MAO - Zusammenfassung

Timo Bergerbusch

25. Januar 2018

Inhaltsverzeichnis

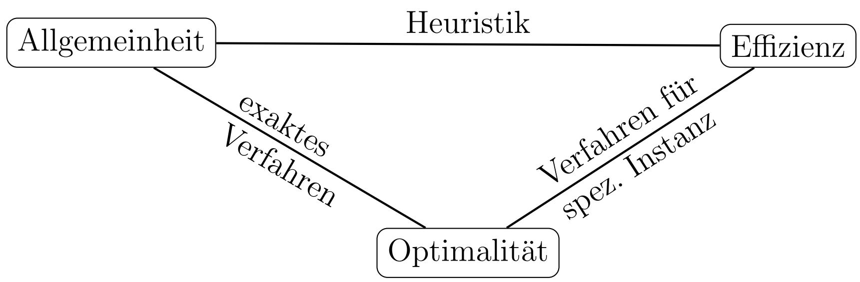
1 Fachbegriffe

- **optimal** : zulässig und bestmöglich
- **Simulation** : Durchführung von Experimenten anhand von Modellen
- **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 $\Pi \in \mathcal{NP}$
- **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 $J^{fix} = J$

- **Performance-Verhältnis** : Probleminstanz $P \in \Pi$, Heuristik $H \Rightarrow R_H(P) = \frac{z_H(P)}{z_{opt}(P)}$
- **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.

2 Typische Probleme

2.1 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…

2.2 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

2.3 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

2.3.1 Typische Distanzmatrixberechnung:

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

2.4 Bin-Packing-Problem (BPP)

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.

2.4.1 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

2.5 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

2.5.1 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)
```

3 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
- 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
- Die Standard Probleme
- (exakte, effiziente) Algorithmen
- $\mathcal{P}$, $\mathcal{NP}$ und polynomiale Reduzierbarkeit
- Begriff der Heuristik

4 2. Kapitel - Greedy Algorithmen

- Eröffnungsverfahren
- Allgemeine Greedy Algorithmen, BPP-Greedy and 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

5 3. Kapitel - Lösungsqualität und Approximation

- Def. des Performance Verhältnis
- 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 Analyse: worst-/average-case und empirische Analyse
- ϵ-Approximationsalgorithmen

6 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 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)