Evolutionäre Algorithmen

Genetische Programmierung

Prof. Dr. Rudolf Kruse **Pascal Held**
{kruse,pheld}@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 den folgenden Ideen:

- Beschreibung einer Problemlösung durch Computerprogramm, das gewisse Eingaben mit gewissen Ausgaben verbindet
- Suche nach passendem Computerprogramm
- Genereller Weg, Computerprogramme zu lernen/zu erzeugen
- Darstellung der Programme durch Parse-Bäume

Lernen von Programmen

Viele Probleme können als „Erlerneneines Programms“ interpretiert werden, z.B.:

- 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, also
  - komplexe 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 die 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 geg. Daten unter Minimierung der Fehlerquadratsumme – *Methode der kleinsten Quadrate*
  - $\mathcal{F} = \{+, -, *, /, \sqrt{\ }, \sin, \cos, \log, \exp, \ldots\}$
  - $\mathcal{T} = \{x_1, \ldots, x_m\} \cup \mathbb{R}$

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

$\mathcal{F}$ und $\mathcal{T}$ sollten abgeschlossen sein, damit es nicht zu Programmabbrüchen kommt, wenn fehlerhafte Parameter oder Parametertypen an Funktionssymbole übergeben werden.
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: $\forall x \leq 0: \quad \log(x) = 0$ o.Ä.
- Kombination verschiedener Funktionsarten
    - z.B. numerische und boolesche Werte ($\text{FALSE} = 0, \text{TRUE} \neq 0$)
- Implementierung von bedingten Vergleichsoperatoren
    - z.B. $\mathit{IF}\ x < 0\ \mathit{THEN} \ldots$
- ...

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.

In Boolescher Aussagenlogik sind $\mathcal{F} = \{\wedge, \neg\}$ und $\mathcal{F} = \{\rightarrow, \neg\}$ vollständige Operatorenmengen, $\mathcal{F} = \{\wedge\}$ nicht

- Generelles Problem des maschinellen Lernens: **Merkmalsauswahl**
- Das Finden der kleinsten vollständigen Menge ist (meistens) NP-schwer
- Oft sind 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)

Beschränkung auf „wohlgeformte" Ausdrücke

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 $(ft_1 \ldots t_n)$ symbolischer Ausdruck
- keine anderen Zeichenfolgen sind symbolische Ausdrücke

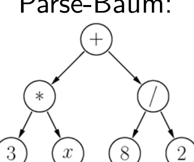
Beispiele:

- „$(+\ (*\ 3\ x)\ (/\ 8\ 2))$" ist symbolischer Ausdruck
  Lisp- bzw. Scheme-artige Schreibweise, Bedeutung: $3 \cdot x + \frac{8}{2}$
- „$2\ 7\ *\ (3\ /$" ist kein symbolischer Ausdruck

Implementierung

- 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)

Parse-Baum:



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

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), $i = 1, \ldots, n$, 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

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 ← ziehe Knoten aus 𝓕 mit gleichverteilter W'keit
 3: } else { if d = d_max {
 4:     n ← ziehe Knoten aus 𝓣 mit gleichverteilter W'keit
 5: } else {
 6:     n ← ziehe Knoten aus 𝓕 ∪ 𝓣 mit gleichverteilter W'keit
 7: }
 8: if n ∈ 𝓕 {
 9:     for each c ∈ 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** $d \leq d_{\text{max}}$ {
2: $n \leftarrow$ ziehe Knoten aus $\mathcal{F}$ mit gleichverteilter W'keit
3: **for each** $c \in$ Argumente von n {
4: Initialisiere-Voll($c, d+1, d_{\text{max}}$)
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** $i \leftarrow 1 \ldots d_{max}$ {
3:     **for** $j \leftarrow 1 \ldots \mu/(2 \cdot d_{max})$ {
4:         $P \leftarrow P \cup$ Initialisiere-Voll(Wurzel, 0, i)
5:         $P \leftarrow P \cup$ Initialisiere-Wachsen(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
- besonders für GP (siehe evolutionäre Prinzipien) geeignet

