

Evolutionäre Algorithmen

Genetische Programmierung

Prof. Dr. Rudolf Kruse **Christian Moewes**

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

Otto-von-Guericke-Universität Magdeburg

Fakultät für Informatik

Institut für Wissens- und Sprachverarbeitung

Übersicht

1. Motivation

- Genetische Programmierung
- Terminal- und Funktionssymbole
- Symbolische Ausdrücke
- Ablauf eines GPs

2. Initialisierung

3. Genetische Operatoren

4. Beispiele

5. Zusammenfassung und Ausblick

Genetische Programmierung

- Genetische Programmierung (GP) basiert auf folgender Idee:
- Beschreibung einer Problemlösung durch Computerprogramm, das gewisse Eingaben mit gewissen Ausgaben verbindet

⇒ Suche nach passendem Computerprogramm

- GP = genereller Weg, Computerprogramme zu lernen/zu erzeugen
- Darstellung der Programme durch sog. Parse-Bäume

Lernen von Programmen

Viele verschiedene Probleme können als Lernprobleme eines Programms interpretiert werden:

- Regelung
- Planen
- Suchen
- Wissensrepräsentation
- symbolische Regression
- Induktion von Entscheidungsbäumen
- ...

Genetische Programmierung

Darstellung der Lösungskandidaten:

- **bisher:** durch Chromosomen fester Länge (Vektor von Genen)
- **jetzt:** durch Funktionsausdrücke bzw. Programme
 - ⇒ **genetische Programmierung**
 - ⇒ 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)
- Mengen \mathcal{F} und \mathcal{T} sind problemspezifisch
- sollten nicht zu groß sein (Beschränkung des Suchraums) und doch reichhaltig genug, um Problemlösung zu ermöglichen

Beispiele zu Symbolmengen

- **Beispiel 1:** Erlernen einer Booleschen Funktion

- $\mathcal{F} = \{\text{and, or, not, if } \dots \text{ then } \dots \text{ else } \dots, \dots\}$
- $\mathcal{T} = \{x_1, \dots, x_m, 1, 0\}$ bzw. $\mathcal{T} = \{x_1, \dots, x_m, t, f\}$

- **Beispiel 2:** Symbolische Regression

- Regression: Bestimmung einer Ausgleichsfunktion zu geg. Daten unter Minimierung der Fehlerquadratsumme – *Methode der kleinsten Quadrate*
- $\mathcal{F} = \{+, -, *, /, \sqrt{}, \sin, \cos, \log, \exp, \dots\}$
- $\mathcal{T} = \{x_1, \dots, x_m\} \cup \mathbb{R}$

Abgeschlossenheit von \mathcal{F} und \mathcal{T}

wünschenswert: \mathcal{F} und \mathcal{T} sollten abgeschlossen sein

- Funktionen $\in \mathcal{F}$ akzeptieren alle möglichen Eingabewerte
 - z.B. Division durch Null würde Ausführung mit Fehler beenden
- falls Abgeschlossenheit nicht gilt, dann Lösen eines Optimierungsproblems mit Bedingungen

⇒ Reparaturmaßnahmen oder Strafterme u.U. nötig

Abgeschlossenheit von \mathcal{F} und \mathcal{T}

verschiedene Strategien garantieren Abgeschlossenheit, z.B.

- Implementierung von gesicherten Versionen von anfälligen Operatoren, z.B.
 - gesicherte Division, die Null oder Maximalwert zurückgibt
 - gesicherte Wurzelfunktion, die mit Absolutwert operiert
 - gesicherte Logarithmusfunktion: $\forall x \leq 0 : \log(x) = 0$ o.Ä.
- Kombination verschiedener Funktionsarten
 - z.B. numerische und boolesche Werte ($\text{FALSE} = 0, \text{TRUE} \neq 0$)
- Implementierung von bedingten Vergleichsoperatoren
 - z.B. *IF* $x < 0$ *THEN* ...
- ...

Vollständigkeit von \mathcal{F} und \mathcal{T}

Ein GP kann nur effizient und effektiv ein Problem lösen, wenn Funktions- und Terminalmenge hinreichend/vollständig sind, um ein angemessenes Programm zu finden.

