Künstliche Intelligenz

Vorlesung 4: Suchverfahren

Lokale Suche

Lokale Suche

- Bislang: beobachtbare, deterministische, bekannte Umgebungen. Die Lösung ist eine Squenz von Aktionen.
- Pfade werden systematisch von einem Ausgangszustand erkundet.
- Jetzt: die Annahmen werden gelockert
- Suche wird lokal durchgeführt: aktuelle Zustände werden ausgewertet und abgeändert.
- Geeignet für Probleme, bei denen es nur auf den Lösungszustand ankommt und nicht auf die Pfadkosten um ihn zu erreichen.
- Hillclimbing, simulated Annealing (simulierte Abkühlung), genetische Algorithmen.

Lokale Suchalgorithmen und Optimierungsprobleme

- Globale Suche: Suchräume werden systematisch erkundet.
- Wie? Pfade werden gespeichert und Alternativen aufgezeichnet.
- Sobald ein Ziel gefunden ist, bildet der Pfad zu diesem Ziel eine Lösung für das Problem.
- Bei vielen Problemen ist der Pfad zum Ziel nicht von Bedeutung: 8-Damen-Problem, Entwurf von integrierten Schaltungen, Layout von Fertigungshallen, Terminplanung von Produktionsaufträgen, automatische Programmierung, Netzoptimierung in der Telekommunikation, Tourenplanung, Portfolioverwaltung.

Lokale Suche

- Pfad zum Ziel spielt keine Rolle mehr $\Rightarrow$ wir können eine andere Klasse von Algorithmen in Betracht ziehen
- Wir arbeiten jetzt mit einem einzelnen aktuellen Knoten (anstelle von mehreren Pfaden)
- Bewegung in Richtung des Nachbar Knoten
- Pfade werden nicht beibehalten
- Lokale Suchalgorithmen sind nicht systematisch

Lokale Suchalgorithmen - Vorteile

1. Sie brauchen sehr wenig Speicher - normalerweise eine konstante Menge
2. Können häufig sinnvolle Lösungen in großen oder unendlichen (stetigen) Zustandsräumen finden, die für die systematische Algorithmen nicht geeignet sind.

OPTIMIERUNGPROBLEME

- Das Ziel besteht darin den besten Zustand gemäß einer Zielfunktion zu finden.
- Viele Optimierungsprobleme passen nicht in das bisherige standard Suchmodell
- Beispiel: reproduktive Fitness → Darwinsche Evolution als Optimierungsversuch. Es gibt aber keinen Zieltest, noch Pfadkosten.

Typologien

- Einfache lokale Suche - ein einziger Nachbar Zustand wird gespeichert
  - Hill climbing $\rightarrow$ selektiert den besten Nachbar
  - Simulated annealing $\rightarrow$ selektiert den probabilistisch besten Nachbar
  - Tabu Suche $\rightarrow$ speichert eine Liste von besuchten Lösungen
- Gestrahlte lokale Suche - mehrere Zustände werden gespeichert (eine Population von Zustände)
  - Evolutionäre Algorithmen
  - Partikelschwarm Optimierung
  - Ameisen Kolonie Optimierung

