Grundlagen der Künstlichen Intelligenz

8. Suchalgorithmen: Lokale Suche

Malte Helmert

Universität Basel

5. April 2013

Einleitung

Suchprobleme: Überblick

Kapitelüberblick klassische Suchprobleme:

- Formalisierung von Suchproblemen ($\rightsquigarrow$ Kapitel 3)
- blinde Suchverfahren ($\rightsquigarrow$ Kapitel 4)
- Heuristiken ($\rightsquigarrow$ Kapitel 5)
- Bestensuche ($\rightsquigarrow$ Kapitel 6)
- Eigenschaften von A*($\rightsquigarrow$ Kapitel 7)
- Lokale Suche ($\rightsquigarrow$ dieses Kapitel)

Lokale Suchverfahren

Lokale Suchverfahren arbeiten nur mit einem (oder wenigen) aktuellen Knoten, statt systematisch Pfade zu untersuchen.

Besonders geeignet für kombinatorische Optimierungsprobleme:

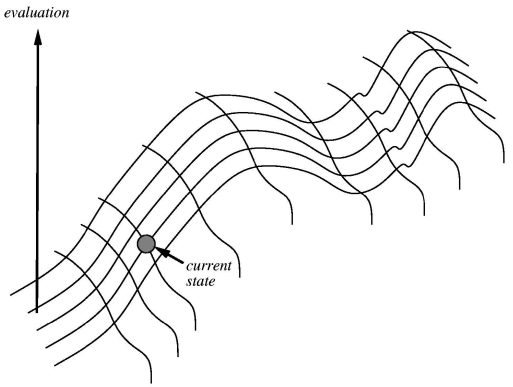
- Lösung ist ein Zustand, nicht der Pfad dorthin.
  Es gibt keine Aktionskosten, aber Zielzustände können verschiedene Kosten/Nutzen haben
- Definition der Suchnachbarschaft ist frei wählbar, nur Menge der Zustände ist durch die Problemdefinition vorgegeben (und ob sie Lösungen und wie teuer sie sind)
- Beispiele:
  - Timetabling
  - Konfigurationsprobleme
  - kombinatorische Auktionen

$\rightsquigarrow$ Fokus im Folgenden auf solchen Problemen,
aber auch für allgemeine klassische Suchprobleme anwendbar

Lokale Suchverfahren

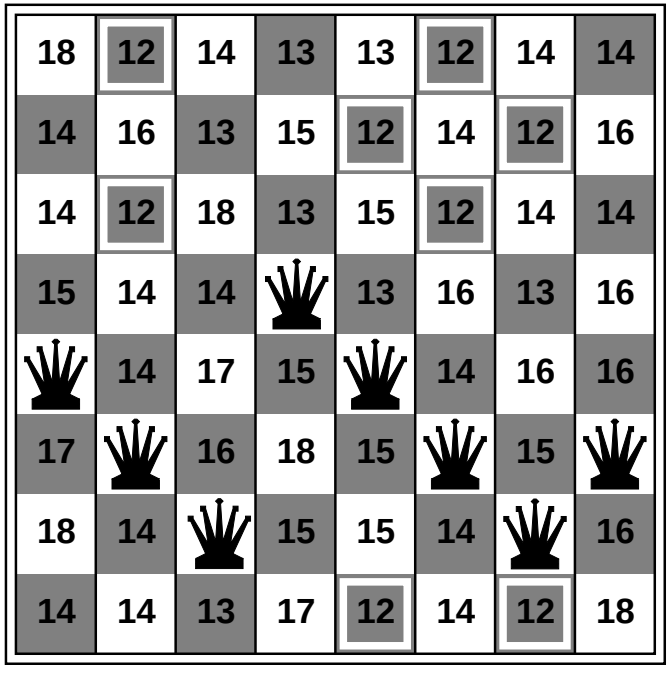
Lokale Suchverfahren: Ideen

- Heuristik schätzt ab, wie weit eine Lösung entfernt ist und/oder wie gut diese Lösung ist
- es werden keine Pfade gemerkt, nur Zustände
- normalerweise ein aktueller Zustand $\rightsquigarrow$ sehr speicherfreundlich (dafür nicht vollständig oder optimal)
- oft Initialisierung mit zufälligem Zustand
- schrittweise Verbesserung durch „Bergsteigen“ (hill-climbing)



Beispiel: 8-Damen-Problem

Aufgabe: Platziere 8 Damen so, dass sie sich nicht bedrohen
Heuristik: Anzahl Damenpaare, die sich bedrohen
Nachbarschaft: bewege eine Dame in ihrer Spalte