- z.B. in Boolescher Aussagenlogik sind $\mathcal{F} = \{\wedge, \neg\}$ und $\mathcal{T} = \{\rightarrow, \neg\}$ vollständige Operatorenmengen, $\mathcal{F} = \{\wedge\}$ nicht
 - generelles Problem des maschinellen Lernens: **Merkmalsauswahl**
 - Finden der kleinsten vollständigen Menge ist (meistens) NP-schwer
- ⇒ gewöhnlich: mehr Funktionen in \mathcal{F} als eigentlich notwendig

Symbolische Ausdrücke

- Chromosomen = Ausdrücke (zusammengesetzt aus Elementen aus $\mathcal{C} = \mathcal{F} \cup \mathcal{T}$ und ggf. Klammern)
 - allerdings: Beschränkung auf „wohlgeformte“ Ausdrücke
- ⇒ üblich: **rekursive Definition** (Präfixnotation):
- Konstanten- und Variablensymbole sind symbolische Ausdrücke
 - sind t_1, \dots, t_n symbolische Ausdrücke und ist $f \in \mathcal{F}$ ein (n -stelliges) Funktionssymbol, so ist $(ft_1 \dots t_n)$ symbolischer Ausdruck
 - keine anderen Zeichenfolgen sind symbolische Ausdrücke
 - **Beispiele** zu dieser Definition:
 - „ $(+ (* 3 x) (/ 8 2))$ “ ist symbolischer Ausdruck
Lisp- bzw. Scheme-artige Schreibweise, Bedeutung: $3 \cdot x + \frac{8}{2}$
 - „ $2 7 * (3 /$ “ ist kein symbolischer Ausdruck

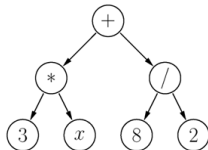
Implementierung

- günstig für Implementierung der GPs: Darstellung symbolischer Ausdrücke durch sog. Parse-Bäume
(Parse-Bäume werden im 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 Listenelement ist Funktionssymbol bzw. Operator
nachfolgende Elemente sind Argumente bzw. Operanden

Ablauf einer Genetischen Programmierung

- Erzeugen einer **Anfangspopulation** zufälliger symbolischer Ausdrücke
- **Bewertung** der Ausdrücke durch Berechnung der Fitness
 - Erlernen Boolescher Funktionen: Anteil korrekter Ausgaben für alle Eingaben bzgl. einer Stichprobe
 - Symbolische Regression: Summe der Fehlerquadrate über gegebene Messpunkte
1-D: Daten (x_i, y_i) , $i = 1, \dots, n$, 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

Übersicht

1. Motivation

2. Initialisierung

„Wachsen“

„Voll“

„Aufsteigend halb-und-halb“

3. Genetische Operatoren

4. Beispiele

5. Zusammenfassung und Ausblick

Initialisierung einer GP-Population

Parameter des Erzeugungsprozesses:

- maximale Verschachtelungstiefe (maximale Baumhöhe) d_{\max} oder
- maximale Anzahl an Knoten im Baum n_{\max}

drei verschiedene Methoden zur Initialisierung [Koza, 1992]:

1. *wachsen* (engl. grow)
2. *voll* (engl. full)
3. *aufsteigend halb-und-halb* (engl. ramped half-and-half)
 - wie in EAs können auch hier Kopien vermieden werden
 - Aufruf von *wachsen* und *voll*: Initialisiere(Wurzel, 0)

„Wachsen“

Algorithm 1 Initialisiere-Wachsen

Input: Knoten n , Tiefe d , Maximaltiefe d_{\max}

```
1: if  $d = 0$  {
2:    $n \leftarrow$  ziehe Knoten aus  $\mathcal{F}$  mit gleichverteilter W'keit
3: } else { if  $d = d_{\max}$  {
4:    $n \leftarrow$  ziehe Knoten aus  $\mathcal{T}$  mit gleichverteilter W'keit
5: } else {
6:    $n \leftarrow$  ziehe Knoten aus  $\mathcal{F} \cup \mathcal{T}$  mit gleichverteilter W'keit
7: }
8: if  $n \in \mathcal{F}$  {
9:   for each  $c \in$  Argumente von  $n$  {
10:     Initialisiere-Wachsen( $c, d + 1, d_{\max}$ )
11:   }
12: } else {
13:   return
14: }
```

