Intelligente Systeme

Einführung

Prof. Dr. Rudolf Kruse Georg Ruß
Christian Moewes

{kruse,russ,cmoewes}@iws.cs.uni-magdeburg.de

Arbeitsgruppe Computational Intelligence
Institut für Wissens- und Sprachverarbeitung
Fakultät für Informatik
Otto-von-Guericke Universität Magdeburg

Evolutionäre Algorithmen

- Folien in Anlehnung an
  - Karsten Weicker [Weicker, 2007], `http://www.evolutionary-algorithms.de`
  - Christian Borgelt, `http://www.borgelt.net/slides/ga.pdf`

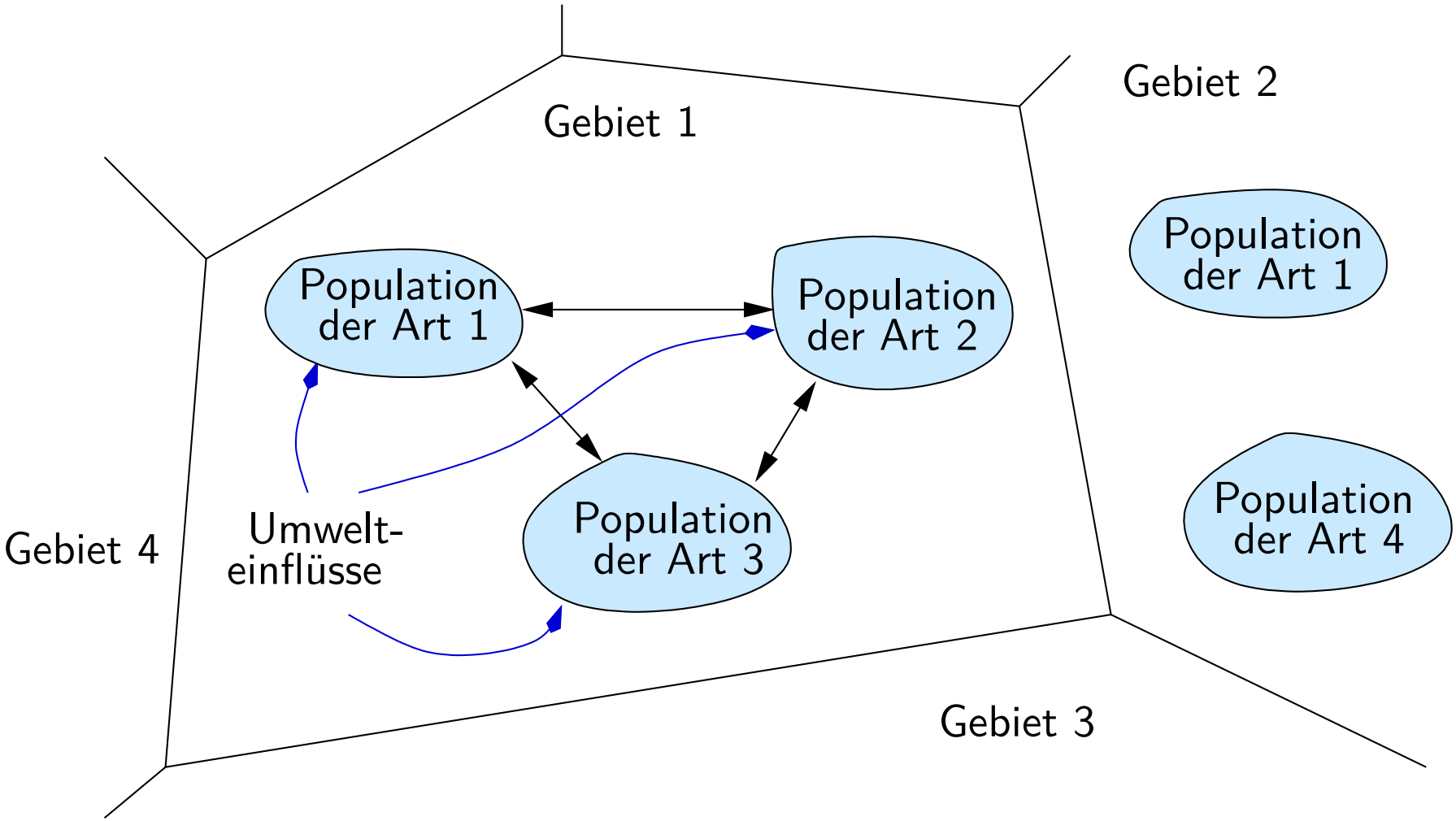
- weitere Literatur: [Gerdes et al., 2004]

Gliederung der Vorlesung

Evolution in der Biologie

- Theorie, daß sich Lebewesen allmählich aus einfachen zu immer komplizierteren Formen entwickelt haben
- reine Beobachtung und Hypothesenbildung durch Lamarck (1809) und Darwin (1859)
- wissenschaftliche Erklärungen auf der Ebene der Populationen in der Populationsgenetik (1908)
- Erklärungen aus der Molekulargenetik durch Watson und Crick (1953)

Biologisches Modell



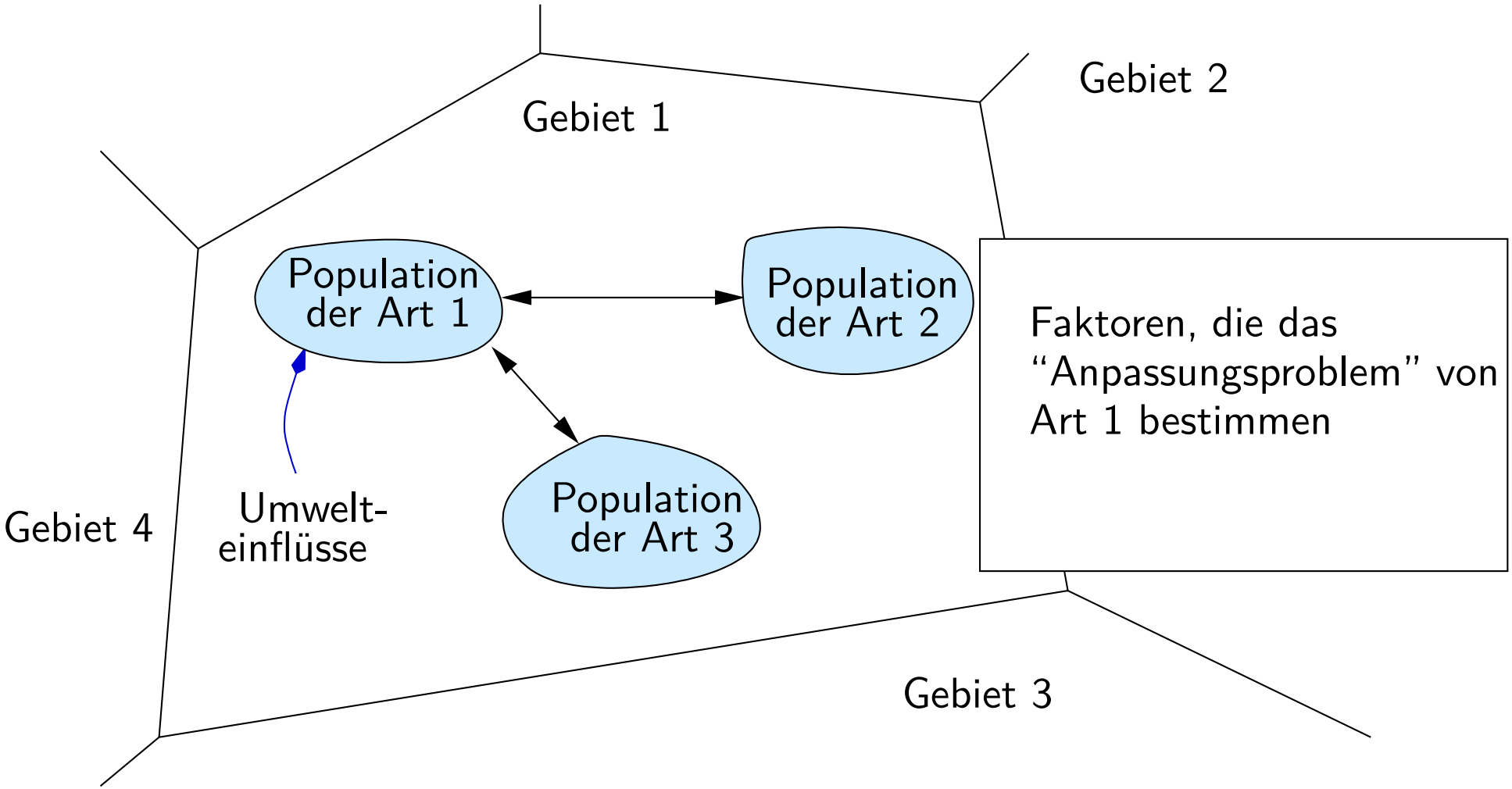
Biologisches Modell

langfristige Anpassung von:

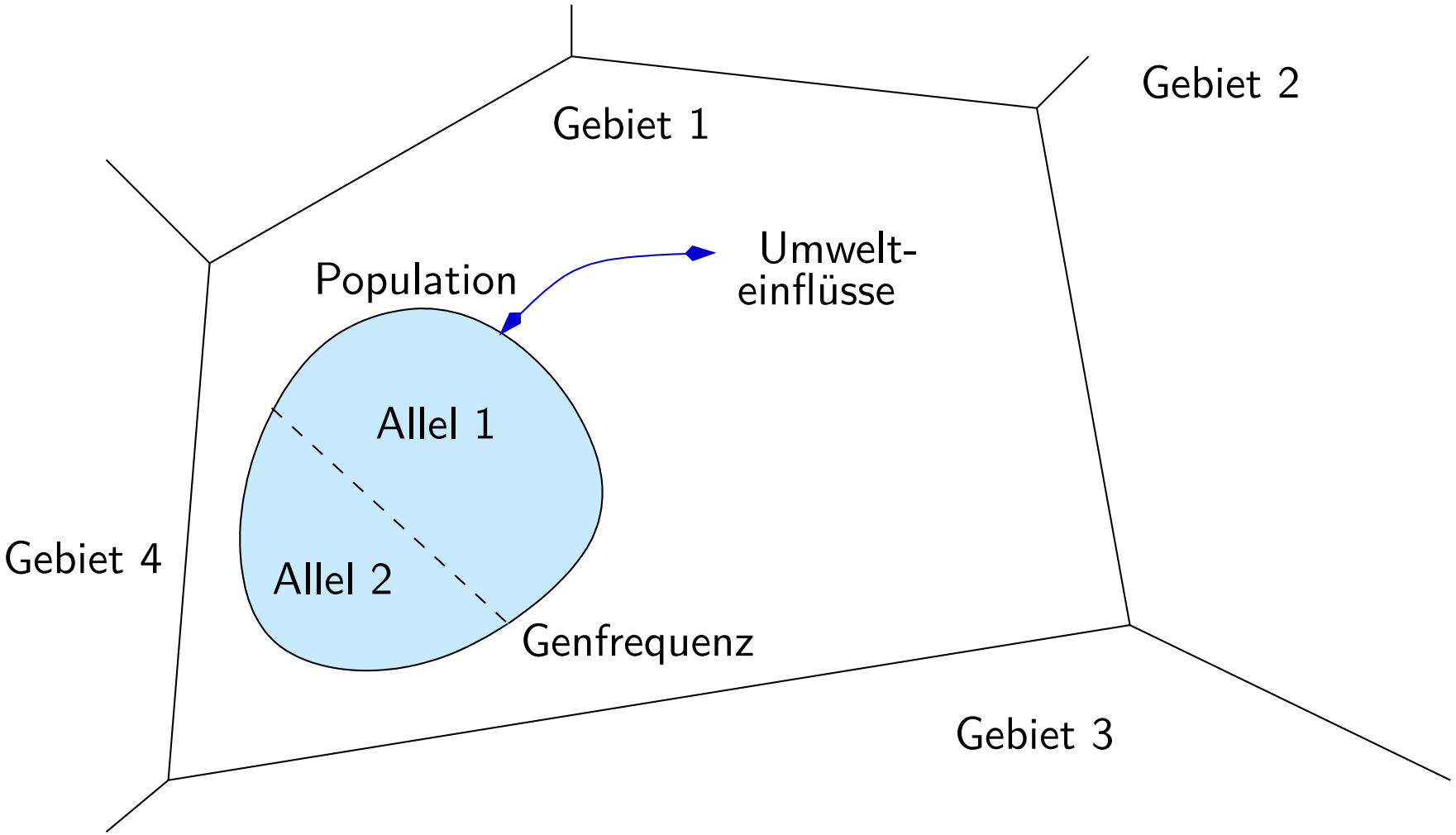
- Nahrungsaufnahme
- Partnerfindung
- Fortpflanzung
- Tarnung

führt zur Entwicklung hochkomplexer Lösungen

Biologisches Modell



Biologische Evolutionsfaktoren



Biologische Evolutionsfaktoren

Evolution ist nur dann möglich, wenn sich die Genfrequenz ändert.
Wann ist das der Fall?

Wenn Faktoren wie

- Mutation
- Selektion
- Genfluß
- Gendrift

auftreten.

Faktor: Mutation

- direkte Änderung der Genfrequenz durch kleine Veränderungen an Individuen auf der molekulargenetischen Ebene
- natürlicher Vervielfältigungsfehler
- Mutationsrate beim Menschen: 10^{-10}, d.h. mit 10^5 Genen mit 10^4 Bausteinen bei jeder zehnten Zellteilung
- meist sind nur kleine Mutationen überlebensfähig

Faktor: Selektion

- Änderung der Genfrequenz durch unterschiedlich viele Nachkommen der einzelnen Individuen
- Fitness eines Allels: $\frac{\text{Anzahl Nachkommen}}{\text{Anzahl Nachkommen des besten Allels}}$
- Unterschiedliche Ursachen:
  - Überlebenschancen
  - Fruchtbarkeit
  - Fähigkeit, einen Partner zu finden
  - Länge der Generationsdauer

Faktoren: Genfluß und Gendrift

- Genfluß:
  - Migration zwischen sonst getrennten Populationen
  - führt zur Änderung der Genfrequenz
  - wird bei Standard-EA nicht imitiert, ist allerdings auf Parallelrechnern relevant
- Gendrift:
  - große Populationen sind stabil bezüglich Zufallsereignissen
  - in kleinen Populationen: zufälliges "Driften" des gesamten Genpools
  - ist ein Faktor bei EA, wird allerdings nicht gezielt eingesetzt und ist eher negativ besetzt

Faktor: Rekombination (Crossover)?

- gemäß der Populationsgenetik ist sie kein Faktor, da in großen Populationen die Genfrequenz gleich bleibt
- Die Sichtweise der Populationsgenetik ist mechanistisch: Gene sind ein Bauplan, wobei die Rekombination sie nur neu zusammenstellt
- modernere Sichtweise: genetisches Netzwerk, wobei Gene abhängig von anderen Genen aktiv sind
- selbstorganisierter, zyklischer "Wachstumsprozess", in dem Rekombination neue Zusammenhänge erschaffen kann
- Benutzung der Rekombination in EA meist durch zufälliges Durchprobieren anderer Kombinationen, wobei die Selbstorganisation unberücksichtigt bleibt

Faktor: Anpassung – Nischenbildung, Koevolution

- Nischenbildung:
  - jede Art belegt eine eigene Nische
  - aufgrund innerartlicher Konkurrenz können sich Arten aufspalten (einnischen)
  - ist in EA nicht üblich, kann allerdings als spezielle Technik bei Mehrzieloptimierung verwendet werden
- Koevolution
  - ökologische Beziehungen zwischen den Arten
    - Konkurrenz
    - Räuber vs. Beute
    - Wirt vs. Parasit
    - Symbiose
  - nicht in Standard-EA
  - Algorithmen mit den verschiedenen ökologischen Beziehungen existieren

Zusammenfassung

Zunächst bleiben unberücksichtigt:

- dynamische Populationsgrößen
- selbstorganisierte Interpretation der DNA
- Sexualität
- Nischenbildung
- Koevolution
- Genfluß
- dynamische Umgebungen

Gliederung der Vorlesung

1. Biologischer Hintergrund
2. **Modell: Evolutionäre Algorithmen (EA)**
3. Probleme und Algorithmen
4. Beispiele für evolutionäre Algorithmen
5. Genetische Programmierung

Modell: evolutionäre Algorithmen (EA)

Evolutionäre Algorithmen sind ein Optimierungsverfahren, das als Vorbild die biologische Evolution hat.

- Optimierungsproblem ersetzt bzw. modelliert die Umweltfaktoren
- Individuen sind Lösungskandidaten
- komplexe biologische Zusammenhänge wie beispielsweise
  - Verwandtschaften
  - Konkurrenz
  - überlappende Lebenszeiten

  müssen modelliert werden

Bestandteile eines EA (I)

- Kodierungsvorschrift für die Lösungskandidaten:
  - Wie die Lösungskandidaten kodiert werden, ist problemspezifisch; es gibt daher keine allgemeinen Regeln. Im Verlauf der Vorlesung werden allerdings einige Aspekte besprochen, die man bei der Wahl einer Kodierung beachten sollte.
- Anfangspopulation
  - Meist werden einfach zufällige Zeichenketten erzeugt. Je nach gewählter Kodierung können aber auch komplexere Verfahren nötig sein.

Bestandteile eines EA (II)

- Bewertungsfunktion für die Individuen
  - Die Bewertungs- oder Fitnessfunktion spielt die Rolle der Umgebung und gibt die Güte der Individuen an. Meist ist die Bewertungsfunktion mit der zu optimierenden Funktion identisch. Sie kann aber auch zusätzliche Elemente enthalten, die einzuhaltende Nebenbedingungen darstellen.
- Auswahlfunktion
  - Auf der Grundlage der Fitnessfunktion bestimmt die Auswahlfunktion, welche Individuen zur Erzeugung von Nachkommen herangezogen werden oder auch unverändert in die nächste Generation gelangen.

Bestandteile eines EA (III)

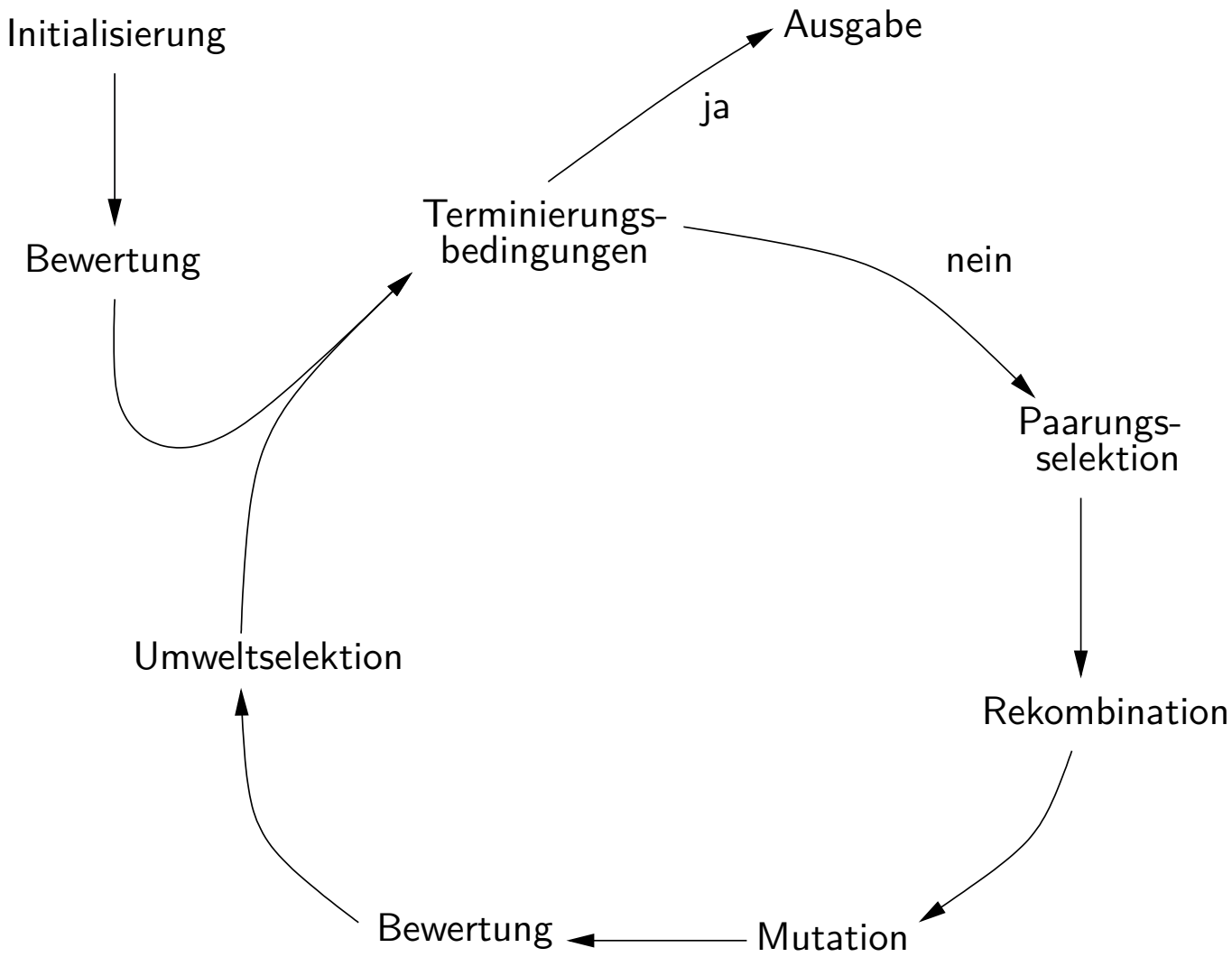
- genetische Operatoren: diese ändern die Lösungskandidaten
  - Mutation – zufällige Veränderung einzelner Gene
  - Crossover – Rekombination von Chromosomen
- Abbruchkriterium, z.B.
  - - eine festgelegte Anzahl von Generationen wurde berechnet,
  - - eine festgelegte Anzahl von Generationen lang gab es keine Verbesserung,
  - - eine vorgegebene Mindestlösungsgüte wurde erreicht
- Darüberhinaus sind für verschiedene Teile des Algorithmus Parameter festzulegen, wie beispielsweise Populationsgröße, Mutationswahrscheinlichkeit etc.