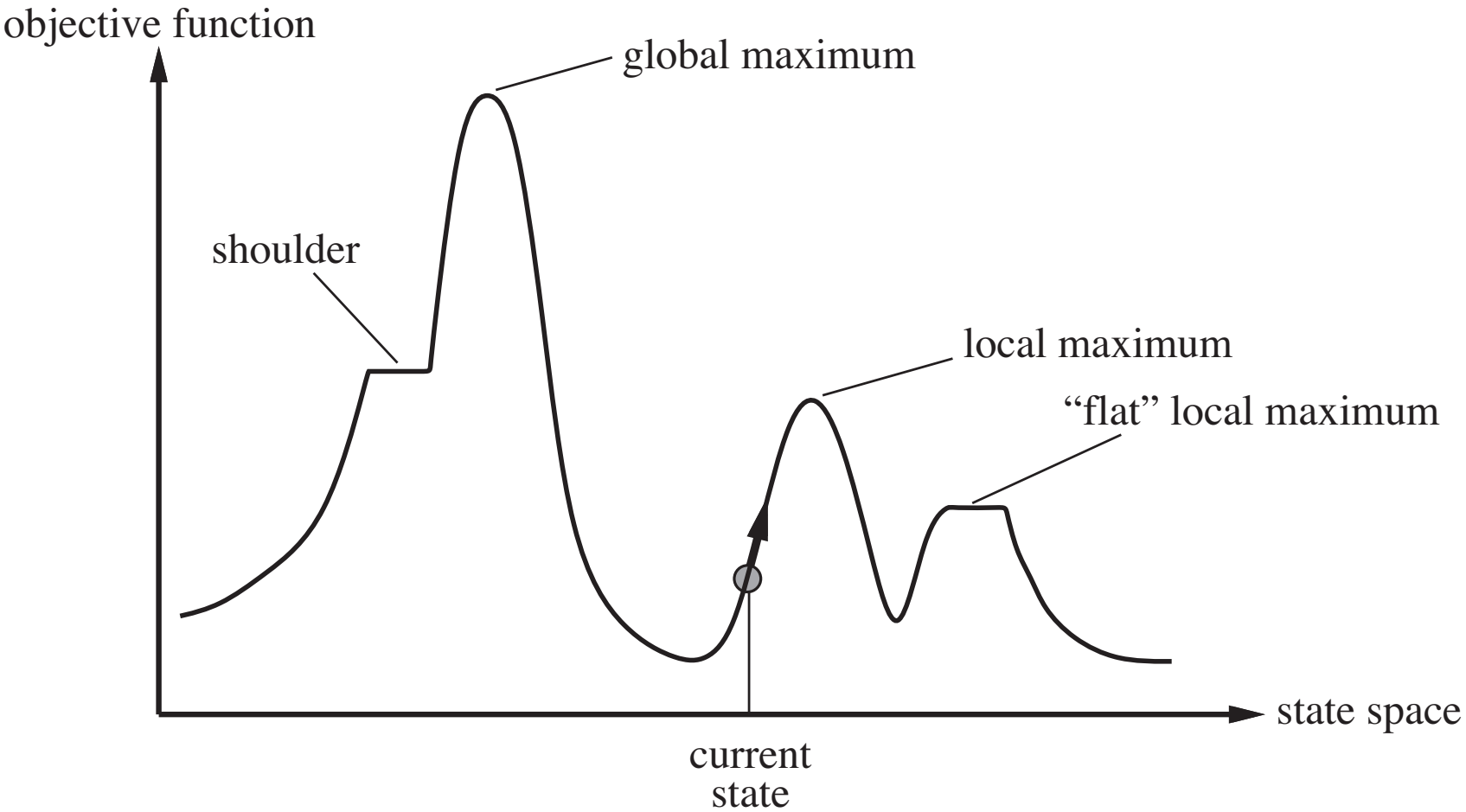
Einfache lokale Suche

- Zustandrepräsentation
- Evaluierung einer möglichen Lösung
- Nachbarn einer Lösung:
  - Wie definiere ich eine Nachbarlösung?
  - Wie werden Nachbarlösungen identifiziert:
    - Zufällig
    - Systematisch
- Wie wird eine mögliche Lösung akzeptiert?
  - Erster Nachbar des laufenden Zustands ist besser
  - Der beste Nachbar ist die bessere Lösung
  - Der beste Nachbar ergibt eine schechtere Lösung
  - Ein zufällig gewählter Nachbar ist die Lösung

Lokale Suche

- Zustandsraum-Landschaft
- Eine Landschaft hat sowohl eine Position (durch den Zustand definiert) als auch eine Erhebung (durch den Wert der heuristischen Kostenfunktion/Zielfunktion definiert)
- Wenn die Erhebung den Kosten entspricht, besteht das Ziel darin, das tiefste Tal zu finden - globales Minimum
- Entspricht die Erhebung einer Zielfunktion - finde höchsten Gipfel/globales Maximum
- lokale Suchalgorithmen erkunden diese Landschaft
- ein vollständiger lokaler Suchalgorithmus findet immer ein Ziel, wenn es ein solches gibt
- ein optimaler Algorithmus findet immer ein globales Minimum/Maximum

ZUSTANDSRAUM-LANDSCHAFT



Hillclimbing

- Schleife die ständig in die Richtung des steigendes Wertes verläuft $\rightarrow$ bergauf.
- Terminiert wenn sie einen Gipfel erreicht, wo kein Nachbar einen höheren Wert hat.
- Der Algorithmus verwaltet keinen Suchbaum. Die Datenstruktur zeichnet für den aktuellen Knoten nur den Zustand und den Wert der Zielfunktion.
- Dieses Bergsteigen sieht nicht über die unmittelbaren Nachbarn des aktuellen Zustandes hinaus.
- Suche die Bergspitze im dicken Nebel, während man gleichzeitig an Gedächtnisverlust leidet.

ALGORITHMUS

```
Bool HC(S){
   x = s1    //initial state
   x*=x      // best solution (found until now)
   k = 0     // # of iterations
   while (not termiantion criteria){
      k = k + 1
      generate all neighbours of x (N)
      Choose the best solution s from N
      if f(s) is better than f(x) then x = s
      else stop
   }  //while
   x* = x
   return x*;
}
```

HILLCLIMBING

- Wird auch als gierige lokale Suche (greedy local search) bezeichnet.
- Der Algorithmus wählt einen guten Nachbarzustand ohne darüber nachzudenken, wo es weitergehen soll.
- Es stellt sich heraus, dass gierige Algorithmen häufig gute Leistungen aufweisen.
- Das Bergsteigen kommt oftmals schnell zu einer Lösung, weil es normalerweise relativ einfach ist, einen schlechten Zustand zu verbessern.
- Im nächsten Beispiel sind 5 Schritte erforderlich um ein lokales Minimum mit Wert $h = 1$ zu erreichen und einer Lösung sehr nahe kommt.

Beispiel

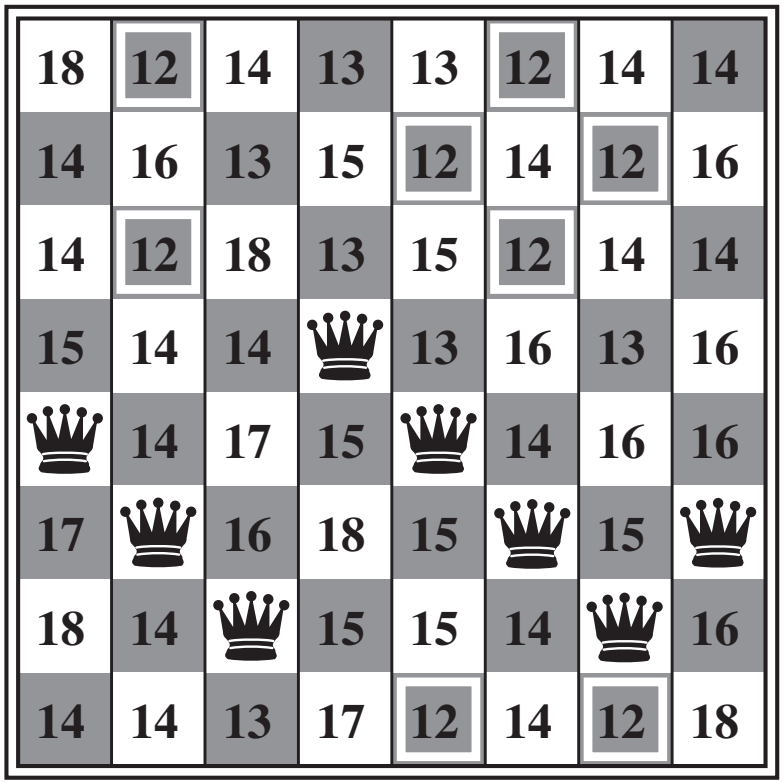


Abbildung 1: Ein 8-Damen-Zustand mit heuristischer Kostenschätzung $h = 17$, die den Wert von h für jeden möglichen Nachfolger zeigt, den man erhält, wenn man eine Dame innerhalb ihrer Spalte verschiebt. Die besten Züge sind markiert.

