

Einführung in die Methoden der Künstlichen Intelligenz

Evolutionäre Algorithmen

PD Dr. David Sabel

SoSe 2014

- **Anwendungsbereich:** Optimierung von Objekten mit komplexer Beschreibung
- **Objekte:** Üblicherweise kodiert als **Bitfolgen** mit **fester Länge**
- **Bewertungsfunktion** für die Objekte
- **Datenstruktur:** (Multi-)Menge von Objekten
- Arbeitsweise: Erzeuge aus Multimenge neue Multimenge
- entspricht daher **paralleler Suche**
- Stoppe wenn optimales Objekt in der Menge

Üblicherweise:

Probleme mit sehr vielen Zuständen,
daher **unmöglich** alle Zustände zu durchsuchen

Idee daher: Simulation der Evolution von Lebewesen

- gute Eigenschaften setzen sich durch
- andere sterben aus

Ziel: Züchte Schweine mit besonders großen Ohren

Ziel: Züchte Schweine mit besonders großen Ohren

Wiederhole bis Ohren groß genug

- Wähle Schweine aus aktueller Zucht mit den größten Ohren
- Paare diese Schweine

Ziel: Züchte Schweine mit besonders großen Ohren

Wiederhole bis Ohren groß genug

- Wähle Schweine aus aktueller Zucht mit den größten Ohren
- Paare diese Schweine

Hoffnung dabei: Eigenschaft „große Ohren“ setzt sich durch

Umsetzung für Evolutionäre Algorithmen

Eingaben

- Anfangspopulation:
 - Multimenge von Individuen (Chromosomen(-satz))
 - Ein Individuum: Bitstring
 - Teilfolge: Gen
- Fitnessfunktion (Bewertung für Individuen)

Algorithmus:

- Erzeuge iterativ nächste Generationen
- Mittels genetischer Operationen (Selektion, Mutation, Rekombination, . . .)
- höhere Fitness = mehr Nachkommen
- Erzeugung zufällig (mit Wahrscheinlichkeiten)

Ausgabe: Optimales Individuum

bzw. Individuum mit bester Fitness in Generation n

Wahl der Anfangspopulation

- Möglichst breit (verschieden)
- Normalerweise: Zufällig

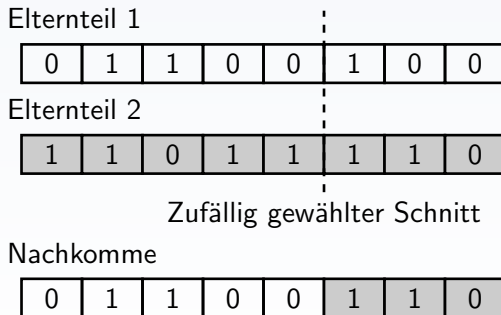
Selektion:

- Auswahl von Individuen aus Generation n für die Erzeugung von Generation $n + 1$
- Höhere Fitness = höhere Auswahlwahrscheinlichkeit

Genetische Operationen (2)

Rekombination (Crossover)

- Wähle zwei Eltern mit Selektion aus
- Wähle zufällige Schnittstelle
- Erzeuge Nachfahren durch Zusammensetzen



Mutation

- verändere zufällig ausgewählte Bits in den Chromosomen ($0 \rightarrow 1$ bzw. $1 \rightarrow 0$)
- i.A. mit geringer Wahrscheinlichkeit

Aussterben

- Wenn Population zu groß
- schlechtestes oder zufälliges Individuum stirbt aus

Ende der Evolution: Wenn vermutlich das Optimum erreicht

Algorithmus Genetische Suche

Eingabe: Anfangspopulation, Fitnessfunktion ϕ , K die Populationsgröße

Datenstrukturen: S, S' : Mengen von Individuen

Algorithmus:

$S :=$ Anfangspopulation;

while (K enthält kein Individuum mit maximalem ϕ) do:

$S' := \emptyset$

 for $i := 1$ to K do:

 Wähle zufällig (mit ϕ gewichtet) zwei Individuen A und B aus S ;

 Erzeuge Nachkomme C aus A und B durch Rekombination;

$C' :=$ Mutation (geringe Wahrscheinlichkeit) von C ;

$S' := S' \cup \{C'\}$;

 end for

$S := S'$

end while

Gebe Individuum mit maximaler Fitness aus

Genetisch vs. Evolutionär

Genetischer Algorithmus

- Bitfolgen
- Selektion, Rekombination, Mutation: alle möglichen Bitfolgen

Evolutionärer Algorithmus

- auch allgemeinere Folgen (z.B. ganze Zahlen)
- Beschränkung auf bestimmte Bitfolgen
- angepasste Operationen (z.B. nur Permutationen als Nachfolger)

