Grundlagen der Künstlichen Intelligenz

21. Kombinatorische Optimierung und lokale Suche

Malte Helmert

Universität Basel

10. April 2015

21.1 Kombinatorische Optimierung

Kombinatorische Optimierung: Einführung

vorige Kapitel: klassische Zustandsraumsuche

- finde Aktionsfolge (Pfad) von Anfangszustand zu Zielzustand mit möglichst niedrigen Kosten
- Schwierigkeit: grosse Anzahl an Zuständen („kombinatorische Explosion“)

dieses Kapitel: kombinatorische Optimierungsprobleme
ähnliches Szenario, aber:

- keine vorgegebenen Aktionen und Transitionen
- suche nicht Pfad, sondern Konfiguration („Zustand“) mit möglichst niedrigen Kosten/möglichst hoher Qualität

Kombinatorisches Optimierungsproblem

Definition (kombinatorisches Optimierungsproblem)

Ein kombinatorisches Optimierungsproblem (KOP)
ist gegeben durch ein 4-Tupel $\langle C, S, opt, v \rangle$ bestehend aus:

- einer Menge von (Lösungs-) Kandidaten C
- einer Menge von Lösungen $S \subseteq C$
- der Optimierungsrichtung $opt \in \{\min, \max\}$
- einer Zielfunktion $v : S \to \mathbb{R}$

Hinweise:

- „Problem“ hier in einem anderen Sinn (entsprechend „Probleminstanz“) als sonst in der Informatik üblich
- praktisch interessante KOPs haben in der Regel zu viele Kandidaten, als dass diese explizit aufgezählt werden könnten

Optimale Lösungen

Definition (optimale Lösung)

Sei $\mathcal{O} = \langle C, S, opt, v \rangle$ ein KOP.

Die optimale Lösungsqualität v^* von $\mathcal{O}$ ist definiert als

$$v^* = \begin{cases} \min_{c \in S} v(c) & \text{falls } opt = \min \\ \max_{c \in S} v(c) & \text{falls } opt = \max \end{cases}$$

(v^* ist undefiniert, wenn $S = \emptyset$ gilt.)
Eine Lösung s von $\mathcal{O}$ heisst optimal, wenn $v(s) = v^*$ gilt.

Kombinatorische Optimierung

Das grundlegende algorithmische Problem:

Kombinatorische Optimierung

Finde eine Lösung mit möglichst guter Qualität
für ein kombinatorisches Optimierungsproblem $\mathcal{O}$
oder weise nach, dass keine Lösung existiert.

Möglichst gut heisst hier, dass die Lösungsqualität
möglichst nahe an der optimalen Qualität v^* liegt.

Relevanz und Schwierigkeit

- Es gibt unzählige praktisch wichtige kombinatorische Optimierungsprobleme.
- Deren Lösung ist einer der zentralen Inhalte der Unternehmensforschung (operations research).
- Viele wichtige kombinatorische Optimierungsprobleme sind NP-vollständig.
- Die meisten „klassischen“ NP-vollständigen Probleme können als kombinatorische Optimierungsprobleme formuliert werden.
  - ⇝ Beispiele: TSP (vgl. Hausaufgaben), VertexCover, Clique, BinPacking, Partition

Suche vs. Optimierung

KOPs haben

- einen Suchaspekt (unter den gegebenen Kandidaten C eine Lösung in S finden) und
- einen Optimierungsaspekt (unter den gegebenen Lösungen S eine Lösung finden, die besonders gut ist).

Reine Such-/Optimierungsprobleme

Wichtige Spezialfälle ergeben sich,
wenn einer dieser Aspekte trivialisiert:

- reine Suchprobleme:
  - alle Lösungen sind gleich gut
  - Schwierigkeit ist, überhaupt eine Lösung zu finden
  - formal: v ist konstante Funktion (z.B. konstant 0); opt kann beliebig gewählt werden (spielt keine Rolle)
- reine Optimierungsprobleme:
  - alle Kandidaten sind Lösungen
  - Schwierigkeit ist, Lösung hoher Qualität zu finden
  - formal: $S = C$

21.2 Beispiel

Beispiel: 8-Damen-Problem

8-Damen-Problem

Wie kann man

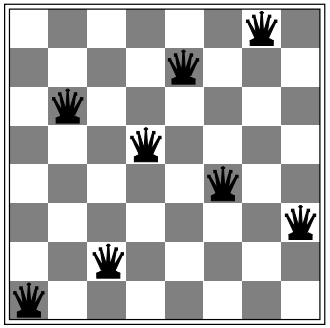
- 8 Damen auf einem Schachbrett platzieren,
- so dass sich keine zwei Damen bedrohen?