Ablauf des EA

1. **procedure** evolution-program;
2. **begin**
3. $t \leftarrow 0$;
4. initialize pop(t);
5. evaluate pop(t);
6. **while not** termination criterion **do**
7. $t \leftarrow t + 1$;
8. select pop(t) from pop(t-1);
9. alter pop(t);
10. evaluate pop(t);
11. **end**
12. **end**

- Durch die Auswahl wird eine Art "Zwischenpopulation" von Individuen mit (im Durchschnitt) hoher Fitness erzeugt.
- Nur die Individuen der Zwischenpopulation können Nachkommen bekommen.

zyklischer Ablauf des EA



Selektion

Prinzip: Bessere Individuen (gemäß Fitness) sollen mit höherer Wahrscheinlichkeit Nachkommen haben.

- Glücksradauswahl: Sektorgrößen des Glücksrads entsprechen den relativen Fitnesswerten der Individuen.
  - Drehe das Glücksrad.
  - Wähle das Individuum, dessen Sektor unter der Markierung liegt.
  - Wiederhole die Selektion so oft, wie es Individuen in der Population gibt.
- Rangauswahl
  - Sortiere die n Individuen I nach ihrer Fitness
  - Bestimme neue Fitness: $\mathrm{fitness}(I_n) = n - \mathrm{rank}(I_n) + 1$
  - Behebt Probleme der Glücksradauswahl bei stark unterschiedlicher Fitness
- Turnierauswahl
  - Wähle zufällig n Individuen aus der Population aus.
  - Aus dieser Auswahl kommt das Individuum mit der höchsten Fitness weiter.
  - Wiederhole, bis die neue Generation groß genug ist.

Zusammenfassung

Im Folgenden:

- Betrachtung individueller Lösungsversuche
- Populationskonzept
- randomisierte Operationen
  - Mutation
  - Selektion
- Lösungskandidaten unterliegen einer Selektion
- Güte/Fitness stehen in einem Zusammenhang zur Weitervererbung

Gliederung der Vorlesung

Optimierungsprobleme

Beispiele

- kürzeste Route für einen Briefträger
- Auftragsreihenfolge für ein Fließband
- Plazierung von Mobilfunkantennen
- adaptive Ampelsteuerung
- Stundenplanung
- Regression

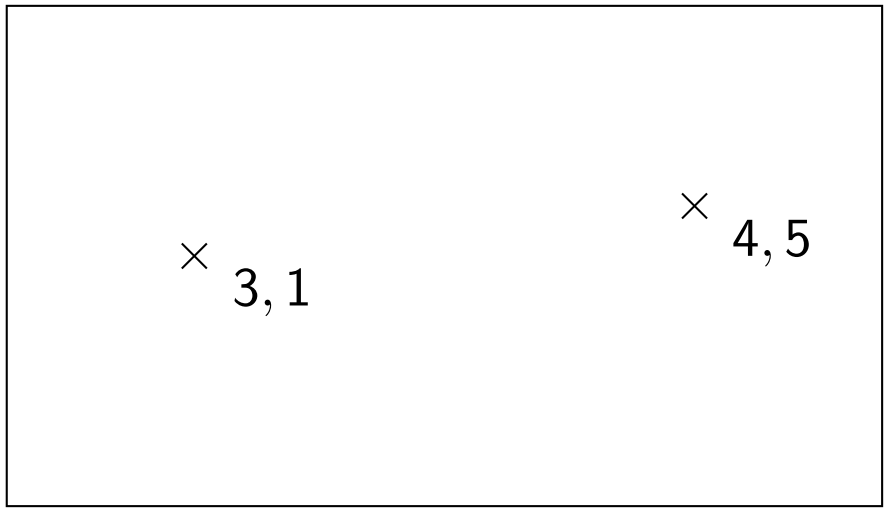
Optimierungsprobleme

weitere Beispiele

- Gradientenabstieg (z.B. bei Neuronalen Netzen)
- Simplex/lineares Programmieren (Operations Research)
- spezielle Algorithmen (Dijkstra, Primfaktoren, A*)
- Backtracking, Branch & Bound

Black-Box-Optimierung

- Jeder Punkt im Suchraum wird mit einer Zahl aus $\mathbb{R}$ assoziiert
- gesucht: Maximum
- Annahme: gewisse Glattheit des Problems
- Suchraum:



Gliederung der Vorlesung

1. Biologischer Hintergrund

2. Modell: Evolutionäre Algorithmen (EA)

3. Probleme und Algorithmen

4. **Beispiele für evolutionäre Algorithmen**
   - Das Problem des Handlungsreisenden
   - n-Damen-Problem
   - Das 0/1-Rucksackproblem
   - Optimierung neuronaler Netze

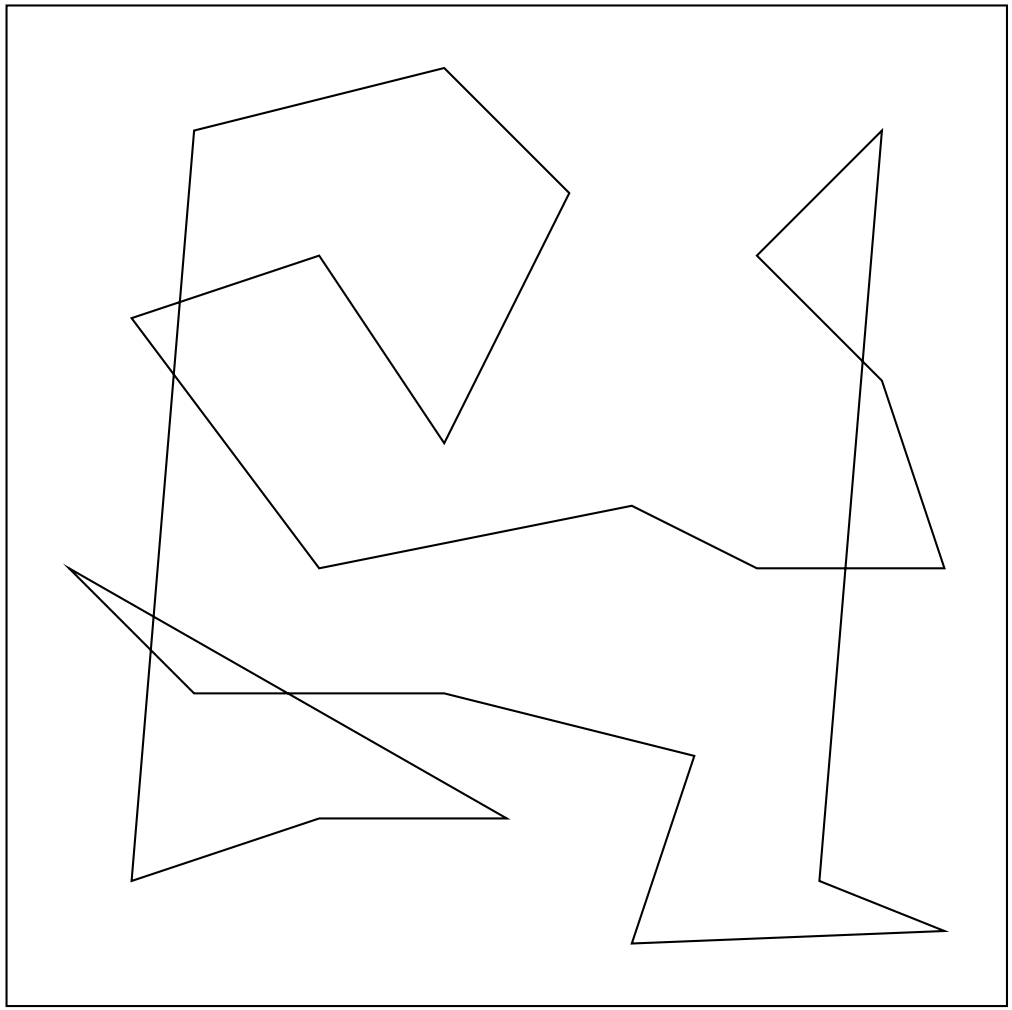
5. Genetische Programmierung

Das Problem des Handlungsreisenden

Definition

- Geg: Graph $G = (V, E, \gamma)$ mit "Straßen" $E \subseteq V \times V$ und "Fahrzeit" $\gamma : E \to \mathbb{R}$
- *Handlungsreisendenproblem* ist $(\mathcal{S}_n, f_{TSP}, <)$ mit $\mathcal{S}_n$ Raum aller Permutationen
- Bewertungsfunktion $f_{TSP}((i_1, \cdots, i_n)) = \gamma((v_{i_n}, v_{i_1})) + \sum_{j=2}^{n} \gamma((v_{i_{j-1}}, v_{i_j}))$
- heißt symmetrisch, wenn für alle $(v_i, v_j) \in E$ sowohl $(v_j, v_i) \in E$ als auch $\gamma((v_i, v_j)) = \gamma((v_j, v_i))$ erfüllt sind.