- erzeugt Bäume von irregulärer Form
- Knoten: zufällige Auswahl aus \mathcal{F} und \mathcal{T} (bis auf Wurzel)
- Zweig mit Terminalsymbol endet auch bevor max. Tiefe erreicht

„Voll“

Algorithm 2 Initialisiere-Voll

Input: Knoten n , Tiefe d , Maximaltiefe d_{\max}

```
1: if  $d \leq d_{\max}$  {  
2:    $n \leftarrow$  ziehe Knoten aus  $\mathcal{F}$  mit gleichverteilter W'keit  
3:   for each  $c \in$  Argumente von  $n$  {  
4:     Initialisiere-Voll( $c, d + 1, d_{\max}$ )  
5:   }  
6: } else {  
7:    $n \leftarrow$  ziehe Knoten aus  $\mathcal{T}$  mit gleichverteilter W'keit  
8: }  
9: return
```

- erzeugt ausbalancierte Bäume
- Knoten: zufällige Auswahl *nur* aus \mathcal{F} (bis zu max. Tiefe)
- bei max. Tiefe: zufällige Auswahl *nur* aus \mathcal{T}

„Aufsteigend halb-und-halb“

Algorithm 3 Initialisiere-Aufsteigend-halb-und-halb

Input: Maximaltiefe d_{\max} , Pop.-größe μ (gerades Vielfaches von d_{\max})

```
1:  $P \leftarrow \emptyset$ 
2: for  $i \leftarrow 1 \dots d_{\max}$  {
3:   for  $j \leftarrow 1 \dots \mu / (2 \cdot d_{\max})$  {
4:      $P \leftarrow P \cup \text{Initialisiere-Voll}(\text{Wurzel}, 0, i)$ 
5:      $P \leftarrow P \cup \text{Initialisiere-Wachsen}(\text{Wurzel}, 0, i)$ 
6:   }
7: }
```

- kombiniert Methoden *wachsen* und *voll*
 - generiert gleiche Anzahl an gewachsenen und vollen Bäumen mit allen möglichen Tiefen zwischen 1 und d_{\max}
- ⇒ große Variation an Baumgrößen und -formen
- besser für GP (siehe evolutionäre Prinzipien)

Übersicht

1. Motivation

2. Initialisierung

3. Genetische Operatoren

Crossover

Mutation

4. Beispiele

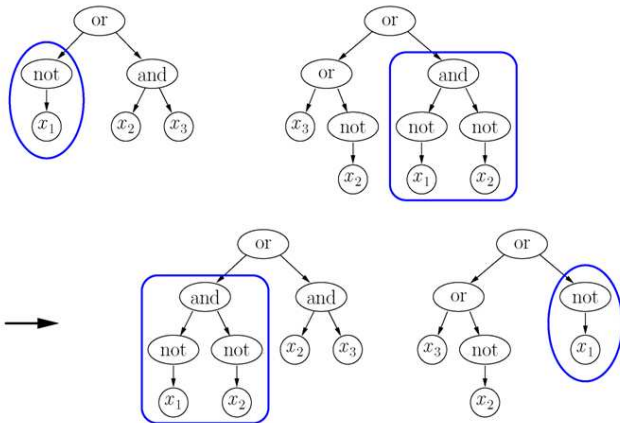
5. Zusammenfassung und Ausblick

Genetische Operatoren

- für gewöhnlich: initiierte Population hat sehr geringe Fitness
- evolutionärer Prozess verändert anfängliche Population durch genetische Operatoren
- für GPs: viele verschiedene genetische Operatoren
- wichtigsten drei:
 - Crossover
 - Mutation
 - klonale Reproduktion (Kopieren eines Individuums)