Adjustiere Parameter

- Erzeugung der initialen Population und Kriterien zum Beenden
- Mutationswahrscheinlichkeit: wieviel Bits werden geflippt?, welche Bits?
- Crossover-Wahrscheinlichkeit: an welcher Stelle wird zerschnitten? Welche Individuen dürfen Nachkommen zeugen?
- Abhängigkeit der Anzahl der Nachkommen von der Fitness
- Größe der Population

Notwendig:

- Finden einer geeigneten Repräsentation der Zustände
- Fitnessfunktion finden
- Finden geeigneter genetischer Operatoren (auf der Repräsentation).

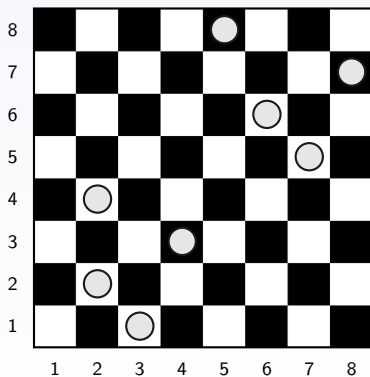
Probleme: lokale Maxima (Bergsteigereffekt)

Alternativen:

- ① Proportional zur Fitness.
 - Optimierung sehr sensibel für genauen Wert und der Steigung der Fitness
 - Plateaus / Optima: sehr flach, keine bevorzugte Auswahl
- ② Proportional zur Reihenfolge innerhalb der Population.
 - z.B. bestes Individuum 2-3 fach wahrscheinlicher als schlechtestes Individuum.
 - Fitness-Funktion kann etwas ungenauer sein
 - Auswahl hängt nicht vom genauen Wert ab.
- ③ Proportional zu einer normierten Fitness
 - z.B. $\text{Fitness} - c$, wobei c minimale Fitness

Beispiel: n-Damen

- Kodierung: Folge von n Zahlen im Bereich $1, \dots, n$
- i . Zahl: Position von Dame in Zeile i



Kodiert: [3,2,4,2,7,6,8,5]

Beispiel: n-Damen (2)

Fitness:

- Fitness: Anzahl nicht bedrohender Damenpaare
- schlechtester Wert: 0, jede Dame bedroht jede andere
- bester Wert: $\binom{n}{2} = \frac{n*(n-1)}{2}$

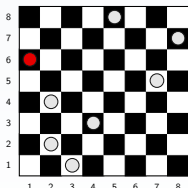
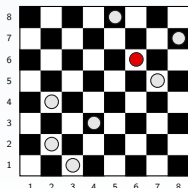
Beispiel: n-Damen (2)

Fitness:

- Fitness: Anzahl nicht bedrohender Damenpaare
- schlechtester Wert: 0, jede Dame bedroht jede andere
- bester Wert: $\binom{n}{2} = \frac{n*(n-1)}{2}$

Mutation:

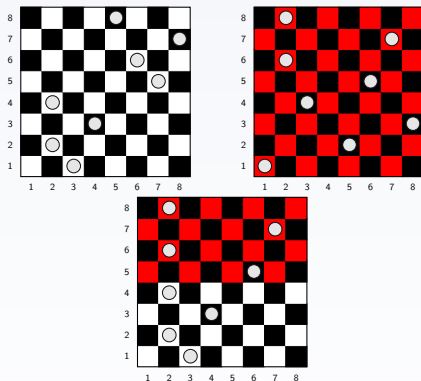
- Ändere einen Wert zufällig auf Wert aus $\{1, \dots, n\}$
- entspricht: bewege Dame in einer Zeile auf anderen Platz



$$[3,2,4,2,7,6,8,5] \rightarrow [3,2,4,2,7,1,8,5]$$

Beispiel: n-Damen (3)

Rekombination:



$[3,2,4,2,7,6,8,5]$ und $[1,5,8,3,6,2,7,2]$ mit Schnitt in der Mitte ergibt $[3,2,4,2,6,2,7,2]$

n-Damen: Bessere Kodierung

Da man schon mehr weiß:

- Erlaube nur Bitfolgen, die Permutationen von $[1, \dots, n]$ sind
- Rekombination erscheint dann eher sinnlos
(fast immer erhält man einen ungünstigen Zustand)

Abhilfen:

- sofortiges Aussterben
- Zufälliges Reparieren
- Rekombination nicht durchführen
- Mutation: Tausche zwei Elemente

Beachte: Reine Mutationsveränderungen sind analog zu Bergsteigen und Best-First-Suche.