Beispieltour TSP



(so könnte ein Individuum der Population aussehen; die Individuen werden im Folgenden mit A, B, C bezeichnet)

Mutation: Kanten vertauschen

1. $B \leftarrow A$
2. $u_1 \leftarrow$ wähle Zufallszahl gemäß $U(\{1, \cdots, n\})$
3. $u_2 \leftarrow$ wähle Zufallszahl gemäß $U(\{1, \cdots, n\})$
4. $B_{u_1} \leftarrow A_{u_2}$
5. $B_{u_2} \leftarrow A_{u_1}$
6. **return** B

Mutation: Invertieren eines Teilabschnitts

Invertierende Mutation für Permutation $A = (A_1, \ldots, A_n)$

1. $B \leftarrow A$
2. $u_1 \leftarrow$ wähle Zufallszahl gemäß $U(\{1, \cdots, n\})$
3. $u_2 \leftarrow$ wähle Zufallszahl gemäß $U(\{1, \cdots, n\})$
4. **if** $u_1 > u_2$
5. **then** vertausche u_1 und u_2
6. **for** $j \in \{u_1, \cdots, u_2\}$
7. **do** $B_{u_2+u_1-j} \leftarrow A_j$
8. **return** B

Ordnungsrekombination

Ordnungsrekombination für Permutation $A = (A_1, \ldots, A_n)$

1. $j \leftarrow$ wähle zufällig gemäß $U(\{1, \cdots, n-1\})$
2. **for** $i \in \{1, \cdots, j\}$
3. **do** $C_i \leftarrow A_i$
4. **for** $i \in \{1, \cdots, n\}$
5. **do if** $B_i \notin \{C_1, \cdots, C_j\}$
6. **then** $j \leftarrow j + 1$
7. $C_j \leftarrow B_i$
8. **return** C

Kantenrekombination

Kantenrekombination für $A = (A_1, \ldots, A_n)$ und $B = (B_1, \ldots, B_n)$
Menge Adj der „benachbarten“ Punkte

1. **for** Knoten $k \in \{1, \cdots, n\}$
2. **do** $Adj(k) \leftarrow \emptyset$
3. **for** $i = 1, \cdots, n$
4. **do** $Adj(A_i) \leftarrow Adj(A_i) \cup \{A_{\text{i mod n})+1}\}$
5. $Adj(A_{(\text{i mod n})+1}) \leftarrow \{A_i\} \cup Adj(A_{(\text{i mod n})+1})$
6. $Adj(B_i) \leftarrow Adj(B_i) \cup \{B_{\text{i mod n})+1}\}$
7. $Adj(B_{(\text{i mod n})+1}) \leftarrow \{B_i\} \cup Adj(B_{(\text{i mod n})+1})$
8. $C_1 \leftarrow$ wähle zufällig gemäß $U(\{A_1, B_1\})$
9. **for** $i = \{1, \cdots, n-1\}$
10. **do** $K \leftarrow \{m \in Adj(C_i) | \#(Adj(m) \backslash \{C_1, \cdots, C_i\}) \text{minimal}\}$
11. **if** $K \neq \emptyset$
12. **then** $C_{i+1} \leftarrow$ wähle gleichverteilt zufällig aus K
13. **else** $C_{i+1} \leftarrow$ wähle gleichverteilt zufällig aus $\{1, \cdots, n\}$
14. **return** C

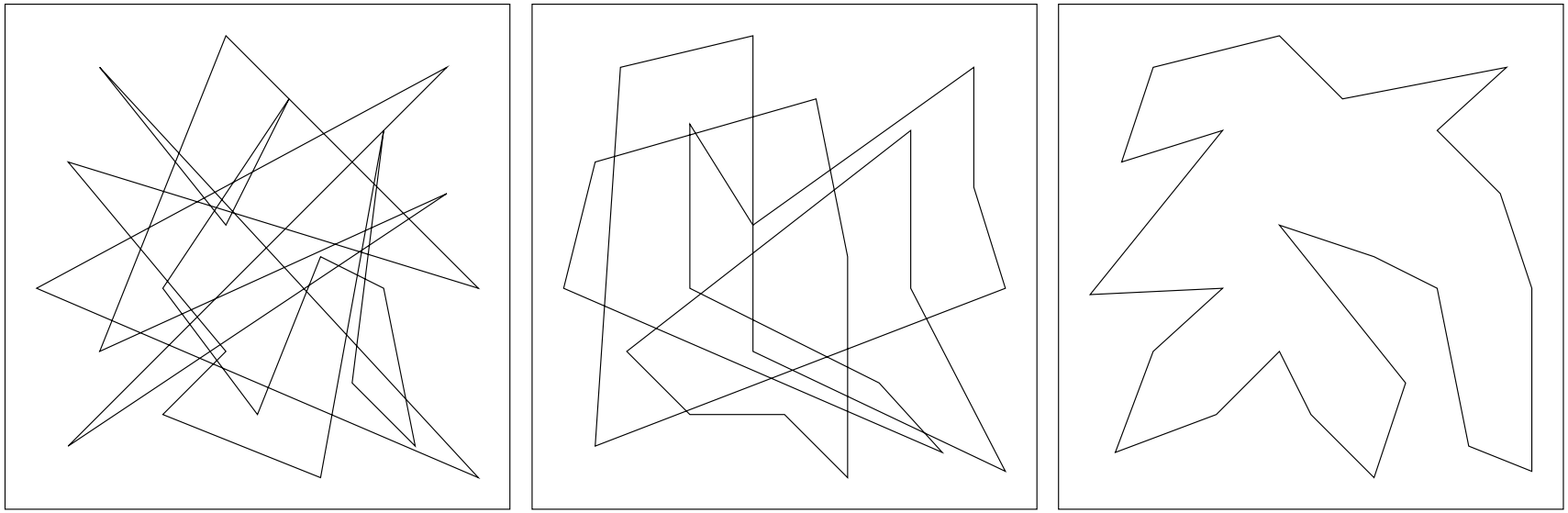
Gesamtalgorithmus

[

EA-Handlungsreisendenproblem(Zielfunktion F, Anzahl der Städte n)]

1. $t \leftarrow 0$
2. $P(t) \leftarrow$ Liste mit 10 gleichverteilt zufälligen Permutationen aus $U(\mathcal{S}_n)$
3. bewerte alle $A \in P(t)$ mit Zielfunktion F
4. **while** $t \leq 2000$
5. **do** $P' \leftarrow \langle\rangle$
6. **for** $i \in \{1, \cdots, 40\}$
7. **do** $A, B \leftarrow$ wähle gleichverteilt zufällig Eltern aus $P(t)$
8. **if** $u < 0.3$ für eine Zufallszahl $u \in U([0, 1])$
9. **then** $A \leftarrow$ Kantenrekombination(A,B)
10. $A \leftarrow$ Invertierende Mutation(A)
11. $P' \leftarrow P' \circ \langle A \rangle$
12. bewerte alle $A \in P'$ mit Zielfunktion F
13. $t \leftarrow t + 1$
14. $P(t) \leftarrow$ 10 beste Individuen aus $P' \circ P(t-1)$
15. **return** bestes Individuum aus $P(t)$

Ablauf des EA, am Beispiel



- Anfangs-Individuum, nach 500 Generationen, nach 1000 Generationen
- Je nach Anwendungsfall kann es sinnvoll sein, nur einige der verfügbaren Operatoren zusammen anzuwenden und die Performance der so entstehenden verschiedenen Algorithmen vergleichend zu bewerten.

Beispiel: Das n-Damen-Problem

- Das Problem besteht darin, n Damen auf einem $n \times n$-Schachbrett so zu plazieren, daß in keiner Reihe, Linie und Diagonale mehr als eine Dame steht. Anders ausgedrückt: die Damen dürfen einander nicht im Weg stehen.
- Wie aus anderen Vorlesungen vielleicht bekannt, kann das n-Damen-Problem leicht mit Hilfe von Backtracking gelöst werden.
- Da sich an diesem Problem einige Aspekte der evolutionären Algorithmen gut verdeutlichen lassen, wird es als Übungsaufgabe vertiefend behandelt.

Beispiel: 0/1-Rucksackproblem

- Wie packe ich meinen Rucksack möglichst effektiv, wenn die Gegenstände, die ich gerne mitnehmen möchte, nicht alle in den Rucksack passen? (Variante mit beispielsweise 100 Gegenständen)
- Sehr komplexes, NP-vollständiges Problem, 2^{100} grundsätzliche Möglichkeiten.
- Klassische Berechnung daher nicht in vertretbarer Zeit möglich.
- Anwendung von Heuristiken und approximierten Ergebnissen, die möglichst nahe am Optimum sind.
- Hier: Verwendung eines evolutionären Algorithmus'.

Beispiel: 0/1-Rucksackproblem

Problem Es soll eine Menge von Gegenständen so verpackt werden, daß ihre Gesamtmasse einen vorgeschriebenen Wert nicht überschreitet und der Gesamtwert möglichst hoch ist. Jeder Gegenstand verfügt über einen individuellen Wert.

Lösung Kodierung der Mitnahme von Gegenständen mit Hilfe eines Vektors aus Binärzahlen, der das Individuum darstellt.