Beispiel

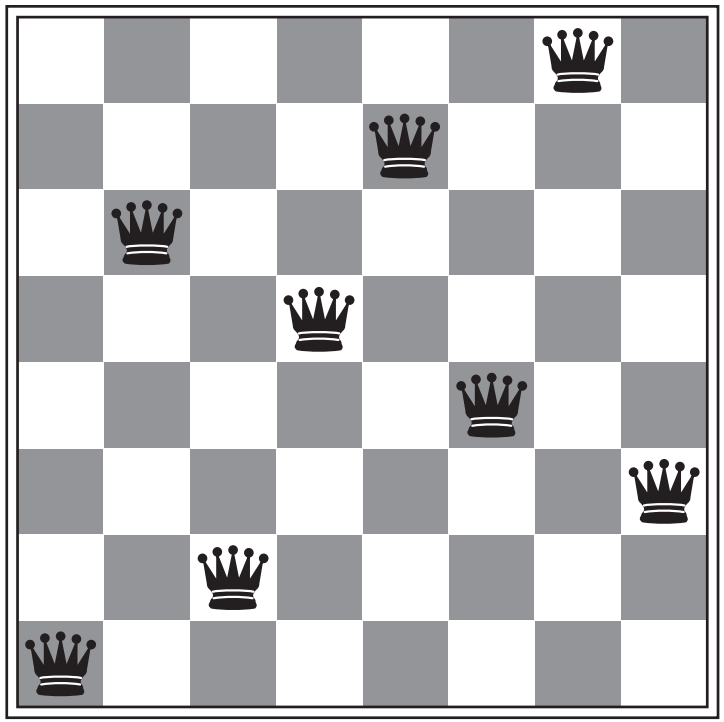


Abbildung 2: Ein lokales Minimum im 8-Damen-Zustandsraum; der Wert des Zustandes ist $h = 1$, doch sind die Kosten bei jedem Nachfolger höher.

Nachteile

- Bergsteigen bleibt aus folgenden Gründen häufig stecken
- Lokale Maxima: Ist eine Spitze, die höher als jeder ihrer Nachbarzustände, aber niedriger als das globale Maximum ist.
- Bergsteigeralgorithmen, die in die Nähe eines lokalen Maximums gelangen, werden nach oben in RIchtung Spitze gezogen, bleiben aber dort stecken und haben keine weitere Möglichkeiten mehr.

Nachteile

- Kammlinien führen zu einer Folge lokaler Maxima, in denen es gierigen Algorithmen sehr schwer fällt zu navigieren.

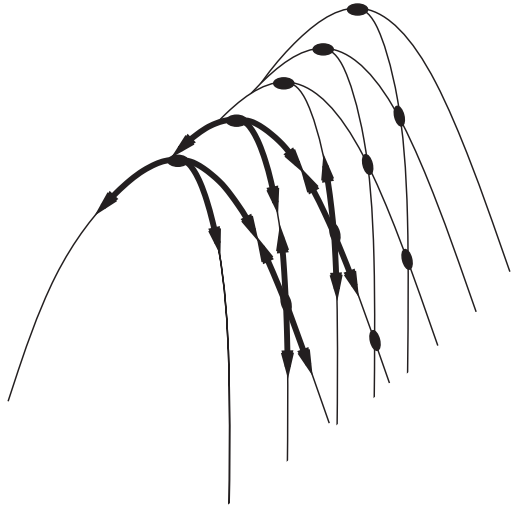


Abbildung 3: Das Zustandsraster (dunkle Kreise) wird von einer Kammlinie überlagert, die von links nach rechts verläuft und eine Folge lokaler Maxima erzeugt, die nicht direkt miteinander verbunden sind. Von jedem lokalen Maximum aus führen sämtliche möglichen Aktionen bergab.

Nachteile

- Plateaus: sind flache Bereiche der Zustandsraum-Landschaft. Es kann sich dabei um ein flaches lokales Maximum handeln, von dem aus es keinen Ausgang nach oben gibt, oder um einen Bergrücken, von dem aus weitere Fortschritte möglich sind. Eine Bergsteigen-Suche verirrt sich möglicherweise auf dem Plateau.
- In jedem Fall erreicht der Algorithmus einen Punkt, an dem kein Fortschritt mehr erziehlt wird.
- 8-Damen-Problem: zufälliger Zustand $\rightarrow$ Der Algorithmus bleibt in 86% aller Fälle stecken
- Löst nur 14% der Probleminstanzen
- Arbeitet schnell und braucht durchschnittlich 4 Schritte, wenn es erfolgreich ist, und 3, wenn es stecken bleibt
- Nicht schlecht für einen Zustandsraum mit 17 Mio Zuständen.

Hillclimbing Versionen

- Stochastisches Bergsteigen: nächster Zustand wird zufällig ausgewählt.
- First-choice: Nachfolger werden zufällig erzeugt bis ein neuer identifiziert wird.
- Random-restart: Restart von einem zufällig generierten Zustand, falls die Suche nicht fortschreitet.