Beispiel mit 5 Damen

Anfangspopulation $\{[3, 2, 1, 4, 5]\}$
 Fitness $\varphi([3, 2, 1, 4, 5]) = 4$
 Mutationen $[3, 2, 1, 4, 5] \rightarrow [5, 2, 1, 4, 3];$
 $[3, 2, 1, 4, 5] \rightarrow [3, 4, 1, 2, 5]$

2.Generation:

Population $\{[5, 2, 1, 4, 3], [3, 4, 1, 2, 5]\}$
 Bewertung $\varphi([5, 2, 1, 4, 3]) = 6, \varphi([3, 4, 1, 2, 5]) = 6$
 Mutationen $[3, 4, 1, 2, 5] \rightarrow [3, 4, 1, 5, 2],$
 $[5, 2, 1, 4, 3] \rightarrow [5, 2, 4, 1, 3]$

3.Generation:

Population $\{[5, 2, 1, 4, 3], [3, 4, 1, 2, 5], [2, 4, 1, 3, 5], [5, 2, 4, 1, 3]\}$
 Bewertung $\varphi([3, 4, 1, 5, 2]) = 8, \varphi([5, 2, 4, 1, 3]) = 10$

Optimum erreicht!

Die Kodierung $[5, 2, 4, 1, 3]$ ist eine Lösung.

Wesentliche Unterschiede zu Standardsuchverfahren:

- Evol. Alg. benutzen **Codierungen der Lösungen**
- Evol. Alg. benutzen eine Suche, die **parallel** ist und auf einer **Menge von Lösungen** basiert.
- Evol. Alg. benutzen **nur die Zielfunktion zum Optimieren**
- Evol. Alg. benutzen **probabilistische** Übergangsregeln.
Der Suchraum wird durchkämmt mit **stochastischen Methoden**

- Gute Kodierung erfordert **annähernde Stetigkeit**:
fast stetiger Zusammenhang zwischen Bitfolge und Fitnessfunktion, d.h.
Flippen von Bits ändert die Fitness nur wenig
- Optimale Lösung sollten durch **Herantasten** auffindbar sein
(nicht versteckte einzelne Punkte)
- Rekombination scheint nur dann sinnvoll, wenn es einen **Zusammenhang von Teilfolgen** (Genen) gibt, und dieser in Fitnessfunktion ablesbar ist.
- D.h. es muss „gute Gene“ geben.

Baustein-Hypothese (Goldberg, 1989)

(als Begründung für Crossover)

Genetische Algorithmen verwenden einfache

Chromosomenbausteine und sammeln und mischen diese um die eigene Fitness zu optimieren.

Hypothese: das Zusammenmischen vieler guter (kleiner) Bausteine ergibt das fitteste Individuum

Parameter einer Implementierung

- Üblicherweise jede Menge einstellbarer Parameter
- Welche Operatoren werden verwendet? (Mutation, Crossover, ...)
- Welche Wahrscheinlichkeiten werden in den Operatoren verwendet? Mutationswahrscheinlichkeit, ...
- Welcher Zusammenhang soll zwischen Fitnessfunktion und Selektionswahrscheinlichkeit bestehen?
- Wie groß ist die Population? Je größer, desto breiter ist die Suche angelegt, allerdings dauert es dann auch länger.
- Werden Individuen ersetzt oder dürfen sie weiterleben?
- Gibt es Kopien von Individuen in der Population?
- ...

Annahme:

- Nur Klonen, keine Rekombination, keine Mutation

Frage: Wie wächst die Häufigkeit einzelner Gene von Generation zu Generation?

- Darstellung eines Gens in der **Schema-Theorie**:
 $G = [* * * * 1 0 1 0 1 * * * *]$.
- 1 0 1 0 1 mit seiner Position ist das Gen
- V die Gesamtpopulation
- S die Menge mit Gen G
- Annahme: Fitness jedes Individuums in S : a
- Annahme: Fitness jedes Individuums in $V \setminus S$: b

Erwartungswert der Anzahl der Nachkommen aus der Menge S ?

Annahme: nächste Generation wird durch $|V|$ -maliges Ziehen aus der Menge V (mit Zurücklegen) berechnet.