Übersicht

1. Motivation

2. Initialisierung

3. Genetische Operatoren
- Crossover
- Mutation

4. Beispiele

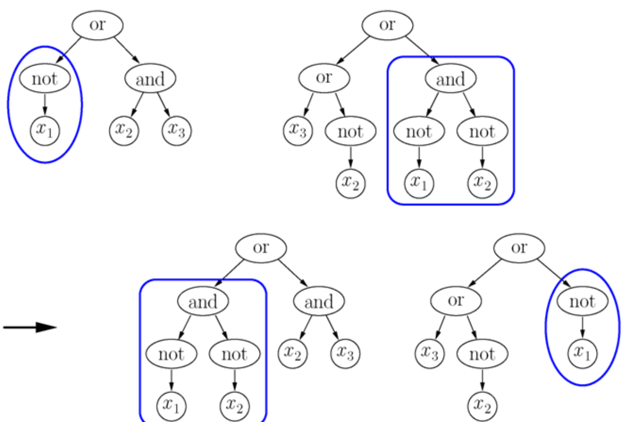
5. Zusammenfassung und Ausblick

Genetische Operatoren

- Initiierte Population hat üblicherweise sehr geringe Fitness
- Der evolutionäre Prozess verändert anfängliche Population durch genetische Operatoren
- für GPs: viele verschiedene genetische Operatoren
- Die wichtigsten drei genetischen Operatoren sind:
  Crossover,
  Mutation und die
  klonale Reproduktion (Kopieren eines Individuums).

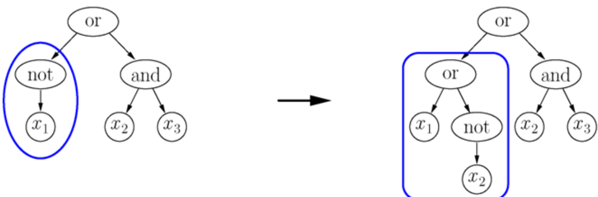
Crossover

- Austausch zweier Teilausdrücke (Teilbäume)



Mutation

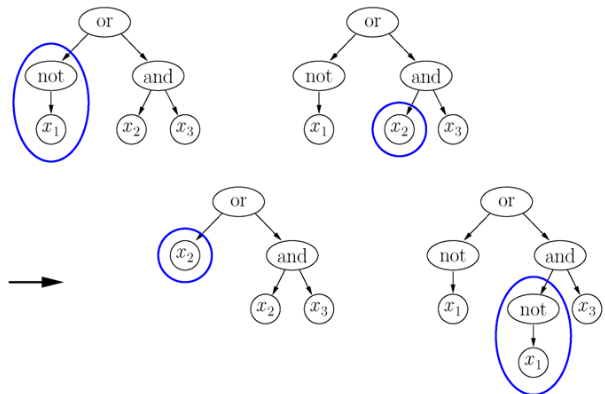
Ersetzen eines Teilausdrucks (Teilbaums) durch zufällig erzeugten:



- Möglichst nur kleine Teilbäume ersetzen
- Bei großer Population: meist nur Crossover und keine Mutation, da hinreichender Vorrat an „genetischem Material"

Vorteil des Crossover

Das Crossover bei GP ist mächtiger als das Crossover von Vektoren: Crossover identischer Elternprogramme führt u.U. zu verschiedenen Individuen



Übersicht

1. Motivation

2. Initialisierung

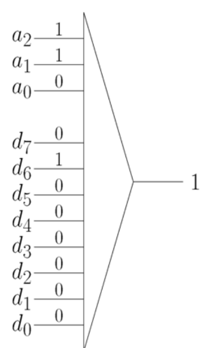
3. Genetische Operatoren

4. Beispiele
   11-Multiplexer
   Stimulus-Response-Agent

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:
  - $\mathcal{T} = \{a_0, a_1, a_2, d_0, \ldots, d_7\}$
  - $\mathcal{F} = \{\text{and}, \text{or}, \text{not}, \text{if}\}$
- Fitnessfunktion: $f(s) = 2048 - \sum_{i=1}^{2048} e_i$, wobei e_i Fehler für i-te Eingabe ist

Beispiel: 11-Multiplexer

Typische Werte

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

Beispiel: 11-Multiplexer

- Nach 9 Generationen: Lösung mit Fitness 2048

if
a_0 if if
a_2 if if a_2 if if
a_1 d_7 if a_0 if d_0 a_1 d_6 d_4 a_2 d_4 if
a_0 d_5 d_0 a_1 if d_1 a_1 d_2 if
a_2 d_7 d_3 a_2 d_7 d_0

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

$\neg(\neg A) \to A, \quad (A \wedge A) \to A, \quad (A \vee A) \to A$

de Morgan'schen Gesetze, usw.