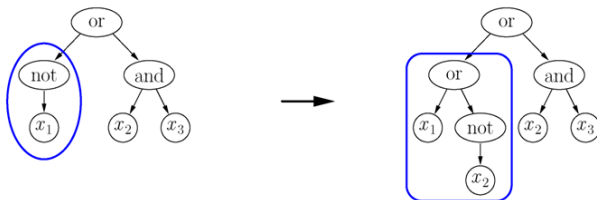
Crossover

- Austausch zweier Teilausdrücke (Teilbäume)



Mutation

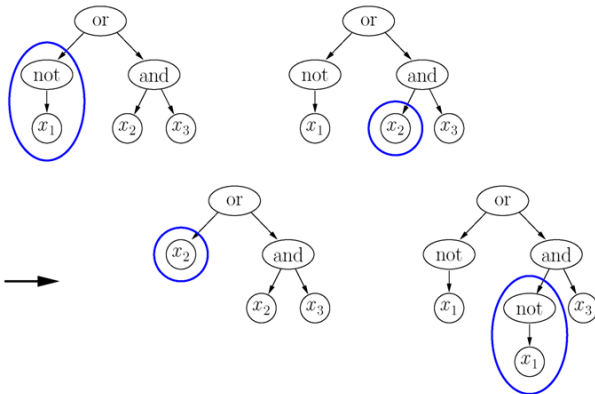
- Ersetzen eines Teilausdrucks (Teilbaums) durch zufällig erzeugten:



- möglichst nur kleine Teilbäume ersetzen
- bei großer Population: meist nur Crossover und keine Mutation, da hinreichender Vorrat an „genetischem Material“

Vorteil des Crossover

- Crossover von GPs ist mächtiger als Crossover von Vektoren
- Grund: Crossover identischer Elternprogramme führt u.U. zu verschiedenen Individuen



Übersicht

1. Motivation

2. Initialisierung

3. Genetische Operatoren

4. Beispiele

11-Multiplexer

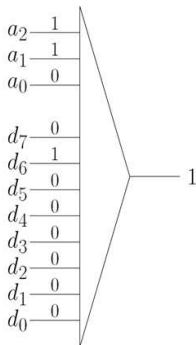
Stimulus-Response-Agent

Symbolische Regression mit Konstanten

5. Zusammenfassung und Ausblick

Beispiel: 11-Multiplexer

Erlernen eines Booleschen 11-Multiplexers [Koza, 1992]



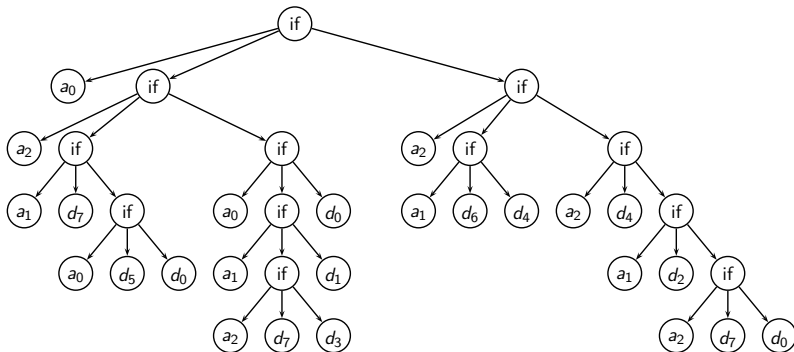
- Multiplexer mit 8 Daten- und 3 Adressleitungen (Zustand der Adressleitungen gibt durchzuschaltende Datenleitung an)
- $2^{11} = 2048$ mögliche Eingaben mit je einer zugehörigen Ausgabe
- Festlegung der Symbolmengen:
 - $\mathcal{T} = \{a_0, a_1, a_2, d_0, \dots, d_7\}$
 - $\mathcal{F} = \{\text{and,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