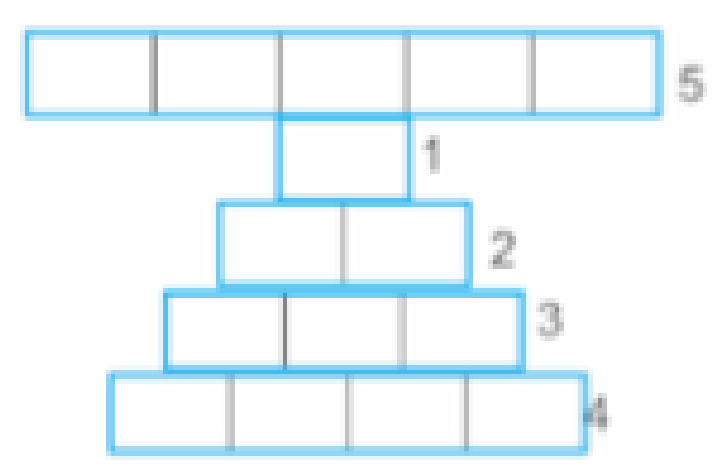
Beispiel - Turmkonstruktion

- Gegeben sind n Rechtecke (gleiche Breite, verschiedene Längen), die in einem Stapel gespeichert sind.
- Baue einen stabilen Turm aus allen Rechtecken.
- Zugriff ist nur auf das oberste Element des Stapels möglich, welches auf einer der zwei Hilfstapel verschoben wird.

Beispiel - Turmkonstruktion

- Darstellung der Lösung
  - Zustand x - Vektor von n Paare (i, j), i ist der Index des Rechtecks ($i = 1, 2, \ldots, n$) und j ist der Index der Stapeln ($j = 1, 2, 3$).
  - Initialer Zustand - Vektor des initialen Turmes
  - Endzustand - Vektor des Endturmes.
- Evaluierung
  - f_1 - Anzahl der richtig positionierten Rechtecke $\rightarrow$ maximiert
  - f_2 - Anzahl der falsch positionierten Rechtecke $\rightarrow$ minimiert
  - $f = f_1 - f_2 \rightarrow$ maximiert
- Nachbarschaft: mögliche Züge - bewege ein Rechteck aus dem Stapel j_1 auf dem j_2
- Akzeptanz einer Lösung: Bester Nachbar der laufenden Lösung besser als die laufende Lösung

Beispiel - Turmkonstruktion



Beispiel - Turmkonstruktion

- Schritt 1:
  - Laufender Zustand = initialer Zustand:
    - $x = s_1 = ((5,1),(1,1),(2,1),(3,1),(4,1))$
    - Rechtecke $1, 2, 3$ sind richtig positioniert
    - Rechtecke $4, 5$ sind falsch positioniert
    - $f(s_1) = 3 - 2 = 1$
  - $x^* = x$
- Nachbarn des laufenden Zustands x - Rechteck $5 \rightarrow$ Stapel 2
  - $s_2 = ((1,1),(2,1),(3,1),(4,1),(5,2))$
  - $f(s_2) = 4 - 1 = 3 > f(x) \rightarrow x = s_2$

Beispiel - Turmkonstruktion

Beispiel - Turmkonstruktion

- Schritt 2: Laufender Zustand =
  $x = ((1,1),(2,1),(3,1),(4,1),(5,2)), f(x) = 3$
- Nachbarn des laufenden Zustands - 2:
  - Rechteck 1 wird auf Stapel 2 verschoben
    $\rightarrow s_3 = ((2,1),(3,1),(4,1),(1,2),(5,2)) \rightarrow$
    $f(s_3) = 3 - 2 = 1 < f(x)$
  - Rechteck 1 wird auf Stapel 3 verschoben
    $\rightarrow s_4 = ((2,1),(3,1),(4,1),(5,2),(1,3)) \rightarrow$
    $f(s_4) = 3 - 2 = 1 < f(x)$
  - Es gibt kein Nachbar, der besser als x ist $\rightarrow$ STOP
- $x^* = x = ((1,1),(2,1),(3,1),(4,1),(5,2))$
- x^* ist ein lokaler Optimum

Beispiel - Turmkonstruktion

Beispiel - Turmkonstruktion

- Evaluierung
  - f_1 ist Summe der Stapelhöhe der richtig eingesetzten Rechtecke (Final $f_1 = 10$) $\rightarrow$ maximieren
  - f_2 ist Summe der Stapelhöhe der falsch eingesetzten Rechtecke (Final $f_2 = 0$) $\rightarrow$ minimieren
- Nachbarschaft: Mögliche Züge - bewege Rechteck i vom Stapel j_1 auf Stapel j_2

Simulated Annealing - simuliertes Abkühlen

- Ein Bergsteigeralgorithmus, der niemals Abwärtsbewegungen in Richtung von Zuständen mit geringerem Wert (oder höhere Kosten) macht, ist garantiert unvollständig
- Er kann an ein lokales Maximum hängen bleiben.
- Ein zufälliges Weitergehen (random walk) - die Bewegung zu einem Nachfolger, der gleichverteilt zufällig aus der Menge der Nachfolger ausgewählt wird - ist vollständig aber äußerst ineffizient.
- Sinnvoll das Bergsteigen mit einem zufälligen Weitergehen zu kombinieren, sodass man sowohl Effizienz und Vollständigkeit erhält.

Simulated Annealing - simuliertes Abkühlen

- Metallurgie: Glühen (Annealing) ist der Prozess, Metalle oder Glas zu tempern oder zu härten, indem man sie auf eine hohe Temperatur aufheizt und dann allmählich abkühlt, sodass das Material in einen kristallinen Zustand mit geringer Energie übergehen kann.
- Wir ändern unsere Perspektive vom Bergsteigen zum Gradientenabstieg (d.h. Kostenminimierung): Bringe einen Tischtennisball in die teifste Spalte einer buckligen Oberfläche. Wenn der Ball weiterrollt, dann bleibt er an einem lokalen Minimum liegen.
- Wenn wir die Oberfläche schütteln, können wir den Ball aus dem lokalen Minimum herausspringen lassen.
- Vorsicht: nicht allzuviel schütteln!
- Simulated Annealing: zuerst stark schütteln (hohe Temperatur) und dann die Intensität des Schüttelns allmählich zu verringern.