Hill-Climbing

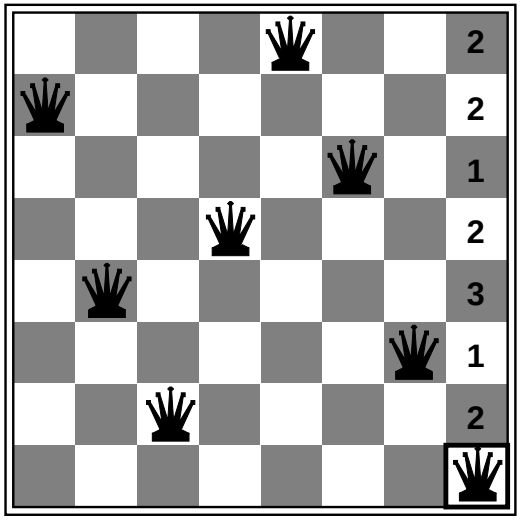
function HILL-CLIMBING(*problem*) **returns** a state that is a local maximum

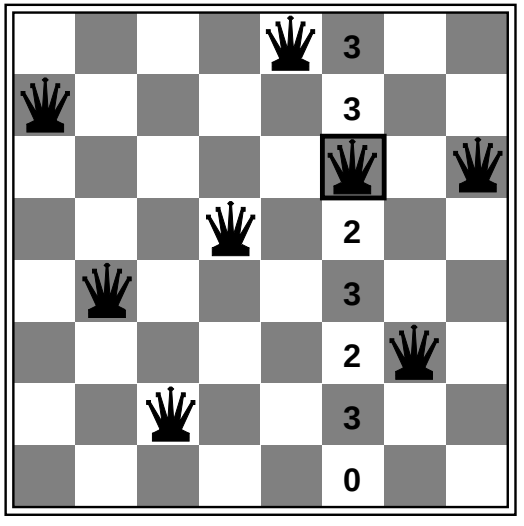
current ← MAKE-NODE(*problem*.INITIAL-STATE)
loop do
neighbor ← a highest-valued successor of *current*
if neighbor.VALUE ≤ current.VALUE **then return** *current*.STATE
current ← *neighbor*

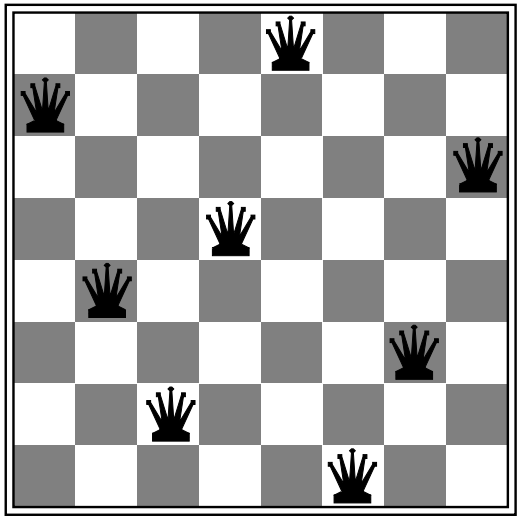
Hier als Maximierung von Qualität (value) formuliert.
Minimierung von Kosten/Heuristik analog.

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).

Probleme von lokalen Suchverfahren

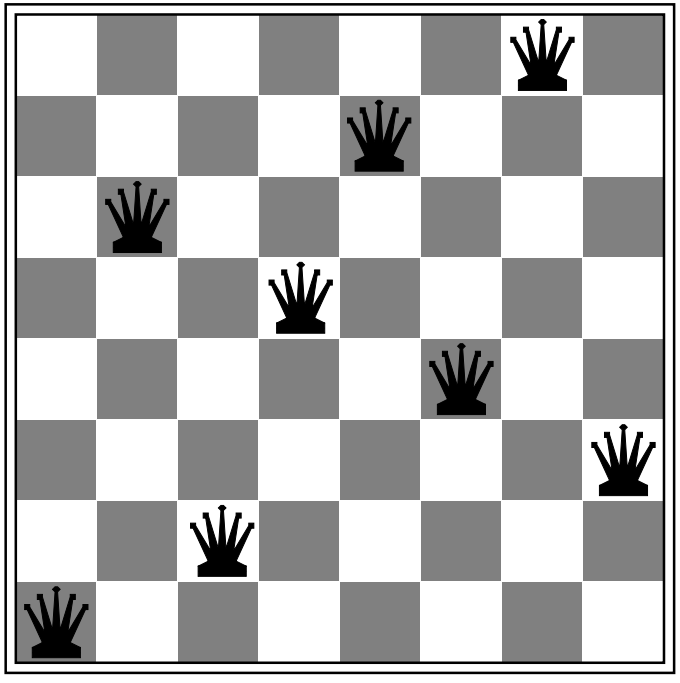
- Lokale Minima:
  alle Nachbarn schlechter als aktueller Zustand
  - Algorithmus bleibt auf diesem Zustand stecken
  - = lokale Maxima, wenn es um Maximierung von Nutzen/Qualität geht statt um Minimierung von Kosten
- Plateaus:
  viele Nachbarn gleich gut wie aktueller Zustand; keiner besser
  - keine Führung zum Zielzustand möglich