11-Multiplexer

- Populationsgröße $|P| = 4000$
- Anfangstiefe der Parse-Bäume: 6, maximale Tiefe: 17
- Fitnesswerte in Anfangspopulation zwischen 768 und 1280, mittlere Fitness von 1063
(Erwartungswert ist 1024, da bei zufälliger Ausgabe im Durchschnitt Hälfte der Ausgaben richtig)
- 23 Ausdrücke haben Fitness von 1280, einer davon entspricht 3-Multiplexer: (if a_0 d_1 d_2)
- fitnessproportionale Selektion
- 90% (3600) der Individuen werden Crossover unterworfen
- 10% (400) werden unverändert übernommen

11-Multiplexer

- nach nur 9 Generationen: Lösung mit Fitness 2048



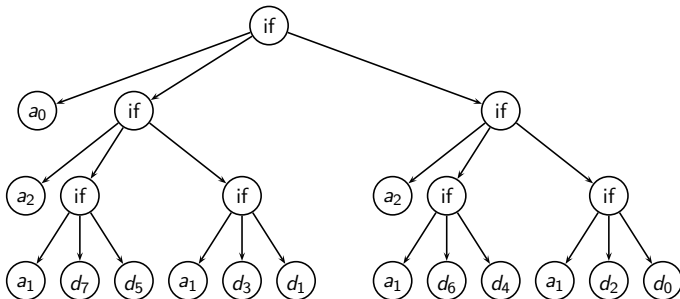
- eher schwer zu interpretieren für Menschen
- kann vereinfacht werden durch Umformung (engl. editing)

Umformung

- asexuelle Operation eines Individuums
 - dient der Vereinfachung durch generelle und spezielle Regeln
 - **generell:** falls Funktion ohne Nebeneffekte im Baum mit konstanten Argumenten auftaucht, dann evaluiere Funktion und ersetze Teilbaum mit Ergebnis
 - **speziell:** hier Aussagenlogik
 - $\neg(\neg A) \rightarrow A$, $(A \wedge A) \rightarrow A$, $(A \vee A) \rightarrow A$, usw.
 - de Morgan'schen Gesetze, usw.
 - Umformung: z.B. als Operator während GP-Suche
- ⇒ Reduktion aufgeblähter Individuen auf Kosten der Diversität
- normalerweise: Umformung nur zur Interpretation der Ergebnisse

11-Multiplexer

- beste Lösung gestutzt durch Aufbereitung:



11-Multiplexer

- **beachte:** beste Lösung durch GP ist hierarchisch
- Zerlegung des 11-Multiplexer-Problems anhand zwei kleinerer:
 - 6-Multiplexer: 2 Adressbits und 4 Datenbits
 - 3-Multiplexer: 1 Adressbit und 2 Datenbits
- GP-Lösung = Komposition zweier 6-Multiplexer:
 a_0 wird genutzt, zu entscheiden ob entweder Adressbit a_1 oder a_2 auf d_7, d_5, d_3, d_1 oder d_6, d_4, d_2, d_0 verweisen sollten
- weiterhin: unterste Ebene der 6-Multiplexer ist Komposition zweier 3-Multiplexer


11-Multiplexer

- bestes Individuum in 9. Generation erreicht bestmögliche Fitness
- Frage: wie wahrscheinlich ist dies anhand blinder Suche?
- Schätzung der Zahl aller booleschen Funktionen:
 - Wie viele boolesche Funktionen gibt es für 11 Variablen?
 - Warum ist dieser Wert nicht hinreichend für GPs?
 - Wie viele Möglichkeiten gibt es mit unbeschränkter Baumtiefe?

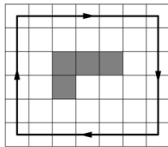
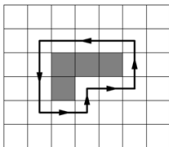
Stimulus-Response-Agent

Erlernen eines Robotersteuerprogramms [Nilsson, 1998]