Nr.	0	1	2	3	4	5	6	7	8	9	
	1	0	1	1	0	0	0	1	0	1	Individuum/Chromosom
Masse	3	4	7	1	12	9	7	8	13	6	Gesamtmasse (=25)
Wert	4	1	2	8	4	5	3	9	11	4	Gesamtwert (=27)

Beispiel: 0/1-Rucksackproblem

- Mutation: z.B. Aus- oder Einpacken eines Gegenstands
- vorher:

Nr.	0	1	2	3	4	5	6	7	8	9	
	1	0	1	1	0	0	0	1	0	1	Individuum/Chromosom
Masse	3	4	7	1	12	9	7	8	13	6	Gesamtmasse (=25)
Wert	4	1	2	8	4	5	3	9	11	4	Gesamtwert (=27)

- nachher:

Nr.	0	1	2	3	4	5	6	7	8	9	
	1	0	1	1	**1**	0	0	1	0	1	Individuum/Chromosom
Masse	3	4	7	1	**12**	9	7	8	13	6	Gesamtmasse (=37)
Wert	4	1	2	8	**4**	5	3	9	11	4	Gesamtwert (=31)

Beispiel: 0/1-Rucksackproblem

- Crossover: Kreuzen zweier Individuen

Nr.	0	1	2	3	4	5	6	7	8	9	
	1	0	1	1	1	**0**	**0**	**1**	**0**	**1**	Individuum 1
	0	**1**	**0**	**1**	**1**	0	1	0	1	1	Individuum 2
	0	**1**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	neues Individuum a
	1	0	1	1	1	0	1	0	1	1	neues Individuum b
Masse(a)	3	4	7	1	12	9	7	8	13	6	Gesamtmasse (=31)
Wert(a)	4	1	2	8	4	5	3	9	11	4	Gesamtwert (=26)
Masse(b)	3	4	7	1	12	9	7	8	13	6	Gesamtmasse (=49)
Wert(b)	4	1	2	8	4	5	3	9	11	4	Gesamtwert (=36)

- Fitness: z.B. Gesamtwert der eingepackten Gegenstände, wobei als Nebenbedingung die zulässige Gesamtmasse nicht überschritten werden darf

Beispiel: Optimierung neuronaler Netze

- Die in den vorangegangenen Vorlesungen behandelten mehrschichtigen neuronalen Netze können mit Hilfe evolutionärer Strategien optimiert werden.
- Hierbei werden beispielsweise die Parameter des Netzes zuerst grob mit Hilfe des EA an die Problemstellung angepaßt, bevor das herkömmliche Gradientenabstiegsverfahren zur Feinabstimmung eingesetzt wird.
- zu optimierende Parameter: Bias-Werte, Gewichte, aber auch die Größe der versteckten Schichten, die Aktivierungsfunktionen oder auch die Lernrate.

Gliederung der Vorlesung

Genetische Programmierung

- **bisher:** Darstellung von Lösungskandidaten/Spielstrategien durch Chromosomen fester Länge (Vektor von Genen)
- **jetzt:** Darstellung durch Funktionsausdrücke oder Programme
  - $\longrightarrow$ **genetische Programmierung**
  - $\longrightarrow$ komplexere Chromosomen variabler Länge

- formale Grundlage: Grammatik zur Beschreibung der Sprache
- Festlegung zweier Mengen
  - $\mathcal{F}$ – Menge der Funktionssymbole und Operatoren
  - $\mathcal{T}$ – Menge der Terminalsymbole (Konstanten und Variablen)
- Die Mengen $\mathcal{F}$ und $\mathcal{T}$ sind problemspezifisch.
- Sie sollten nicht zu groß sein (Beschränkung des Suchraums) und doch reichhaltig genug, um eine Problemlösung zu ermöglichen.

Beispiele zu Symbolmengen

- **Beispiel 1:** Erlernen einer Booleschen Funktion
  - $\mathcal{F} = \{\text{and}, \text{or}, \text{not}, \text{if} \ldots \text{then} \ldots \text{else} \ldots, \ldots\}$
  - $\mathcal{T} = \{x_1, \ldots, x_m, 1, 0\}$ bzw. $\mathcal{T} = \{x_1, \ldots, x_m, t, f\}$
- **Beispiel 2:** Symbolische Regression
  - Regression: Bestimmung einer Ausgleichsfunktion zu gegebenen Daten unter Minimierung der Fehlerquadratsumme – *Methode der kleinsten Quadrate*
  - $\mathcal{F} = \{+, -, *, /, \sqrt{\ }, \sin, \cos, \log, \exp, \ldots\}$
  - $\mathcal{T} = \{x_1, \ldots, x_m\} \cup \mathbb{R}$
- **beachte:** Alle Chromosomen müssen auswertbar sein.
  also: ggf. Vervollständigung des Definitionsbereichs einer Funktion, z.B.
  - $\forall x \leq 0: \quad \log(x) = 0$ oder $=$ kleinste darstellbare Fließkommazahl
  - $\tan\left(\frac{\pi}{2}\right) = 0$ oder $=$ größte darstellbare Fließkommazahl

Symbolische Ausdrücke

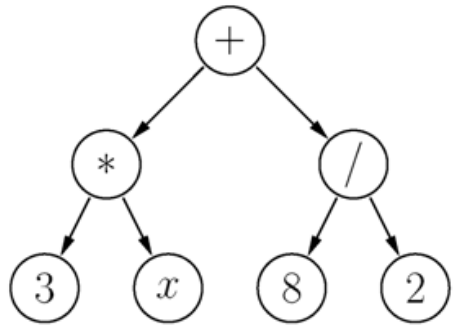
- Die Chromosomen sind nun Ausdrücke, die aus Elementen aus $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ zusammengesetzt sind (ggf. werden noch Klammern hinzugefügt).
- allerdings: Beschränkung auf „wohlgeformte“ symbolische Ausdrücke. Übliche **rekursive Definition** (Präfixnotation):
  - Konstanten- und Variablensymbole sind symbolische Ausdrücke.
  - Sind $t_1, \ldots, t_n$ symbolische Ausdrücke und ist $f \in \mathcal{F}$ ein (n-stelliges) Funktionssymbol, so ist $(f t_1 \ldots t_n)$ ein symbolischer Ausdruck.
  - Keine anderen Zeichenfolgen sind symbolische Ausdrücke.
- **Beispiele** zu dieser Definition:
  - „$(+\ (*\ 3\ x)\ (/\ 8\ 2))$“ ist ein symbolischer Ausdruck.
    Lisp- bzw. Scheme-artige Schreibweise, Bedeutung: $3 \cdot x + \frac{8}{2}$
  - „$2\ 7\ *\ (3\ /$“ ist kein symbolischer Ausdruck

Implementierung

- Für die Implementierung der genetischen Programmierung ist es günstig, symbolische Ausdrücke durch sogenannte Parse-Bäume darzustellen.
  (Parse-Bäume werden in einem Parser z.B. eines Compilers verwendet, um arithmetische Ausdrücke darzustellen und anschließend zu optimieren.)

symbolischer Ausdruck:
$(+\ (*\ 3\ x)\ (/\ 8\ 2))$

Parse-Baum:



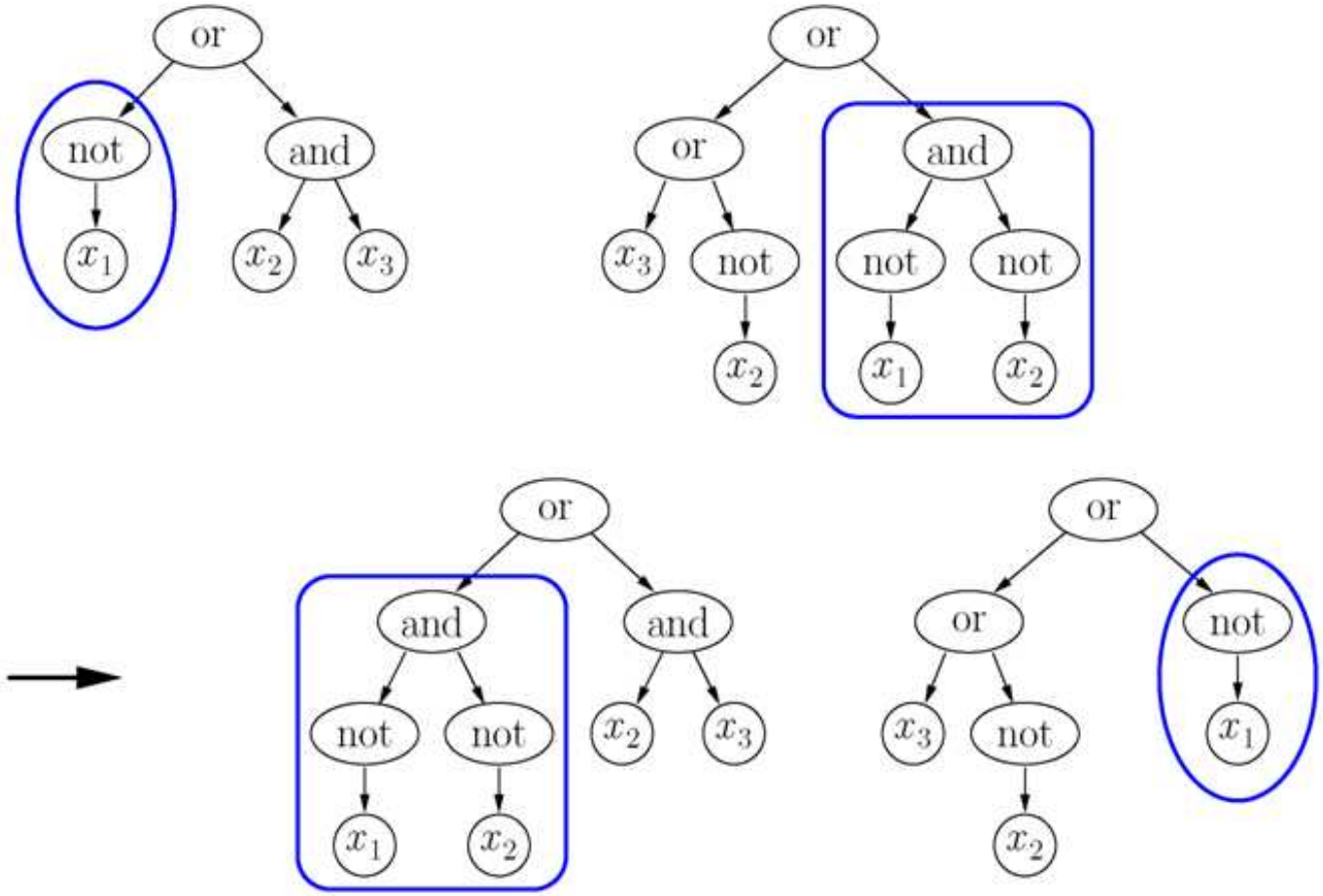
- In Lisp/Scheme sind Ausdrücke verschachtelte Listen:
  Erstes Element der Liste ist das Funktionssymbol bzw. der Operator.
  Die nachfolgenden Elemente sind die Argumente bzw. Operanden.