ALGORITHMUS

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling the probability of downward steps

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

Algorithmus

- Die innere Schleife ist Bergsteigen ähnlich
- Anstatt die beste Bewegung auszuwählen, greift der Algorithmus auf eine zufällige Bewegung heraus.
- Verbessert die Bewegung die Situation, wird sie akzeptiert.
- Andernfalls akzeptiert der Algorithmus die Bewegung mit einer Wahrscheinlichkeit kleiner als 1.
- Die Wahscheinlichkeit sinkt exponentiell mit der Schlechtigkeit der Bewegung, um den Betrag ΔE um den die Auswertung verschlechtert wird.
- Die Wahrscheinlichkeit verringert sich ebenfalls, wenn die Temperatur T sinkt: schlechte Bewegungen treten zu Beginn wahrscheinlicher auf, wenn T hoch ist, und werden immer unwahrscheinlicher, wenn T sinkt.
- Wenn der Zeitplan den Wert von T langsam genug sinken lässt, findet der Algorithmus ein globales Optimum mit einer Wahrscheinlichkeit nahe 1.

Algorithmus

- Stop Bedingungen:
  - Eine Lösung ist gefunden
  - Die Anzahl der Iterationen wurde erreicht
  - Die Einfriertemperatur wurde erreicht ($T = 100$)

Algorithmus

- Wie wird die Wahrscheinlichkeit gewählt?
  - $p = 0.1$
  - p fällt mit den Iterationen
  - p fällt mit den Iterationen und der Fehler $\mid f(s) - f(x) \mid$ steigt
    - $p = \exp(- \mid f(s) - f(x) \mid / T)$
    - Für hohe Temperaturen T wird fast jeder Nachbar akzeptiert
    - Für kleine T nur Nachbarn, die besser als s sind werden akzeptiert
  - Für große Fehlerwerte sind die Wahrscheinlichkeiten klein

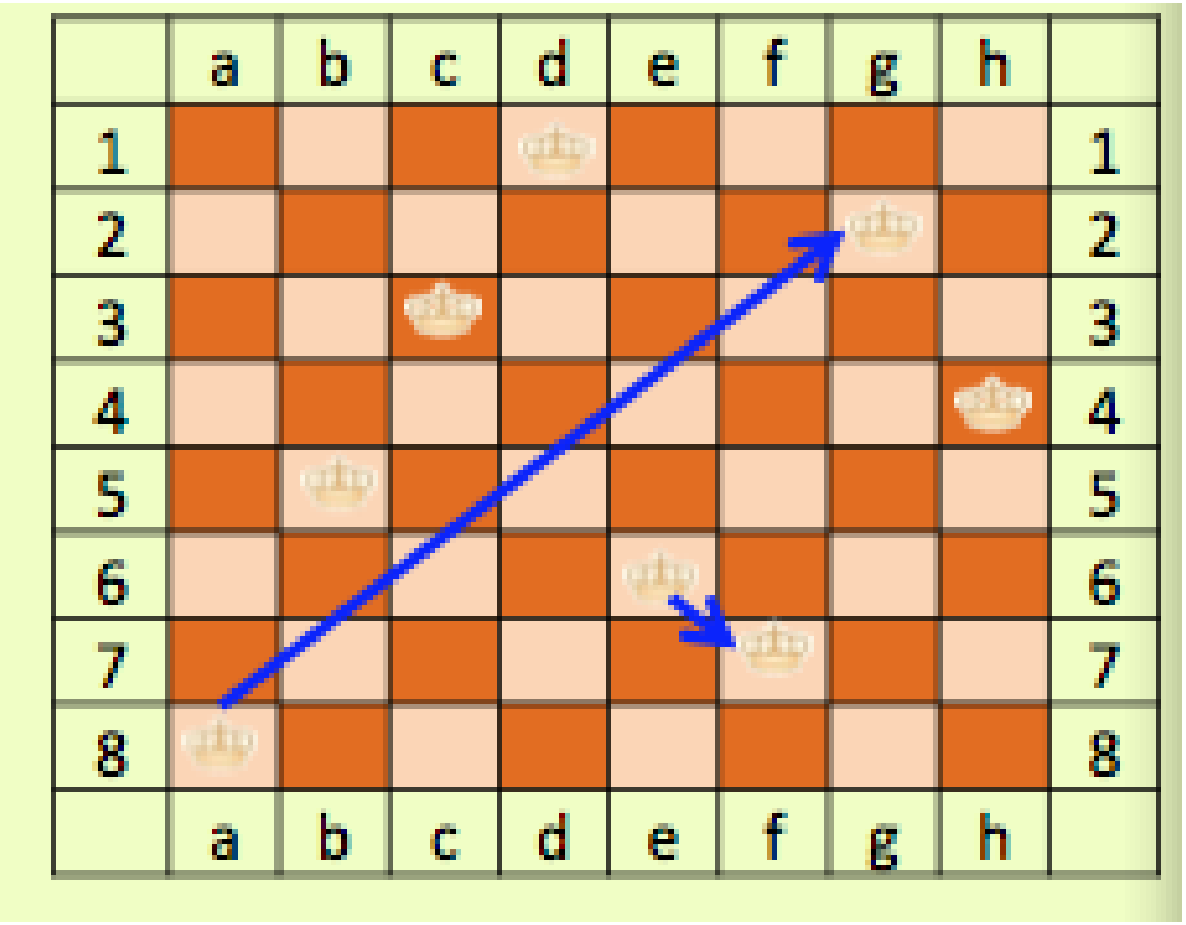
Beispiel: 8-Damen-Problem

- Lösungsrepräsentation:
  - Zustand X - Permutation von n Elemente;
    $x = (x_1, x_2, \ldots, x_n)$, x_i die Zeile auf der die Dame der j-ten Spalte bewegt wird.
  - Keine Spalten- und Zeilenattacke sind erlaubt; Diagonalattacke sind möglich
  - Initialer Zustand: ein zufällig gewählter Zustand (d.h. eine zufällige Permutation)
  - Endzustand: eine Permutation ohne Attacke
- Evaluierung: F - Summe der von einer Dame attackierten Damen $\rightarrow$ Minimisieren
- Nachbarn: mögliche Züge (2 Elemente einer Permutation vertauschen)

