglichst gleich groß
- Update $u(T_t, t)$:

exponentiell: $T_{t+1} = \alpha \cdot T_t$ mit $\alpha \in \{0.8, 0.99\}$

Lundy-Mees: $T_{t+1} = \frac{T_t}{1+\beta T_t}$ für kleines β

Aarts Korst: komplex. Mathematik

6.9.4 Verwandte Methoden zu SA

Durch Änderung des *zufallsgesteuerten* (mit Wahrscheinlichkeit $\exp(-\frac{\Delta}{T})$ akzeptieren) in ein *deterministisches Akzeptanzkriterium* erhält man andere Metaheuristiken.

Bedingte deterministische Akzeptanz, wenn...

Threshold Accepting: ...die Verschlechterung $\Delta \leq \delta_t$ erfüllt für fallende Folge von Schwellwerten (δ_t), $\delta_t \to 0$ für $t \to \infty$

Sintflut-Algorithmus: ...der Zielfunktionswert $c(x') \leq \delta_t$ erfüllt für fallende Folge von Schwellwerten (δ_t), $\delta_t \to LB$ für $t \to \infty$

Record-To-Record-Travel: ...der Zielfunktionswert $c(x') \leq \delta_t$ erfüllt, wobei die Schwellwerte δ_t von der besten gefundenen Lösung (=record) abhängen (z.B. $\delta_t = (1+\varepsilon)c(x_{best})$ für $\varepsilon > 0$ klein)