Ablauf einer Genetischen Programmierung

- Erzeugen einer **Anfangspopulation** zufälliger symbolischer Ausdrücke.
  Parameter des Erzeugungsprozesses:
  - maximale Verschachtelungstiefe (maximale Baumhöhe)
  - Wahrscheinlichkeit für die Wahl eines Terminalsymbols
- **Bewertung** der Ausdrücke durch Berechnung der Fitness.
  - Erlernen Boolescher Funktionen:
    Anteil korrekter Ausgaben für alle Eingaben bzw. in einer Stichprobe.
  - Symbolische Regression:
    Summe der Fehlerquadrate über die gegebenen Messpunkte.
    1-D: Daten (x_i, y_i), $i = 1, \ldots, n$, Fitness $f(c) = \sum_{i=1}^{n} (c(x_i) - y_i)^2$
- **Selektion** mit einem der besprochenen Verfahren.
- Anwendung **genetischer Operatoren**, meist nur Crossover.

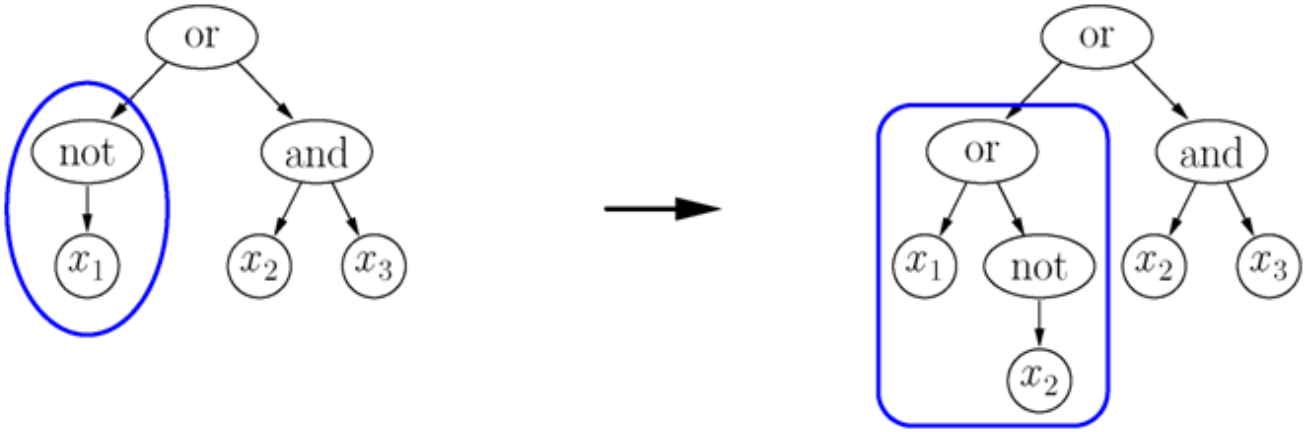
Crossover

- Crossover besteht im Austauschen zweier Teilausdrücke (Teilbäume)



Mutation

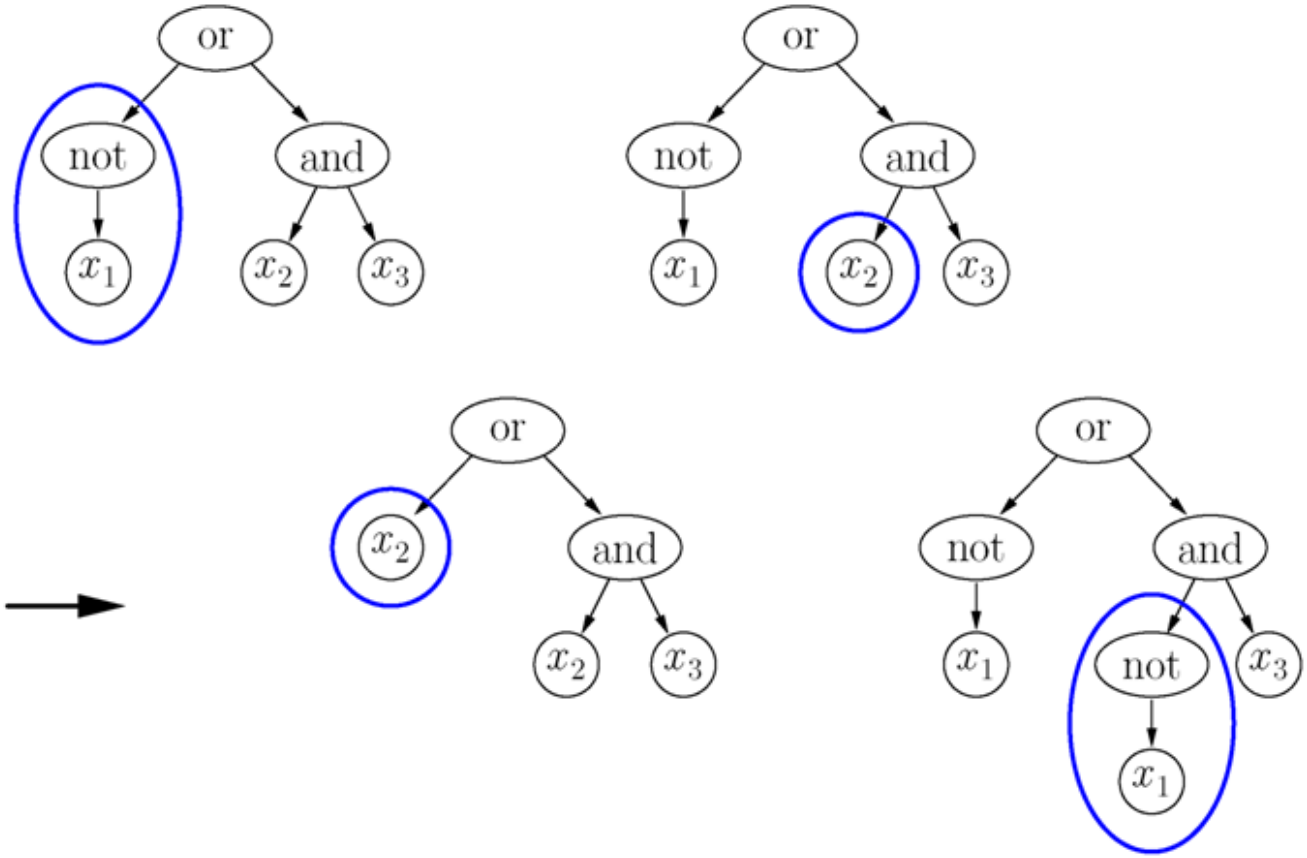
- Mutation besteht im Ersetzen eines Teilausdrucks (Teilbaums) durch einen zufällig erzeugten neuen:



- Es sollten möglichst nur kleine Teilbäume ersetzt werden.
- Meist wird nur Crossover und keine Mutation verwendet.
- Es sollte dann aber darauf geachtet werden, dass die Population groß genug ist, um einen hinreichend großen Vorrat an „genetischem Material“ zur Verfügung zu haben, aus dem neue Lösungskandidaten rekombiniert werden können.

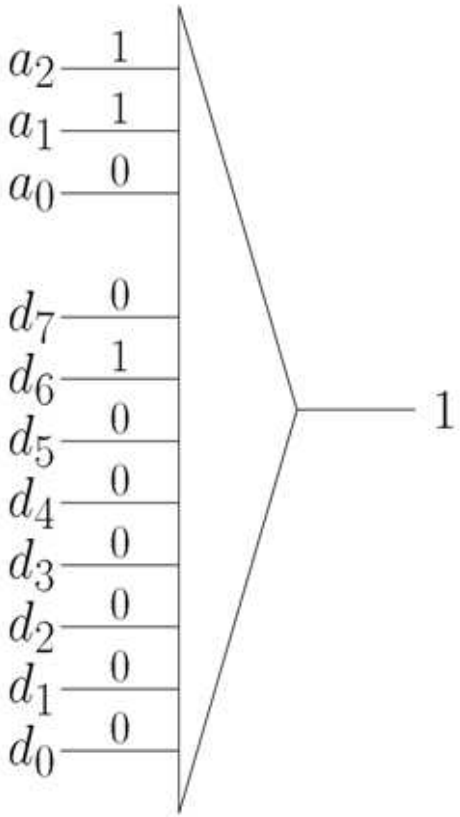
Vorteil des Crossover

- **beachte:** Crossover ist mächtiger als für Vektoren, denn ein Crossover identischer Eltern kann zu neuen Individuen führen.