Beispiel: 8-Damen-Problem

- Akzeptanz einer Lösung: wähle zufällig einen Nachbar.
  - besser als der Zustand
  - schlechter als der Zustand mit Wahrscheinlichkeit $P(\Delta E) = e^{\frac{\Delta E}{T}}$
  - ΔE Energie Unterschied zwischen zwei Zustände
  - T Temperatur, $T(k) = \frac{100}{k}$, k Anzahl der Iterationen

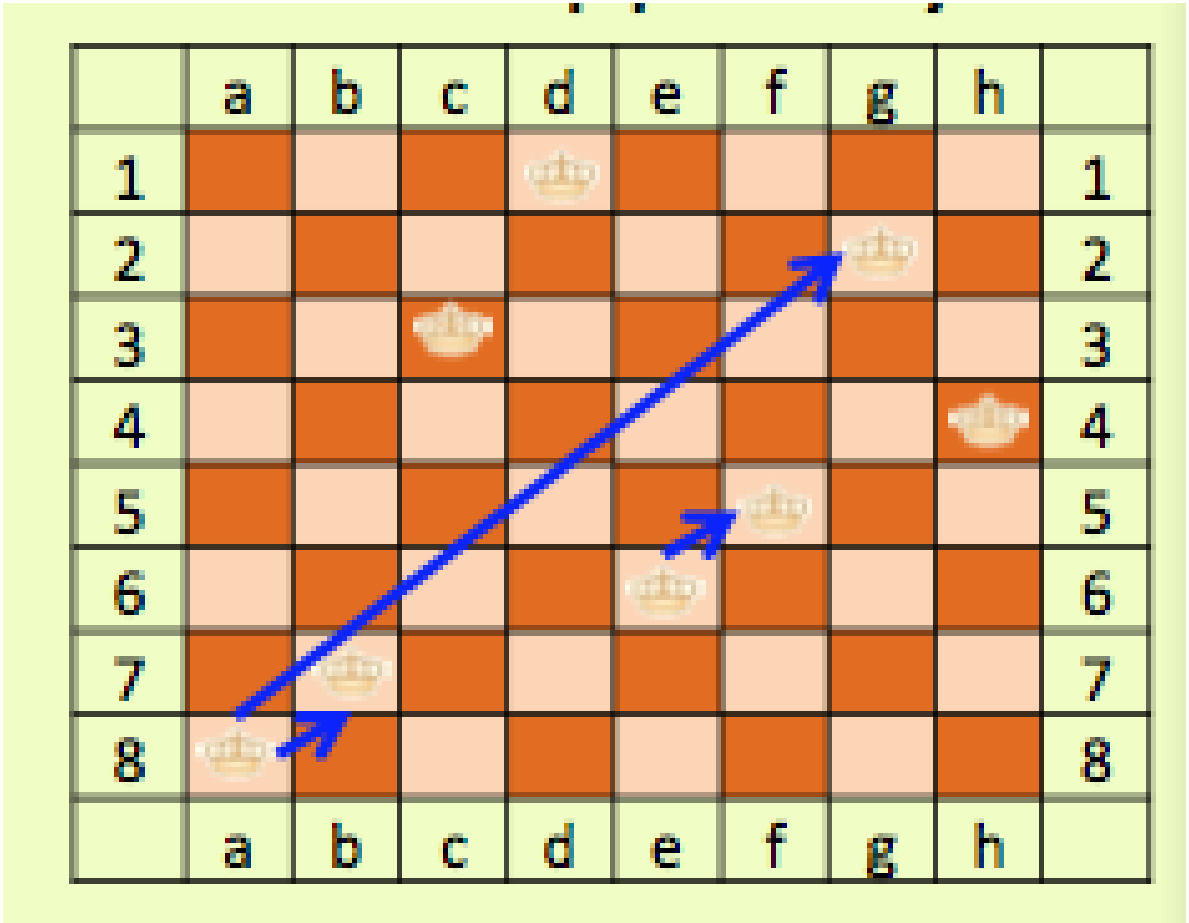
Beispiel: 8-Damen-Problem



Beispiel: 8-Damen-Problem

- Schritt 1
- Laufender Zustand = initialer Zustand x
- $s_1 = (8, 5, 3, 1, 6, 7, 2, 4)$
- $f(s_1) = 1 + 1 = 2$
- $x = x^*$
- $T = \frac{100}{1} = 100$
- Ein Nachbar von $x \rightarrow$ Dame auf Zeile 5 ist mit Dame auf Zeile 7 vertauscht

Beispiel: 8-Damen-Problem



- $s_2 = (8, 7, 3, 1, 6, 5, 2, 4)$
- $f(s_2) = 1 + 1 + 1 = 3 > f(x)$
- $\Delta E = f(s_2) - f(s_1) = 1$
- $P(\Delta E) = e^{\frac{-1}{100}}$
- $r = \text{ random}(0, 1)$
- if $r < P(\Delta E) \rightarrow x = s_2$

Lokale Strahlsuche

- Verwaltet k Zustände statt nur ein einziges
- Beginnt mit k zufällig erzeugten Zuständen
- Bei jedem Schritt werden alle Nachfolger aller k Zustände erzeugt
- Ist einer davon ein Ziel, wird der Algorithmus beendet
- Andernfalls wählt er die besten k Nachfolger aus der vollständigen Liste aus und wiederholt den Ablauf

Lokale Strahlsuche

- Auf den ersten Blick erscheint die lokale Suche mit k Zuständen nichts anderes zu sein als die paralelle statt sequentielle Ausführung von k zufälligen Neustarts.
- Die beiden Algorithmen sind sehr unterschiedlich
- Bei einer Suche mit zufälligem Neustrat wird jeder Suchprozess unabhängig von den anderen ausgeführt.
- Bei einer lokalen Strahlsuche werden wichtige Informationen zwischen den parallelen Such-Threads übergeben.
- Der Algorithmus bricht fruchtlose Suchen schnell ab und konzentriert seine Ressourcen dort, wo der größte Fortschritt erzielt wird.