Wahrscheinlichkeit: Gewichtung mit der Fitness

- Wahrscheinlichkeit ein Individuum aus S zu ziehen:

$$p(S) = \frac{a*|S|}{a*|S|+b*(|V|-|S|)}$$

- Wahrscheinlichkeit ein Individuum aus $V \setminus S$ zu ziehen:

$$p(V \setminus S) = \frac{b*(|V|-|S|)}{a*|S|+b*(|V|-|S|)}$$

- Erwartungswert für Größe des neuen S : $p(S) * |V|$
- Wachstum, wenn $p(S) * |V| > S$
- relativer Anteil: $rel(S) = \frac{|S|}{|V|}$

Statistische Analyse (4)

Mit Generationen

- $S(t)$: S in Generation t
- $rel(S(t))$: rel. Anteil von S in Generation t

Rekursionsgleichung für $rel(S(t))$:

$$\begin{aligned}
 rel(S(t+1)) &= \frac{p(S(t)) * |V|}{|V|} = p(S(t)) = \frac{a * |S(t)|}{a * |S(t)| + b * (|V| - |S(t)|)} \\
 &= \frac{a * |S(t)|}{a * |S(t)| + b * (|V| - |S(t)|)} * \frac{1}{\frac{|V|}{|V|}} = \frac{\frac{a * |S(t)|}{|V|}}{\frac{a * |S(t)| + b * (|V| - |S(t)|)}{|V|}} \\
 &= \frac{a * \frac{|S(t)|}{|V|}}{a * \frac{|S(t)|}{|V|} + b * \left(\frac{|V|}{|V|} - \frac{|S(t)|}{|V|}\right)} = \frac{a * rel(S(t))}{a * rel(S(t)) + b * (1 - rel(S(t)))}
 \end{aligned}$$

Statistische Analyse (5)

$$rel(S(t+1)) = \frac{a * rel(S(t))}{a * rel(S(t)) + b * (1 - rel(S(t)))}$$

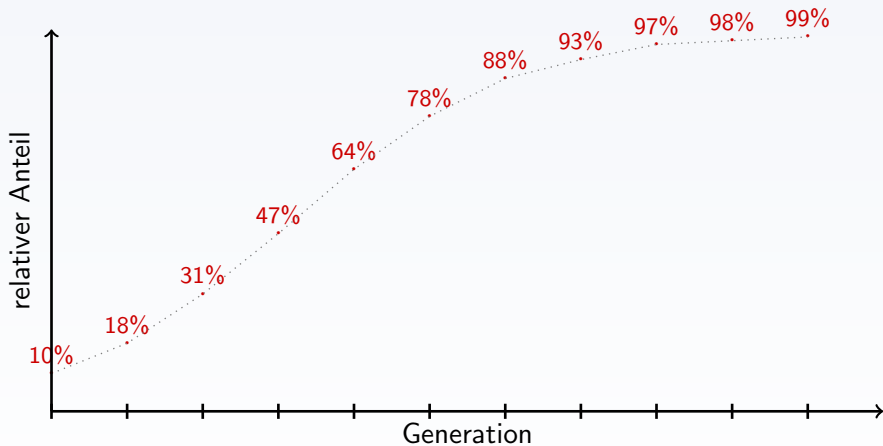
Beispiel: Wenn $a = 2b$ ergibt das

$$rel(S(t+1)) = \frac{2 * b * rel(S(t))}{2 * b * rel(S(t)) + b * (1 - rel(S(t)))} = \frac{2 * rel(S(t))}{rel(S(t)) + 1}$$

Mit $rel(S(1)) = 0.1$:

t	1	2	3	4	5	6	7	8	9	10	11
$rel(S(t))$	0,1	0,18	0,31	0,47	0,64	0,78	0,88	0,93	0,97	0,98	0,99

Bild dazu



Evolutionäre Algorithmen für Optimierungen von komplexen Problemstellungen, die

- keinen einfachen Lösungs- oder Optimierungsalgorithmus haben,
- bei denen man relativ leicht sagen kann, wie gut diese Lösungen sind
- die nicht in Echt-Zeit lösbar sein müssen, d.h. man hat ausreichend (Rechen-)Zeit um zu optimieren.
- bei denen man aus einer (bzw. zwei) (Fast-)Lösungen neue Lösungen generieren kann.
- wenn man die Problemstellung leicht mit weiteren Parametern versehen will, und trotzdem noch optimieren können will.

Weitere Anwendungsbeispiele

SAT-Solving

- Gegeben: Aussagenlogische Formeln in KNF
- $(\underbrace{l_{1,1} \vee l_{1,2} \dots l_{1,n_1}}_{\text{Klausel}}) \wedge \dots \wedge (\underbrace{l_{m,1} \vee l_{m,2} \dots l_{m,n_m}}_{\text{Klausel}})$
- Modell: Jede Klausel muss erfüllt sein

Kodierung als evolutionärer Algorithmus:

- Individuum: $[0,1,\dots]$ Belegung der Variablen (geordnet) = Interpretation
- Fitness: Anzahl der erfüllten Klauseln (zwischen 0 und m)
- Mutation: Flippe Belegung einer Variablen
- Rekombination: Zerschneide zwei Interpretationen zu einer neuen