Beispiel: 11-Multiplexer

Erlernen eines Booleschen 11-Multiplexers [Koza, 1992]



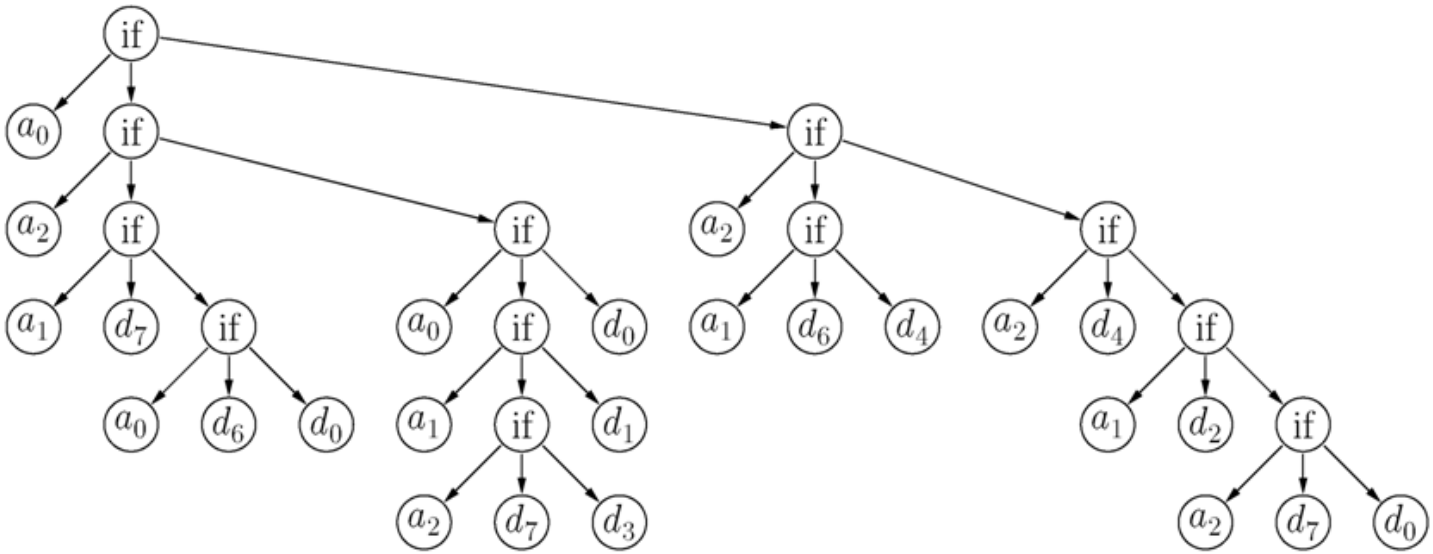
- Multiplexer mit 8 Daten- und 3 Adressleitungen (Der Zustand der Adressleitungen gibt die durchzuschaltende Datenleitung an.)
- $2^{11} = 2048$ mögliche Eingaben mit je einer zugehörigen Ausgabe
- Festlegung der Symbolmengen:
  - $\mathcal{T} = \{a_0, a_1, a_2, d_0, \ldots, d_7\}$
  - $\mathcal{F} = \{\text{and}, \text{or}, \text{not}, \text{if}\}$
- Fitnessfunktion: $f(s) = 2048 - \sum_{i=1}^{2048} e_i$, wobei e_i der Fehler für die i-te Eingabe ist.

11-Multiplexer

- Populationsgröße popsize = 4000
- Anfangstiefe der Parse-Bäume: 6, maximale Tiefe: 17
- Die Fitnesswerte in der Anfangspopulation liegt zwischen 768 und 1280. Die mittlere Fitness liegt bei 1063.
  (Der Erwartungswert liegt bei 1024, da bei zufälliger Ausgabe im Durchschnitt die Hälfte der Ausgaben richtig sind.)
- 23 Ausdrücke haben eine Fitness von 1280. Einer davon entspricht einem 3-Multiplexer: (if a_0 d_1 d_2)
- fitnessproportionale Selektion
- 90% (3600) der Individuen werden dem Crossover unterworten.
  Die Restlichen werden unverändert übernommen.

11-Multiplexer

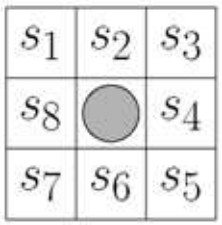
- Nach nur 9 Generationen ist folgende Lösung gefunden (Fitness 2048):



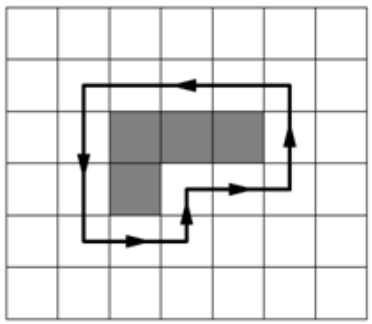
Stimulus-Response-Agent

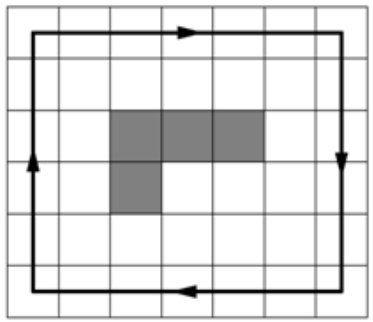
Beispiel: Erlernen eines Robotersteuerprogramms [Nilsson, 1998]

- Betrachte einen Stimulus-Response-Agenten in einer Gitterwelt:



  - 8 Sensoren $s_1, \ldots, s_8$ liefern Zustand der Nachbarfelder
  - 4 Aktionen: go east, go north, go west, go south
  - direkte Berechnung der Aktion aus den Sensoreingaben, kein Gedächtnis
- **Aufgabe:** Umlaufe ein im Raum stehendes Hindernis oder laufe die Begrenzung des Raumes ab!





Stimulus-Response-Agent

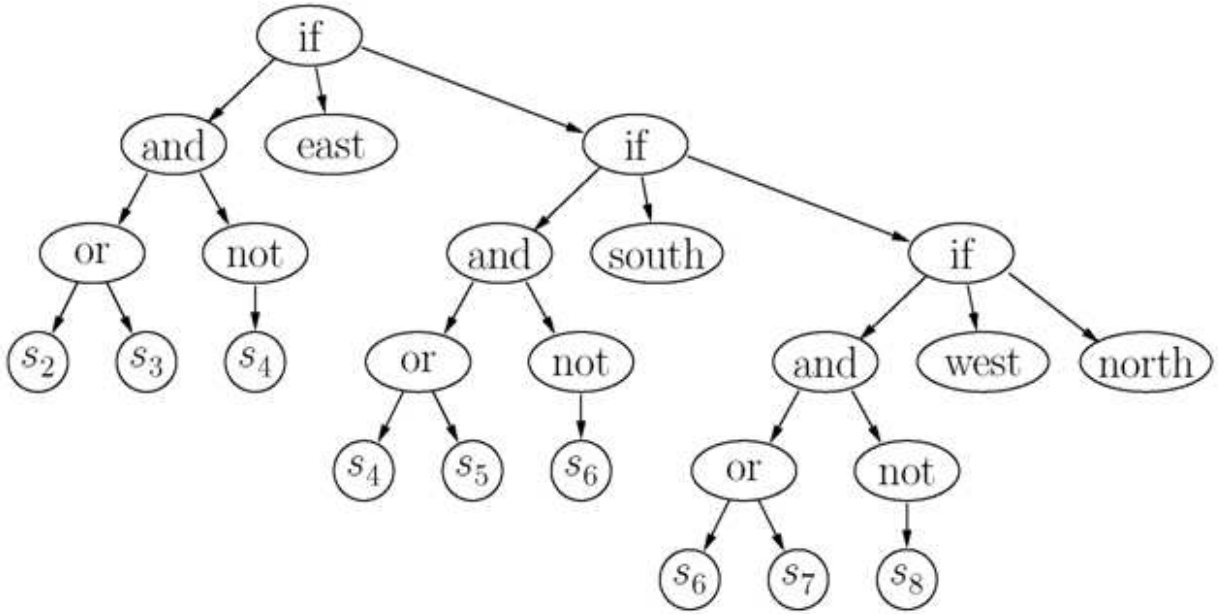
- Symbolmengen:
  - $\mathcal{T} = \{s_1, \ldots, s_8, \text{east}, \text{north}, \text{west}, \text{south}, 0, 1\}$
  - $\mathcal{F} = \{\text{and}, \text{or}, \text{not}, \text{if}\}$
- Vervollständigung der Funktionen, z.B. durch

$$(\text{and } x\ y) = \begin{cases} \text{false}, & \text{falls } x = \text{false}, \\ y, & \text{sonst.} \end{cases}$$

  (Beachte: So kann auch eine logische Operation eine Aktion liefern.)
- Populationsgröße popsize = 5000, Turnierauswahl mit Turniergröße 5
- Aufbau der Nachfolgepopulation
  - 10% (500) Lösungskandidaten werden unverändert übernommen.
  - 90% (4500) Lösungskandidaten werden durch Crossover erzeugt.
  - $<$1% der Lösungskandidaten werden mutiert.
- 10 Generationen (ohne Anfangspopulation) werden berechnet.

Stimulus-Response-Agent

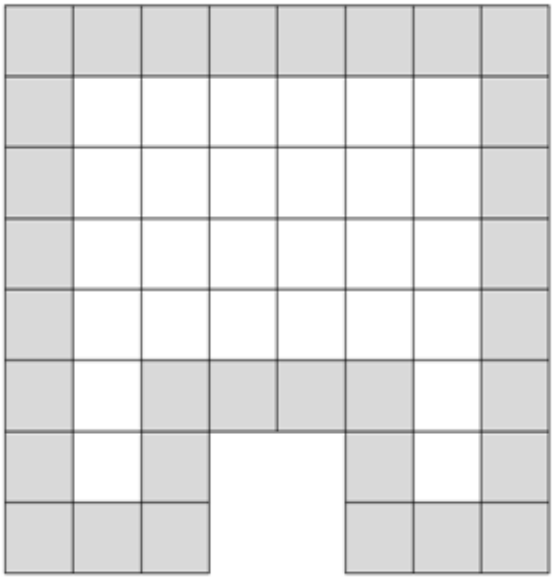
- **optimale, von Hand konstruierte Lösung:**



- Es ist höchst unwahrscheinlich, dass genau diese Lösung gefunden wird.
- Um die Chromosomen einfach zu halten, ist es u.U. sinnvoll, einen Strafterm zu berechnen, der die Komplexität des Ausdrucks misst.

Stimulus-Response-Agent

- Bewertung einer Kandidatenlösung anhand eines Testraumes:



  - Ein perfekt arbeitendes Steuerprogramm lässt den Agenten die grau gezeichneten Felder ablaufen.
  - Das Startfeld wird zufällig gewählt.
  - Ist eine Aktion nicht ausführbar oder wird statt einer Aktion ein Wahrheitswert geliefert, so wird die Ausführung des Steuerprogramms abgebrochen.

- Ein durch ein Chromosom gesteuerter Agent wird auf 10 zufällige Startfelder gesetzt und seine Bewegung verfolgt.
- Die Zahl der insgesamt besuchten Randfelder (grau unterlegt) ist die Fitness. (maximale Fitness: $10 \cdot 32 = 320$)

Wandverfolgung

Im Beispiel der Wandverfolgung kann gut mit der Selektionsmethode der Turnierauswahl gearbeitet werden.