Stochastische Strahlsuche

- Anstatt die besten k aus dem Pool der möglichen Nachfolger auszuwählen, wähle k Nachfolger zufällig aus

Tabu Suche

- Starte mit einem Zustand welcher gegen irgendwelcher Beschränkungen verstößt
- Elimiere diese Verstöße (suche die beste Nachbar-Lösung zu der laufenden Lösung)
- Speichere: laufender Zustand, besuchte Zustände und Bewegungen
- Akzeptanz: Bester Nachbar zu der laufenden Lösung, besser als die laufende Lösung und noch nicht besucht

ALGORITHMUS

```
bool TS(S){
    Select x∈S   //S - search space
    x*=x         //best solution until a moment
    k = 0        //iteration number
    T = ∅        //list of tabu moves
    while (not termination criteria){
       k = k + 1
       generate a subset of solutions in the neighbourhood N-T of
   x
       choose the best solution s from N-T and set x=s.
       if f(x)<f(x*) then x*=x
       update T with moves of generating x
    }  //while
    return x*;
}
```

Stop Bedingungen

- Feste Anzahl von Iterationen
- Vorgegeben Anzahl von Iterationen ohne Verbesserung
- Hinreichende Nähe zur Lösung (falls bekannt)
- Abbau unbesuchter Elemente einer Nachbarschaft

Tabu Suche

- Analyse: konvergiert schnell zum globalem Optimum
- Vorteile: Der Algorithmus ist allgemein und ist leicht zu implementieren + kurze Laufzeit
- Nachteile: Identifikation der Nachbarn in stetige Suchräume, große Anzahl von Iterationen, Identifizierung globaler Optima nicht garantiert

Tabu Suche - Anwendungen

- Bestimmung 3D Struktur der Proteinen in Aminosäure Sequenzen
- Optimierung in Kommunikationnetzwerke
- Planung in Produktionsanlagen
- Network design in optische Telekommunikation
- Automatisches Routing von KFZ
- Partitionieren von Graphen
- Planning in audit Systeme

Genetische Algorithmen

- Variante der stochastischen Strahlsuche
- Nachfolgerzustände werden erzeugt, indem zwei übergeordnete (Eltern-) Zustände kombiniert werden, anstatt einen einzelnen Zustand zu verändern.
- Starte mit einern Menge k zufällig erzeugten Zuständen: Population
- Jeder Zustand (Individuum) wird durch eine Zeichenfolge aus einem endlichen Alphabet dargestellt: $\{0, 1\}$
- Beispiel: 8-Damen Zustand stellt jeweils die Position von acht Damen das, die jeweils in eine Spalte mit acht Quadraten stehen $\rightarrow 8 \times \log_2 8 = 24$ Bit.
- Alternativ könnte der Zustand auch mit acht Ziffern dargestellt werden, die im Bereich zwischen 1 und 8 liegen.

Genetische Algorithmen

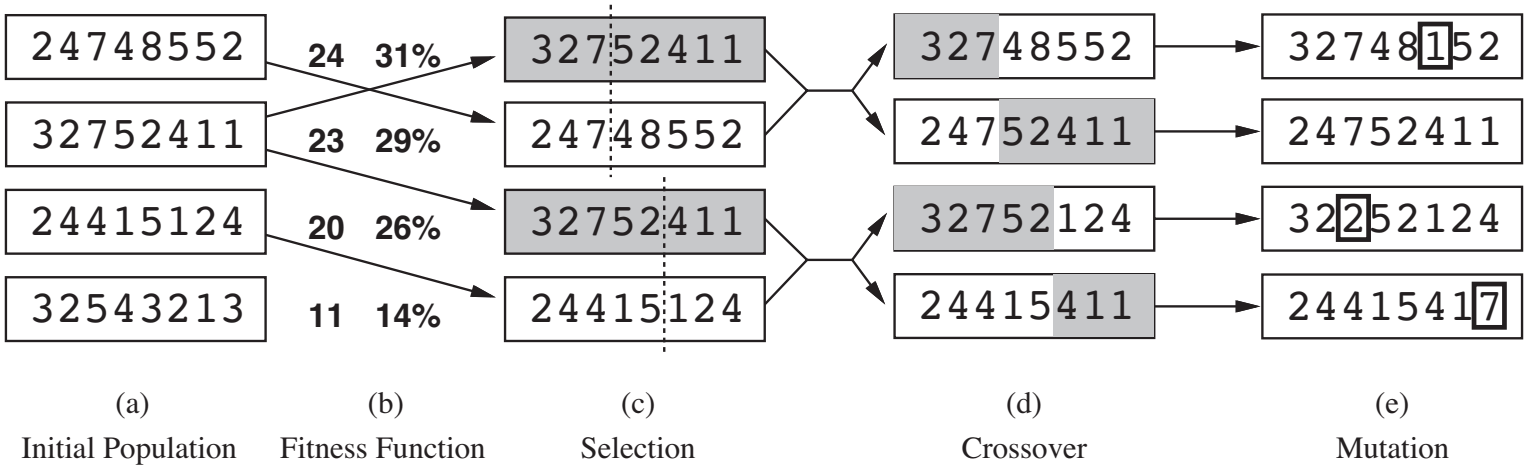


Abbildung 4: Der genetische Algorithmus, hier veranschaulicht an Zeichenfolgen aus Ziffern, die 8-Damen-Zustände darstellen. Die Anfangspopulation in (a) wird nach der Fitness-Funktion in (b) geordnet, woraus sich Fortpflanzungspaare in (c) ergeben. Die produzieren eine Nachkommenschaft in (d), die in (e) einer Mutation unterzogen wird

Genetische Algorithmen

- Zielfunktion = Fitness-Funktion. Für bessere Zustände werden höhere Werte zurückgegeben.
- Für das 8-Damen-Problem verwenden wir die Anzahl der sich nicht angreifenden Damenpaare, die einen Wert von 28 für eine Lösung aufweist.
- Die Werte der 4 Lösungen sind 24, 23, 20 und 11.
- In dieser Variante ist die Wahrscheinlichkeit, zur Reproduktion ausgewählt zu werden, direkt proportional zur Fitness-Punktzahl.
- In (c) werden zwei Paare zufällig entsprechend den Wahscheinlichkeiten in (b) zur Reproduktion ausgewählt. Ein Individuum wird zweimal ausgewählt und eines überhaupt nicht.