- betrachte Stimulus-Response-Agenten in Gitterwelt:

s_1	s_2	s_3
s_8		s_4
s_7	s_6	s_5

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



Stimulus-Response-Agent

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

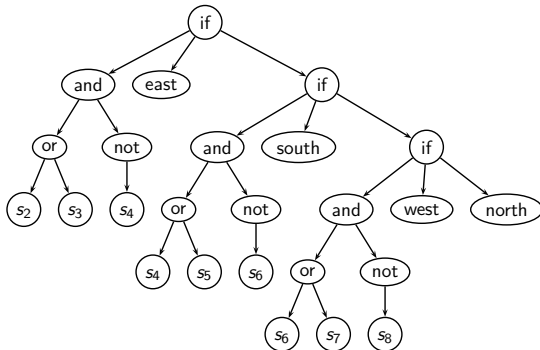
$$(\text{and } x \ y) = \begin{cases} \text{false,} & \text{falls } x = \text{false,} \\ y, & \text{sonst.} \end{cases}$$

(beachte: so kann auch logische Operation Aktion liefern)

- Populationsgröße $|P| = 5000$, 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

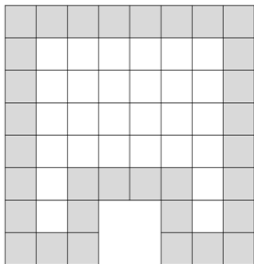
- optimale, von Hand konstruierte Lösung:



- es ist höchst unwahrscheinlich, genau diese Lösung zu finden
- um Chromosomen einfach zu halten, ist es u.U. sinnvoll, Strafterm zu berechnen, der Komplexität des Ausdrucks misst

Stimulus-Response-Agent

- Bewertung einer Kandidatenlösung anhand eines Testraumes:



- perfekt arbeitendes Steuerprogramm lässt Agenten grau gezeichneten Felder ablaufen
- Startfeld wird zufällig gewählt
- ist Aktion nicht ausführbar oder wird statt Aktion Wahrheitswert geliefert, so wird Ausführung des Steuerprogramms abgebrochen
- durch Chromosom gesteuerter Agent wird auf 10 zufällige Startfelder gesetzt und seine Bewegung verfolgt
- Zahl der insgesamt besuchten Randfelder (grau unterlegt) ist Fitness (maximale Fitness: $10 \cdot 32 = 320$)

Wandverfolgung

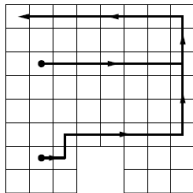
meisten der 5000 Programme in Generation 0 sind nutzlos

- (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 neben einer Wand
 - Fitness von 5
- bestes Programm hat Fitness von 92
 - schwer zu lesen, hat redundante Operatoren
 - Weg mit zwei Startpunkten auf nächster Folie beschrieben (Osten bis zu Wand, dann Norden bis nach Osten oder Westen möglich und dann in Ecke oben links gefangen)

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:

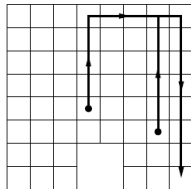


(Bewegung von zwei
Startpunkten aus)

Beste Individuen der Generationen 2 und 6

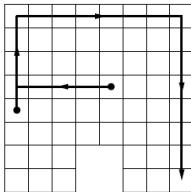
Bestes Individuum in Generation 2:

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



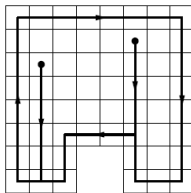
Bestes Individuum in Generation 6:

```
(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)))
```