- Aus der aktuellen Generation von SR-Steuerprogrammen wird eine festgelegte Anzahl zufällig ausgewählt.
- Diese ausgewählten Individuen treten gegeneinander an. Dazu wird bestimmt, wer die aktuelle Aufgabe besser erfüllt.
- Der/die Gewinner des Turniers werden in die nächste Generation übernommen.

Wandverfolgung

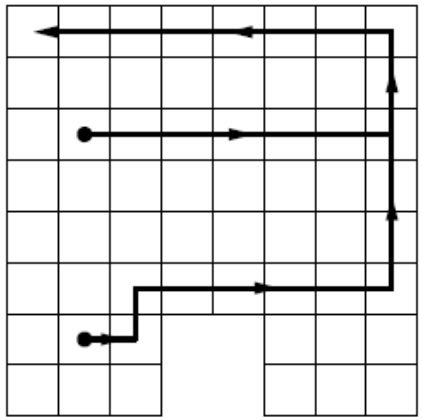
Die meisten der 5000 Programme in der Generation 0 sind nutzlos.

- `(and sw ne)`
  - wertet nur aus und terminiert dann
  - Fitness von 0
- `(or east west)`
  - liefert manchmal `west` und geht somit einen Schritt nach Westen
  - landet manchmal neber einer Wand
  - Fitness von 5
- Das beste Programm hat eine Fitness von 92. Es ist schwer zu lesen und hat redundante Operatoren. Der Weg mit zwei Startpunkten wird auf dem nachfolgendem Bild beschrieben.
  (Es läuft nach Osten bis zu einer Wand, dann nach Norden bis es nach Osten oder Westen kann und dann in der Ecke oben links gefangen ist.)

Bestes Individuum der Generation 0

```
(and (not (not (if  (if  (not s1)
                         (if  s4 north east)
                         (if  west 0 south))
                    (or  (if  s1 s3 s8) (not s7))
                    (not (not north)))))
     (if  (or  (not (and (if  s7 north s3)
                         (and south 1)))
               (or  (or  (not s6) (or s4 s4))
                    (and (if  west s3 s5)
                         (if  1 s4 s4))))
          (or  (not (and (not s3)
                         (if  east s6 s2)))
               (or  (not (if  s1 east s6))
                    (and (if  s8 s7 1)
                         (or  s7 s1))))
          (or  (not (if  (or  s2 s8)
                         (or  0 s5)
                         (or  1 east)))
               (or  (and (or  1 s3)
                         (and s1 east))
                    (if  (not west)
                         (and west east)
                         (if  1 north s8))))))
```

Bestes Individuum in Generation 0:

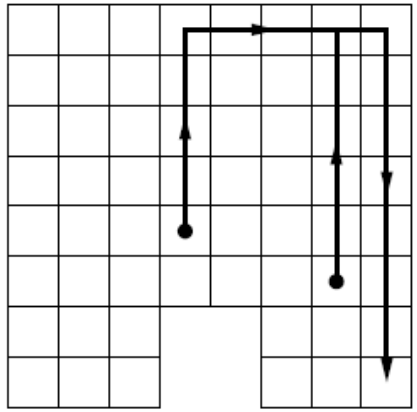


(Bewegung von zwei Startpunkten aus)

Beste Individuen der Generationen 2 und 6

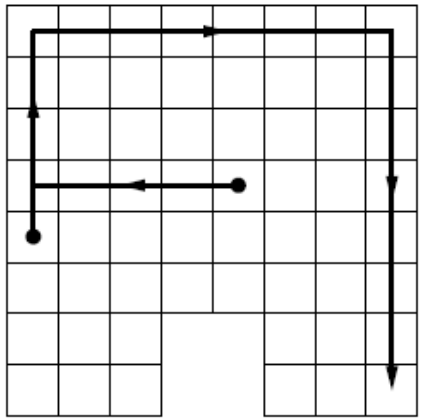
Bestes Individuum in Generation 2:

```
(not (and (if  s3
               (if  s5 south east)
               north)
          (and not s4)))
```



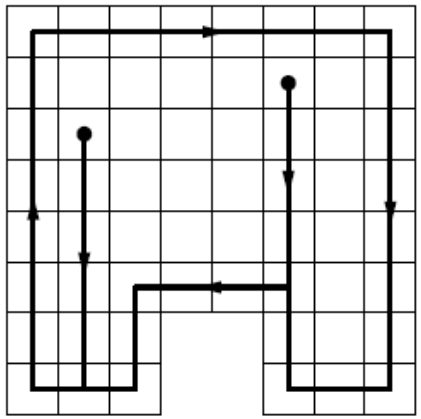
Bestes Individuum in Generation 6:

```
(if  (and (not s4)
          (if  s4 s6 s3))
     (or  (if  1  s4 south)
          (if  north east s3))
     (if  (or  (and 0 north)
               (and s4 (if  s4
                            (if  s5 south east)
                            north)))
          (and s4 (not (if  s6 s7 s4)))
          (or  (or  (and s1 east) west) s1)))
```

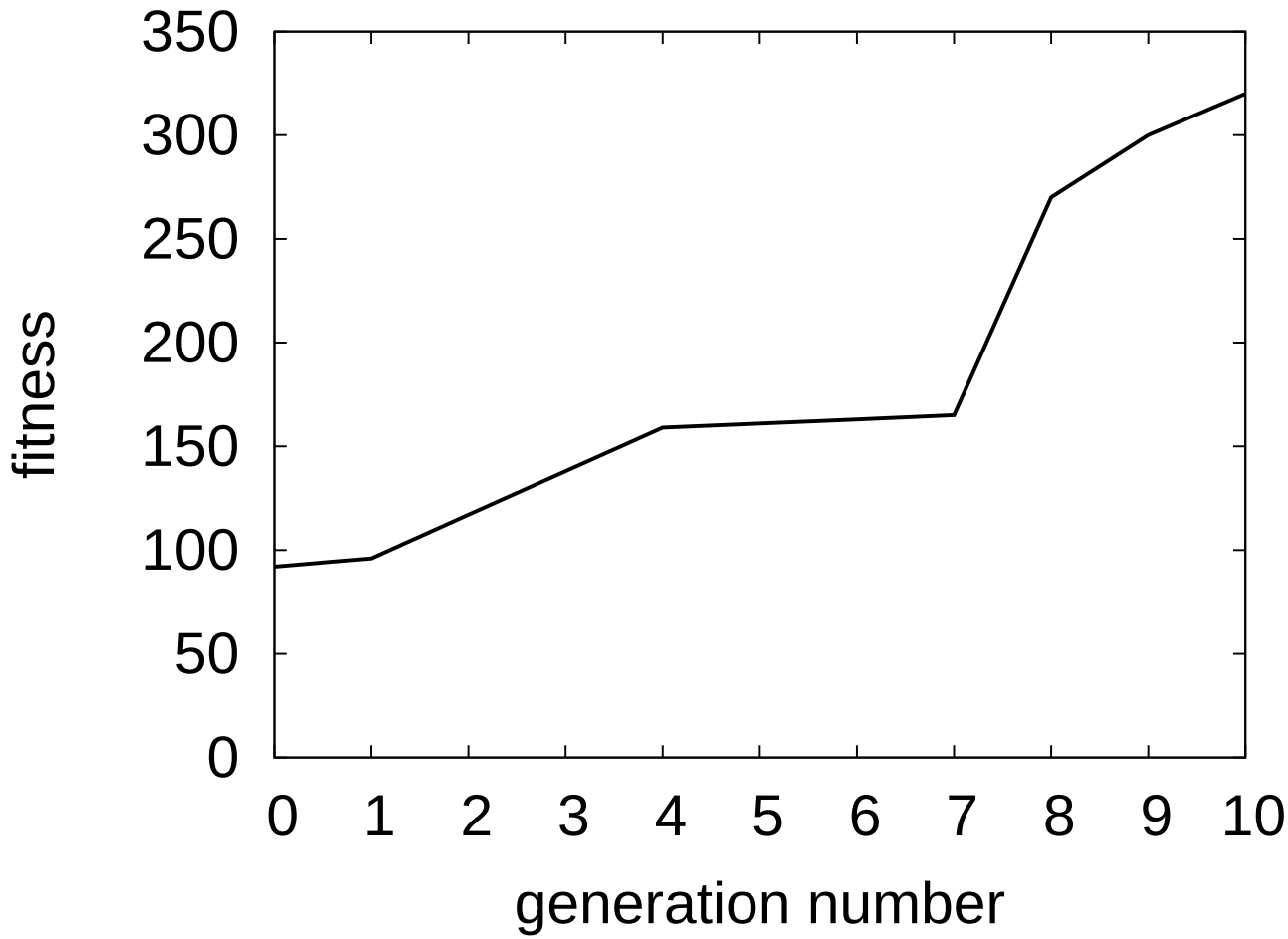


Bestes Individuum der Generation 10

```
(if  (if  (if  s5 0 s3)
          (or  s5 east)
          (if  (or  (and s4 0)
                    s7)
               (or  s7 0)
               (and (not (not (and s6 s5)))
                    s5)))
     (if  s8
          (or north
              (not (not s6)))
          west)
     (not (not (not (and (if  (not south)
                              s5
                              s8)
                         (not s2))))))
```



Entwicklung der Fitness



Entwicklung der Fitness im Laufe des Lernvorgangs
(bestes Individuum der jeweiligen Generation)

Details...

Im Sommersemester findet unsere Veranstaltung

Evolutionäre Algorithmen

statt, in der die vorangehenden Themen detailliert und vertiefend behandelt werden.

Literatur I

Gerdes, I., Klawonn, F., and Kruse, R. (2004).

Evolutionäre Algorithmen: genetische Algorithmen - Strategien und Optimierungsverfahren - Beispielanwendungen.
Vieweg, Wiesbaden, Germany.

Koza, J. R. (1992).

Genetic Programming: On the Programming of Computers by Means of Natural Selection.
MIT Press, Boston, MA, USA.

Nilsson, N. J. (1998).

Artificial Intelligence: A New Synthesis.
Morgan Kaufmann Publishers, Inc., San Francisco, CA, USA.

Weicker, K. (2007).

Evolutionäre Algorithmen.
Teubner, Stuttgart, Germany, 2 edition.