Genetische Algorithmen

- Für jedes zu verbindende Paar wird ein Kreuzungspunkt zufällig aus den Positionen in der Zeichenfolge ausgewählt (Crossover).
- Hier befinden sich die Kreuzungspunkte nach der dritten Ziffer im ersten Paar und nach der fünften Ziffer im zweiten Paar.
- In (d) werden die eigentlichen Nachkommen erzeugt, indem die Eltern-Zeichenfolgen am Kreuzungspunkt gekreuzt werden.
- Das erste Kind des ersten Paares erhält die ersten drei Ziffern vom ersten Elternteil und die restlichen vom zweiten Elternteil. Das zweite Kind erhält die ersten drei Ziffern vom zweiten Elternteil und die übrigen Ziffern vom zweiten Elternteil.

Genetische Algorithmen

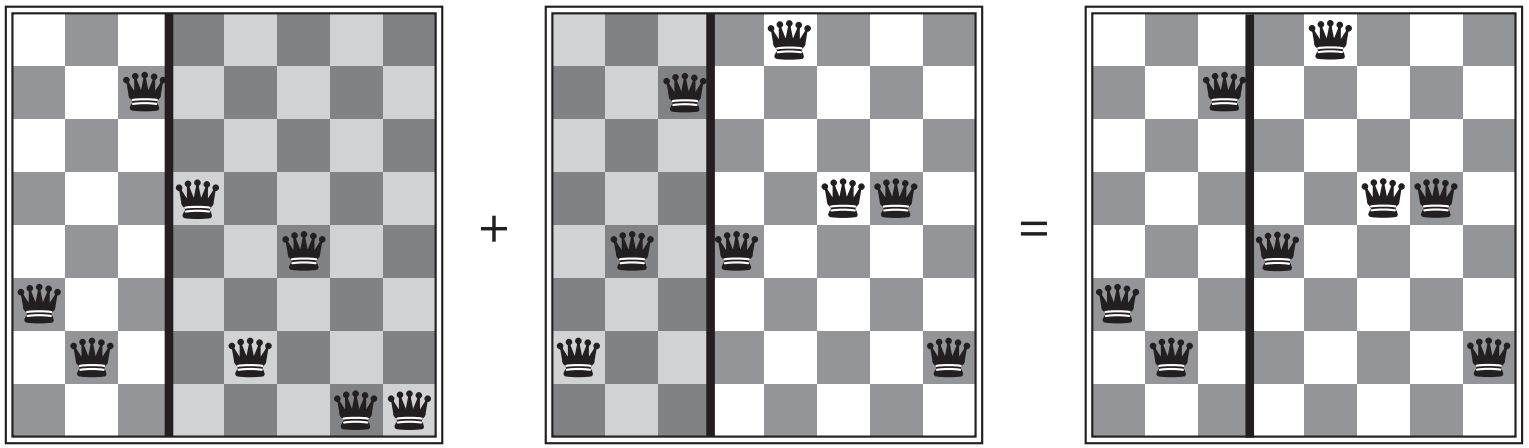


Abbildung 5: Die 8-Damen-Zustände, die den beiden ersten Eltern entsprechen, sowie dem ersten Nachkommen. Die grau schattierten Spalten gehen bei der Kreuzung verloren. Die nicht schattiert dargestelten Spalten werden beibehalten

Genetische Algorithmen

- Die Kreuzung bei recht verschiedenen Elternteilen kann einen Zustand hervorbringen, der sich erheblich von beiden Elternzuständen unterscheidet.
- In (e) unterliegt jede Position einer zufälligen Mutation mit einer kleinen unabhängigen Wahrscheinlichkeit.
- Eine Ziffer wurde im 1., 3. und 4. Nachkommen mutiert
- Im 8-Damen-Problem entspricht dies der zufälligen Auswahl einer Dame, die dann auf ein zufälliges Quadrat in ihrer Spalte verschoben wird.

Genetische Algorithmen

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
    new-population ← empty set
    loop for i from 1 to SIZE(population) do
      x ← RANDOM-SELECTION(population, FITNESS-FN)
      y ← RANDOM-SELECTION(population, FITNESS-FN)
      child ← REPRODUCE(x, y)
      if (small random probability) then child ← MUTATE(child)
      add child to new-population
    population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x)
  c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

Genetische Algorithmen

- Wie die stochastische Strahlensuche kombinieren genetische Algorithmen eine Aufwärtstendenz mit einer zufälligen Erkundung und dem Austausch von Informationen zwischen parallelen Such-Threads.
- Der wichtigste Vorteil ergibt sich aus der Kreuzungsoperation.
- Der Vorteil stammt aus der Möglichkeit des Crossover, große Buchstabenblcoke zu kombinieren, die sich unabhängig voneinander entwickelt haben, um sinnvolle Funktionen auszuführen und damit den Grad der Granularität der Suche zu erhöhen.
- Beispiel: Platziere die ersten drei Damen auf den Positionen 2, 4 und 6 (wo sie sich gegenseitig nicht angreifen). Dies könnte einen sinnvollen Block bilden, der sich mit anderen Blöcken kombinieren läßt, um eine Lösung zu konstruieren.

Anwendungen

- Optimierungprobleme
- Layout von Schaltkreisen
- Produktionsablaufplanung

Es blebt aber noch viel zu tun, um die Bedingungen herauszuarbeiten, unter denen genetische Algorithmen die beste Leistung erbringen.