- ursprünglich 1848 vorgestellt
- Varianten: Brettgrösse; andere Figuren; mehr Dimensionen

Das Problem hat 92 Lösungen bzw. 12 Lösungen, wenn man symmetrische Lösungen (Rotation/Spiegelung) nur einmal zählt.

Beispiel: 8-Damen-Problem

Aufgabe: Platziere 8 Damen so auf einem Schachbrett, dass sie sich nicht bedrohen



Ist dieser Kandidat eine Lösung?

Formal: 8-Damen-Problem

Wie können wir das Problem formal definieren?

Idee:

- offensichtlich muss in jeder „Spalte“ genau eine Dame stehen
- beschreibe Kandidaten als 8-Tupel, wobei der i-te Eintrag die „Zeile“ für die Dame in der i-ten Spalte angibt

formal: $\mathcal{O} = \langle C, S, opt, v \rangle$ mit

- $C = \{1, \ldots, 8\}^8$
- $S = \{\langle r_1, \ldots, r_8 \rangle \mid \forall 1 \leq i < j \leq 8 : r_i \neq r_j \wedge |r_i - r_j| \neq |i - j|\}$
- v konstant, opt egal (reines Suchproblem)

21.3 Lokale Suche: Hill-Climbing

Algorithmen für kombinatorische Optimierungsprobleme

Welche Algorithmen verwendet man, um kombinatorische Optimierungsprobleme zu lösen?

- Formulierung als klassische Suchprobleme $\rightsquigarrow$ vorige Kapitel
- Formulierung als Constraint-Netz $\rightsquigarrow$ folgende Kapitel
- Formulierung als logisches Erfüllbarkeitsproblem $\rightsquigarrow$ später
- Formulierung als mathematisches Optimierungsproblem (LP/IP) $\rightsquigarrow$ nicht in dieser Vorlesung
- lokale Suche $\rightsquigarrow$ dieses Kapitel

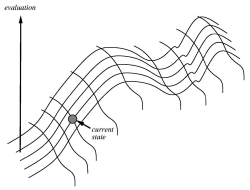
Suchverfahren für kombinatorische Optimierung

- Die Grundidee der heuristischen Suche ist für KOPs anwendbar, wenn wir Kandidaten als Zustände betrachten.
- Der wesentliche Unterschied: es gibt keine „Aktionen“ als Teil der Problemdefinition.
  - Stattdessen können wir selbst aussuchen, welche Kandidaten wir als „benachbart“ betrachten wollen.
  - Definition der Nachbarschaft wichtiger Aspekt beim Entwurf guter Algorithmen für gegebenes KOP
- Der „Pfad zum Ziel“ spielt aus Benutzersicht keine Rolle.
  - keine Pfadkosten; „Eltern“ und „erzeugende Aktionen“ egal
  - $\rightsquigarrow$ keine Suchknoten nötig

Lokale Suchverfahren: Ideen

Grundideen von lokalen Suchverfahren für KOPs:

- Heuristik h schätzt ab, wie „gut“ ein Kandidat ist
  - bei reinen Optimierungsproblemen oft Zielfunktion v selbst
  - bei reinen Suchproblemen oft Abstandsschätzung zur nächsten Lösung (wie in der Zustandsraumsuche)
- es werden keine Pfade gemerkt, nur Kandidaten
- oft nur ein aktueller Kandidat $\rightsquigarrow$ sehr speicherfreundlich (dafür nicht vollständig oder optimal)
- oft Initialisierung mit zufälligem Kandidaten
- schrittweise Verbesserung durch „Bergsteigen“ (hill-climbing)



Hill-Climbing

Hill-Climbing (für Maximierungsprobleme)

current := a random candidate
repeat:
    next := a neighbor of *current* with maximum h value
    if $h(next) \leq h(current)$:
        return *current*
    current := *next*

Anmerkungen:

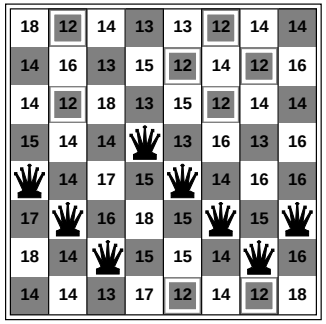
- Suche als Wanderung (walk) „bergauf“ in einer Landschaft, die durch die Nachbarschaftsbeziehung definiert ist.
- Heuristik-Werte definieren „Höhe“ des Geländes
- analoger Algorithmus für Minimierungsprobleme wird traditionell ebenfalls „Hill-Climbing“ genannt, auch wenn das Bild nicht ganz passt.

Eigenschaften von Hill-Climbing

- terminiert immer, wenn Kandidatenmenge endlich (Warum?)
- keine Garantie, dass Ergebnis eine Lösung ist
- wenn Ergebnis eine Lösung ist, ist sie lokal optimal (bzgl. h), aber es gibt keine globalen Qualitätsgarantien

Beispiel: 8-Damen-Problem

Aufgabe: Platziere 8 Damen so, dass sie sich nicht bedrohen
mögliche Heuristik: Anzahl Damenpaare, die sich bedrohen
(Formulierung als Minimierungsproblem)
mögliche Nachbarschaft: bewege eine Dame in ihrer Spalte



Performance-Zahlen für 8-Damen-Problem

- Problem hat $8^8 \approx 17$ Millionen Kandidaten.
  (zur Erinnerung: darunter 92 Lösungen)
- Nach zufälliger Initialisierung findet Hill-Climbing in etwa 14% der Fälle eine Lösung.
- Im Durchschnitt nur etwa 4 Schritte!

21.4 Verbesserungen für lokale Suche

Verbesserung von Hill-Climbing?

mögliche Verbesserung:

- „Stagnation“ (Schritte ohne Verbesserung) erlauben
- Anzahl Schritte beschränken, um Terminierung zu garantieren
- am Ende besten besuchten Kandidaten zurückliefern
  - reine Suchprobleme: terminieren, sobald Lösung gefunden

Beispiel 8-Damen-Problem:

- bei max. 100 erlaubten Schritten Lösung in 94% der Fälle
- im Durchschnitt 21 Schritte, bis Lösung gefunden

$\rightsquigarrow$ funktioniert hier sehr gut;
bei schwierigen Problemen aber oft nicht gut genug

Probleme von lokalen Suchverfahren

Probleme für Hill-Climbing:

- Lokale Optima: alle Nachbarn schlechter als aktueller Kandidat
- Plateaus: viele Nachbarn gleich gut wie aktueller Kandidat; keiner besser

Folgen:

- Algorithmus kann bei diesem Kandidaten stecken bleiben
- selbst wenn nicht: keine Führung zu (guter) Lösung möglich

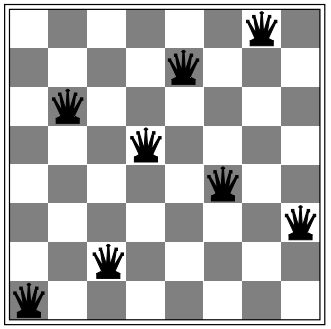
mögliche Abhilfen:

- (manchmal) zufällige Schritte
- Breitensuche zu besserem Kandidaten
- Neustart

Beispiel: lokales Minimum im 8-Damen-Problem

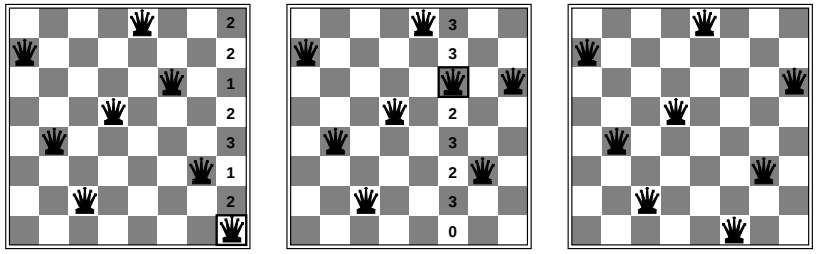
Lokales Minimum:

- Kandidat hat 1 Konflikt
- alle Nachbarn haben mindestens 2



Beispiel: Hill-Climbing für 8-Damen-Problem

Eine mögliche Variation von Hill-Climbing:
wähle zufällig eine Spalte und setze Dame dort auf Feld mit minimal vielen Konflikten.