Massnahmen: zufällige Bewegung, Breitensuche oder Neustart

Beispiel: lokales Minimum im 8-Damen-Problem

Lokales Minimum:

- Zustand hat 1 Konflikt
- alle Nachbarn haben mindestens 2



Performance-Zahlen für 8-Damen-Problem

- Problem hat $8^8 \approx 17$ Millionen Zustände.
- Nach zufälliger Initialisierung findet Hill-Climbing in etwa 14% der Fälle direkt eine Lösung.
- Im Durchschnitt nur etwa 4 Schritte!
- bessere Verfahren: „Seitwärtsbewegung“ (Schritte ohne Verbesserung) erlauben und Anzahl Schritte beschränken
- Bei max. 100 erlaubten Schritten: Lösung in 94% der Fälle, im Durchschnitt 21 Schritte zur Lösung

Simulierte Abkühlung

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

function SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  inputs: $problem$, a problem
    $schedule$, a mapping from time to “temperature”

  $current \leftarrow$ MAKE-NODE($problem$.INITIAL-STATE)
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** $current$
    $next \leftarrow$ a randomly selected successor of $current$
    $\Delta E \leftarrow next$.VALUE – $current$.VALUE
    if $\Delta E > 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{\Delta E/T}$

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

Genetische Algorithmen

Die Evolution findet sehr erfolgreich gute Lösungen.

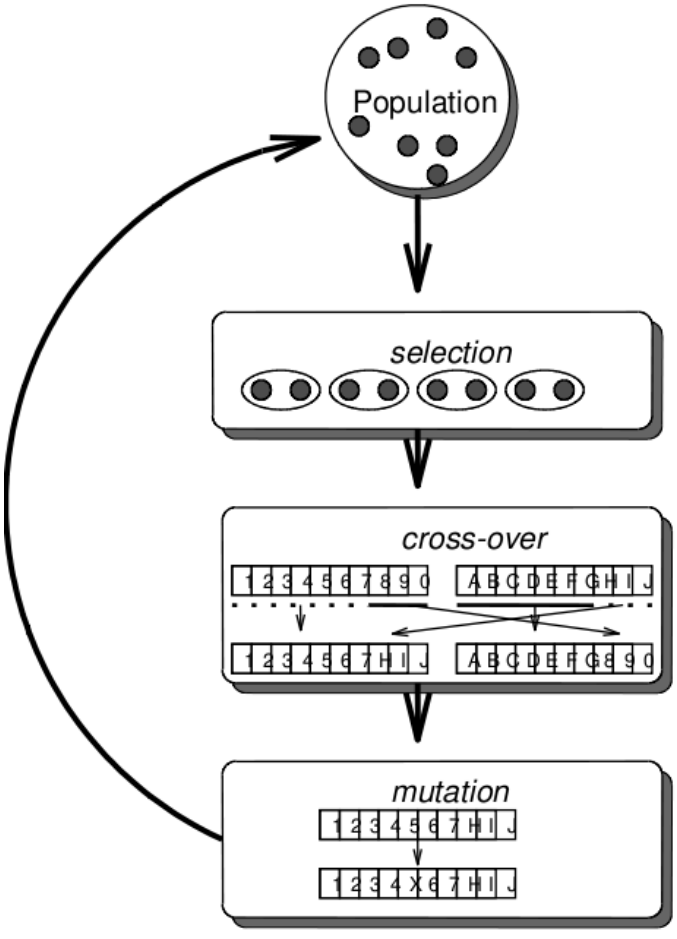
Idee: Simuliere Evolution durch Selektion, Kreuzen und Mutation von Individuen.

Zutaten:

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

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

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.

Zusammenfassung

Zusammenfassung

- lokale Suchverfahren betrachten einen oder wenige Zustände auf einmal und versuchen, schrittweise Verbesserungen zu erzielen
- besonders häufig im Einsatz für kombinatorische Optimierungsprobleme, wo der Pfad zum Ziel nicht wichtig ist, sondern nur der gefundene Zielzustand
- Simuliertes Abkühlung und genetische Algorithmen sind komplexere Suchverfahren, die typische Ideen aus der lokalen Suche aufgreifen (Randomisierung, Weiterverfolgung von vielversprechenden Zuständen)