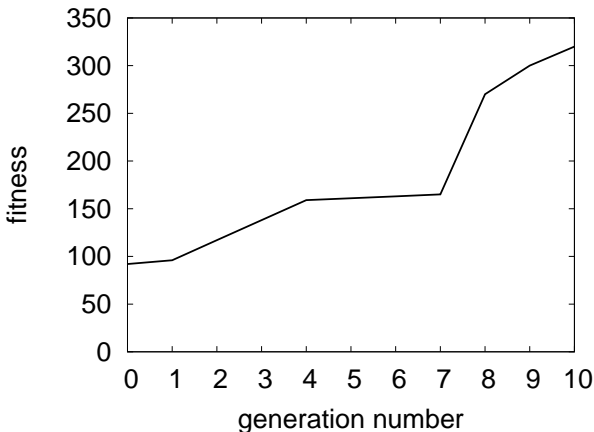


Bestes Individuum der Generation 10

```
(if (if (if (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)

Symbolische Regression mit Konstanten

- bisher: symbolische Regression boolescher Funktionen
 - sind am einfachsten (nur zwei Werte: „wahr“ oder „falsch“)
- nun: symbolische Regression komplizierterer Funktionen
 - wünschenswert: reellwertiger Definitions- und Wertebereich
 - z.B. Funktion der Fläche eines Kreises $a = \pi r^2$
 - Konstante $\pi \in \mathcal{T}$, da geometrisches Problem
 - weitere Elemente aus \mathcal{T} nicht so offensichtlich. . .

Symbolische Regression mit Konstanten

- unpraktisch: alle realen Zahlen in \mathcal{T}
- selbst für \mathbb{N} „explodiert“ Größe von \mathcal{T}
- **Lösung:** zufällig flüchtige/kurzlebige Konstanten (engl. ephemeral random constants) [Koza, 1992]
 - zusätzliches Terminal bezeichnet durch \mathbb{R}
 - bei Initialisierung für jedes Individuum: $\mathbb{R} \leftarrow$ zufälliger Wert aus sinnvollem Intervall
 - Crossover „bewegt“ \mathbb{R} 's zwischen Bäumen
 - arithmetische Operatoren ergeben neue Werte/Elemente in \mathcal{T}

Übersicht

1. Motivation

2. Initialisierung

3. Genetische Operatoren

4. Beispiele

5. Zusammenfassung und Ausblick

Problem der Introns

Erweiterungen

Problem der Introns

- mit zunehmender Generationenzahl: Individuen wachsen
 - Grund: sogenannten **Introns**:
 - Biologie: Teile der DNA ohne Information
 - inaktive (evtl. veraltete) Abschnitte außerhalb eines Gens oder funktionslose Abschnitte außerhalb der Gene (engl. *junk DNA*)
 - z.B. arithm. Ausdruck $a + (1 - 1)$ leicht zu vereinfachen
 - in `if 2 < 1 then ...else ...` ist then-Zweig sinnlos
 - Veränderungen durch Operatoren in aktiven Teilen des Individuums haben meist negative Wirkung auf Güte
 - Änderungen an Introns sind güteneutral
- ⇒ führt zu künstlichem Aufblähen der Individuen
- ⇒ aktiver Programmcode nimmt relativ ab – Optimierung stagniert

Verhinderung von Introns

unterschiedliche Techniken:

- Nutzen von modifizierten Operatoren:
 - bei **Brutrekombination** werden aus zwei Eltern durch unterschiedliche Parametrisierung sehr viele Kinder erzeugt, wovon nur Bestes in nächste Generation kommt
 - **Intelligente Rekombination** wählt gezielte Crossover-Punkte
 - durch **fortwährend leichte Veränderungen der Bewertungsfunktion** können Randbedingungen so verändert werden, dass inaktive Programmteile (Introns) wieder aktiv werden
⇒ funktioniert allerdings nur bei nicht-trivialen Introns, die durch immer ähnliche Eingabedaten definiert werden
- Bestrafung großer Individuen ⇒ Benachteiligung während Selektion

Erweiterungen

Kapselung (Encapsulation) automatisch definierter Funktionen


- potentiell gute Teilausdrücke sollten vor Zerstörung durch Crossover und Mutation geschützt werden
- für Teilausdruck (eines guten Chromosoms) wird neue Funktion definiert und das sie bezeichnende Symbol ggf. der Menge \mathcal{F} hinzugefügt
- Zahl der Argumente der neuen Funktion ist gleich der Zahl der (verschiedenen) Blätter des Teilbaums

Iterationen

Rekursion


...

Literatur zur Lehrveranstaltung I

 Banzhaf, W., Nordin, P., Keller, R. E., and Francone, F. D. (1998).


Genetic Programming — An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.

Morgan Kaufmann Publisher, Inc. and dpunkt-Verlag, San Francisco, CA, USA and Heidelberg, 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.