Gute lokale Suchverfahren kombinieren oft Zufall (Exploration) mit Heuristik (Exploitation).

Ausblick: Simulierte Abkühlung

Simulierte Abkühlung (simulated annealing) ist ein lokales Suchverfahren, bei dem systematisch „Rauschen“ injiziert wird: erst viel, dann immer weniger.

- Wanderung mit fester Anzahl Schritte N (andere Varianten möglich)
- zu Beginn ist es „heiss“, und die Wanderung verläuft weitestgehend zufällig
- mit der Zeit wird es kälter (gesteuert durch Temperaturverlauf schedule)
- je kälter es wird, desto unwahrscheinlicher wird es, dass ein Übergang zu einem schlechteren Nachbarn zugelassen wird

Sehr erfolgreich in bestimmten Anwendungen, z. B. VLSI-Layout.

Simulierte Abkühlung: Pseudo-Code

Simulated Annealing (für Maximierungsprobleme)

curr := a random candidate
best := **none**
for each $t \in \{1, \ldots, N\}$:
  if is_solution(*curr*) **and** (*best* **is none or** $v(curr) > v(best)$):
    best := *curr*
  T := schedule(t)
  next := a random neighbor of *curr*
  $\Delta E := h(next) - h(curr)$
  if $\Delta E \geq 0$ **or** with probability $e^{\frac{\Delta E}{T}}$:
    curr := *next*
return *best*

Ausblick: Genetische Algorithmen

Die Evolution findet sehr erfolgreich gute Lösungen.

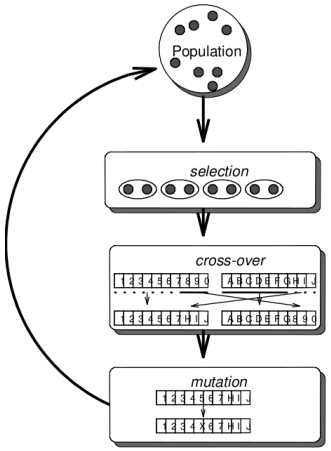
Idee: Simuliere Evolution durch Selektion, Kreuzen und Mutation von Individuen.

Zutaten:

- Kodierung von Kandidaten durch String von Symbolen (oder Bits)
- Fitness-Funktion: bewertet Güte von Kandidaten (entspricht Heuristik)
- Eine Bevölkerung von k (z. B. 10–1000) „Individuen“ (Kandidaten)

Beispiel 8-Damen-Problem: Kodierung als String von 8 Zahlen. Fitness ist Anzahl nicht-attackierender Paare. Population besteht aus 100 derartigen Kandidaten.

Selektion, Mutation und Kreuzung



Viele Varianten:
Wie wird selektiert? Wie wird gekreuzt? Wie wird mutiert?

Selektion anhand von Fitness-Funktion; anschliessend Paarung

Bestimmung von Punkten, an denen gekreuzt wird, dann Rekombination

Mutation: String-Elemente werden mit bestimmter Wahrscheinlichkeit verändert.

21.5 Zusammenfassung

Zusammenfassung (1)

kombinatorische Optimierungsprobleme:

- Finde unter vielen Kandidaten eine Lösung von möglichst hoher Qualität.
- Abgrenzung zur Zustandsraumsuche: keine Aktionen, Pfade etc.; nur „Zustand“ wichtig
- häufig gelöst durch lokale Suchverfahren

Zusammenfassung (2)

lokale Suche:

- Betrachte einen (oder wenige) Kandidaten auf einmal; versuche schrittweise Verbesserungen zu erzielen
- Schwierigkeit: lokale Optima und Plateaus
- Abhilfe: Abwägen von Exploration vs. Exploitation (z.B. durch Zufall und Neustarts)
- Simuliertes Abkühlung und genetische Algorithmen sind komplexere Suchverfahren, die typische Ideen aus der lokalen Suche aufgreifen (Randomisierung, Weiterverfolgung von viel versprechenden Kandidaten)