Weitere Anwendungsbeispiele (2)

Handlungsreisenden-Problem

- n Städte mit Verbindungen
- Finde kürzestet Tour, die alle Städte einmal besucht

Evolutionäre Algorithmen

- Viele Kodierungen und Operatoren
- Z.B. Individuum: Folge aller Städte (Permutation)
- Fitnessfunktion: Weglänge (negativ)
- Mutation: Muss Permutation erhalten, daher z.B. tauschen
- Rekombination: Muss Reparaturmechanismus haben
- Ein vorgeschlagener Operator ist z.B. das partielle Invertieren:
Invertiere Teiltour

Weitere Anwendungsbeispiele (3)

Verteil- und Planungsprobleme

- Verteilen von Studenten auf Übungsgruppen
Programm von Björn Weber (ehemaliger Student)
- usw.

Alte Übungsaufgabe

Betrachten Sie das folgende Optimierungsproblem:

- Zu einer Lehrveranstaltung finden m Übungsgruppen statt.
- Es gibt n Studierende, die auf die Übungsgruppen verteilt werden sollen.
- Maximal k Studierende dürfen an jeder Übungsgruppe teilnehmen.
- Jeder Studierende gibt eine Liste seiner Wünsche ab, indem er alle m Gruppen in eine Reihenfolge bringt (d.h. eine Permutation der Zahlen von 1 bis m). Die erste Zahl hat dabei die größte Priorität.

Kodieren Sie das Optimierungsproblem als Problem eines Evolutionären Algorithmus, indem Sie

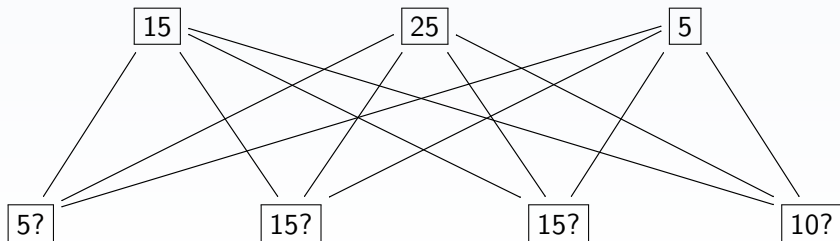
- die Beschreibung eines Individuums (Chromosomensatzes),
und
- die Fitnessfunktion

angeben.

Beispiel: Transportproblem (Michalewicz (1992))

- 3 Warenlager: L_1, L_2, L_3
- 4 Zielpunkten Z_1, \dots, Z_4 .
- Kosten für den Transport von Waren von L_i nach Z_i pro Einheit.

Lagerbestand			Warenwünsche			
L_1	L_2	L_3	Z_1	Z_2	Z_3	Z_4
15	25	5	5	15	15	10



Beispiel (2)

Die Kosten pro Einheit sind:

	Z_1	Z_2	Z_3	Z_4
L_1	10	0	20	11
L_2	12	7	9	20
L_3	0	14	16	18

Eine optimale Lösung ist:

	5	15	15	10
15	0	5	0	10
25	0	10	15	0
5	5	0	0	0

Die Kosten sind:

$$5 * 0 + 10 * 11 + 10 * 7 + 15 * 9 + 5 * 0 = 315$$

Beispiel (3)

Kodierung als evolut. Algorithmus

Zwei Varianten:

- 1 Lösungsmatrix als Individuum.
- 2 Individuum: Matrix wie Lösungsmatrix aber Prioritäten 1,2,3,... als Einträge.

Fitness: Kosten (negativ)

- 1 exakte Kodierung, aber viele verschiedene Individuen, ungültige Individuen!
- 2 alle Kodierungen gültig, Permutationen der Zahlen 1 bis 12

Mutation und Crossover:

- 1 verschiebe eine Wareneinheit auf eine andere Fuhre. Kann ungültig sein. Crossover fast immer ungültig
- 2 Austausch von 2 Prioritäten, Crossover: braucht Reparatur
Inversion: invertiere die Reihenfolge der Prioritäten:

Programme:

- Es gibt fertige generische Implementierungen mit
- Eingabe der Kodierung, Fitness
- und Parametereinstellungen

Schlussbemerkungen (2)

Evolutionäre Algorithmen

- für komplexe nicht gut verstande Probleme
- Wahl der Fitnessfunktion bestimmt die Güte

Wenn man das Problem besser versteht:

- kann man mit spezialisierter Suche / Verfahren meistens die evolutionären Algorithmen schlagen
- da man mehr Wissen hat...