

MAO - Zusammenfassung

Timo Bergerbusch

26. Januar 2018

Inhaltsverzeichnis

| | | |
|----------|---|-----------|
| 1 | Typische Probleme | 3 |
| 1.1 | Matching | 3 |
| 1.2 | Set-Packing/Covering/Partitioning | 3 |
| 1.3 | Traveling-Salesman-Problem (TSP) | 3 |
| 1.3.1 | Typische Distanzmatrixberechnung: | 4 |
| 1.4 | Bin-Packing-Problem (BPP) | 4 |
| 1.5 | Rucksackproblem (KP) | 4 |
| 2 | 1. Kapitel - Einführung | 5 |
| 3 | 2. Kapitel - Greedy Algorithmen | 8 |
| 4 | 3. Kapitel - Lösungsqualität und Approximation | 9 |
| 5 | 4. Kapitel - Lokale Suche | 9 |
| 6 | 5. Kapitel - Metaheuristiken | 12 |
| 6.1 | Einführung | 12 |
| 6.2 | Iterierte Lokale Suche (ILS) | 13 |
| 6.2.1 | Idee | 13 |
| 6.2.2 | Algorithmus | 13 |
| 6.2.3 | Stärken | 14 |
| 6.2.4 | Design | 14 |
| 6.3 | Variable Neighborhood Descent (VND) | 14 |
| 6.3.1 | Idee | 14 |

| | | |
|----------|--|-----------|
| 6.3.2 | Algorithmus | 15 |
| 6.4 | Tabu Search | 15 |
| 6.4.1 | Idee | 15 |
| 6.4.2 | attributives vs. explizites Gedächtnis | 16 |
| 6.4.3 | Tabu Restriktionen | 16 |
| 6.4.4 | Kurzzeitgedächtnis | 16 |
| 6.4.5 | Anspruchskriterium | 16 |
| 6.4.6 | Algorithmus | 17 |
| 7 | Large Neighborhood Search (LNS) | 17 |
| 7.1 | Idee | 18 |
| 7.2 | Algorithmus | 18 |
| 7.3 | Destroy-Operatoren | 18 |

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 $M = 1, \dots, m$
- n Teilmengen $M_1, \dots, M_n \subset M$
- Kosten/Profite der Teilmengen c_1, \dots, 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

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)

Gegeben:

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

Gesucht:

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

Greedy-Algorithmus

Unterscheide die versch. Strategien

- Weight Decreasing: absteigend nach Gewicht sortiert
- Next-Fit: Zuordnung zum zuletzt geöffneten
- First-Fit: Zuordnung zum ersten 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

Greedy-Algorithmus

Algorithmus 1 : Greedy-Algorithmus für das Rucksack-Problem

// Input: absteigend sortierte Elemente $i \in \{1, \dots, n\}$ nach
relativem Profit p_i/w_i

SETZE $C^{Rest} := C$

for $j = 1$ bis n (gemäß Sortierung) do

 if $w_j \leq C^{Rest}$ then

 SETZE $\bar{x}_j := 1$

 SETZE $C^{Rest} := C^{Rest} - w_j$

 else

 SETZE $\bar{x}_j := 0$

// Output: $(\bar{x}_1, \dots, \bar{x}_n)$

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

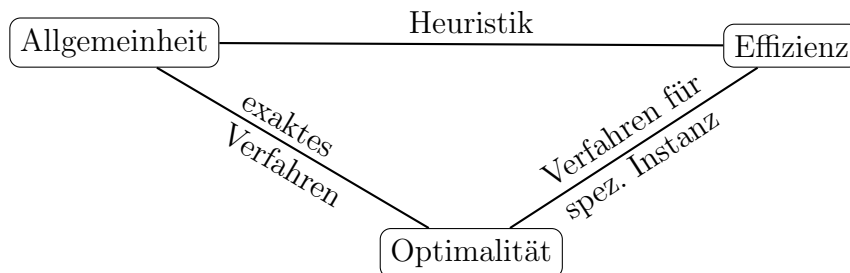
- Netzwerk-Design: Welche Standorte und welche Aufgaben dort?
- Transportplanung
- Standort Planung
- "Letzte Meile": Bezirkseinteilung und Postbotenrouten

- **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.-gesetzte 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., so dass $\text{time}_A(P) \leq k_1 + k_2 f(n)$ 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 $f : \Pi_2 \rightarrow \Pi_1$ und $g : L(\Pi_1) \rightarrow L(\Pi_2)$
 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  $J^{fix} := \emptyset$  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  $T = \emptyset$ 
repeat
    Wähle nächste Kante  $\{i, j\} \in E$  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 $\mathcal{N} : X \rightarrow Pot(X)$
- **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  $\mathcal{N}$ 
SETZE  $t := 0$  und  $x^0 := x$ 
repeat
    Durchsuche die Nachbarschaft  $\mathcal{N}(x^t)$  nach einem verbessernden
    Nachbarn  $x' \in \mathcal{N}(x^t)$ 
    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') → Füge (x', x'') hinzu → Lösche (x'', x''') → ...

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

⇒ 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

⇒ man benötigt $g_1 > 0$ und $g_2 > 0$ (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  $x = (x_1, x_2, \dots, x_n, x_1)$ 
SETZE  $G^* := 0$ 
for  $i_1 = 1, \dots, n$  und  $s \in \{-1, +1\}$  do
  SETZE  $t_1 := x_{i_1}, t_2 := x_{i_1+s}$ 
  SETZE  $Bound := c_{t_1, t_2} - G^* / 2$ 
  for  $t_3 \in N(t_2)$  mit  $c_{t_2, t_3} < Bound$  do
    SETZE  $i_2 := Position(t_3)$  und  $t_4 := x_{i_2-s}$ 
    SETZE  $G := c_{t_1, t_2} - c_{t_2, t_3} + c_{t_3, t_4} - c_{t_4, t_1}$ 
    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. Perturbation (ILS, VND)
- "Def.": Strategien zum Steuern des Suchprozesses; unabhängig vom Problem; Konkretisierung \Rightarrow Heuristik für spez. Optimierungsproblem
- Klassifikation ähnlich wie hier mit Erweiterung:
 - Zufall: deterministisch 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  $x^* := \text{LokaleSuche}(x)$ 
repeat
  SETZE  $x' := \text{Perturbation}(x^*, \text{Historie})$ 
  SETZE  $x'^* := \text{LokaleSuche}(x')$ 
  SETZE  $x^* := \text{AkzeptanzKriterium}(x^*, x'^*, \text{Historie})$ 
until  $\text{Terminierungsbedingung}(\text{Historie})$ 
// Output: beste gefundene Lösung
```

1. $\text{LokaleSuche}(x)$: Verbesserung einer Lösung x zu einem lokalen Optimum
2. $\text{Perturbation}(x, \text{historie})$: verwirbelt eine Lösung
3. $\text{AkzeptanzKriterium}(x, x', \text{historie})$: 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
- **Perturbation:**
 - 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  $\mathcal{N}_k(x^t)$  nach (verbesserndem) Nachbarn
     $x' \in \mathcal{N}_k(x^t)$  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 Tabu Search

6.4.1 Idee

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

6.4.2 attributives vs. explizites Gedächtnis

| Name | attributiv | explizit |
|--------------------|--|--|
| Speichern von: | einzelne Attribute einer Lösung(-sübergangs) | komplette Lösungen |
| Typische Elemente: | <ul style="list-style-type: none"> • Vorhanden sein eines Obj. in einer Lösung • Anzahl von Objekten • Austausch von Objekten | Verwaltung von Elite-Lös. |
| Vorteil: | geringer Speicheraufwand | test kein Tabu für nicht besuchte Lösungen |
| Nachteil: | mögl. tabu für nicht besuchte Lösung | Speicheraufwendig |

6.4.3 Tabu Restriktionen

| Nr. | Basis-Attribut | Tabu-Restriktion |
|-----|-------------------|--|
| 1 | Inc(i) | Verbietet Verringern von x_i auf 0 |
| 2 | Dec(j) | Verbietet 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

6.4.4 Kurzzeitgedächtnis

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

6.4.5 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 Zielfunktionswert | 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 |

6.4.6 Algorithmus

Algorithmus 1 : Tabu-Search mit Kurzzeitgedächtnis

// Input: Zulässige Lösung x^0

SETZE den Iterationszähler $t := 0$.

repeat

 // (Bestensuche)

 Durchsuche die gesamte Nachbarschaft $\mathcal{N}^t(x^t)$ und wähle eine bezüglich einer Bewertungsfunktion beste Lösung $x' \in \mathcal{N}^t(x^t)$, 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)

7 Large Neighborhood Search (LNS)

Alternative Namen: Ruin and Recreate, Fix and Optimize

7.1 Idee

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

7.2 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 \rightarrow Diversifikation
- Zerstörungsgrad:
 - zu gering \rightarrow keine Exploration des Lösungsraums
 - zu hoch \rightarrow zeitaufwendig und wenig erfolgversprechend

7.3 Destroy-Operatoren

| Name | Beschreibung | Effekt |
|------------------------|--|-----------------|
| Random Destroy | Entfernen zufällige Elemente | Diversifikation |
| Worst/Critical 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 |

7.4 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

8 Adaptive Neighborhood Search (ALNS)

8.1 Idee

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

8.1.1 Algorithmus

Algorithmus 2 : Adaptive Large Neighborhood Search

```
// Input: Zulässige Lösung  $x$ 
SETZE  $x^b = x$ ;  $\rho^- = (1, \dots, 1)$ ;  $\rho^+ = (1, \dots, 1)$ ;
repeat
    WÄHLE Destroy- und Repair-Operatoren  $d \in \Omega^-$  und  $r \in \Omega^+$  unter
        Verwendung von  $\rho^-$  und  $\rho^+$ 
    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  $\rho^-$  und  $\rho^+$ 
until Abbruchkriterium erfüllt ist
return  $x^b$ 
// Output: Beste gefundene Lösung
```

- robust durch adaptiven Mechanismus
- keine explizite Nachbarschafts Auswahl nötig