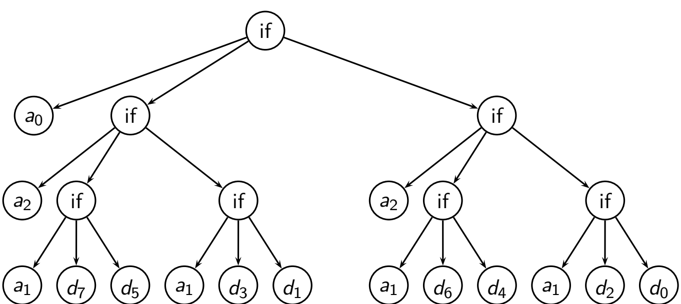
- Umformung: z.B. als Operator während GP-Suche

  Reduktion aufgeblähter Individuen auf Kosten der Diversität

  normalerweise: Umformung nur zur Interpretation der Ergebnisse

11-Multiplexer

Beste Lösung gestutzt durch Aufbereitung:



Beispiel: 11-Multiplexer

bestes Individuum in 9. Generation erreicht bestmögliche Fitness

Frage: wie wahrscheinlich ist dies anhand blinder Suche?

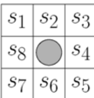
Schätzung der Zahl aller booleschen Funktionen:

- 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

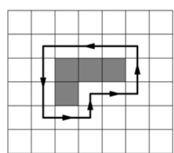
Erlernen eines Robotersteuerprogramms [Nilsson, 1998]

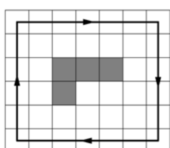
Betrachte Stimulus-Response-Agenten in Gitterwelt:



- 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 $s_1, \ldots, s_8$, kein Gedächtnis

Aufgabe: umlaufe ein im Raum stehendes Hindernis oder laufe Begrenzung des Raumes ab!





Stimulus-Response-Agent

Symbolmengen:

- $\mathcal{T} = \{s_1, \ldots, s_8, \text{east}, \text{north}, \text{west}, \text{south}, 0, 1\}$
- $\mathcal{F} = \{\text{and}, \text{or}, \text{not}, \text{if}\}$

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 logische Operation Aktion liefern)

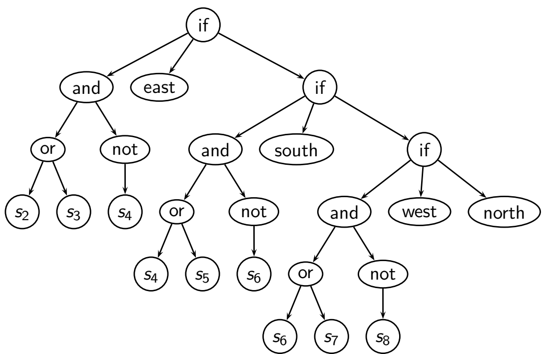
Populationsgröße $|P| = 5000$, Turnierauswahl mit Turniergröße 5

Aufbau der Nachfolgepopulation

- 10% (500) Lösungskandidaten werden unverändert übernommen
- 90% (4500) Lösungskandidaten werden durch Crossover erzeugt
- $<$1% der Lösungskandidaten werden mutiert

Stimulus-Response-Agent

Optimale, von Hand konstruierte Lösung:

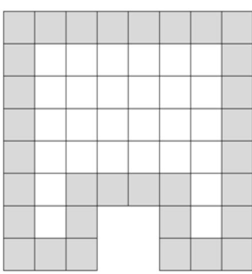


Es ist höchst unwahrscheinlich, genau diese Lösung zu finden

Um Chromosomen einfach zu halten, ist es u.U. sinnvoll, einen Strafterm zu nutzen, 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

Die 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

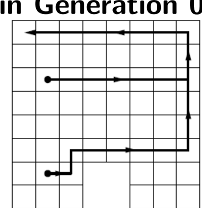
Das beste Programm hat Fitness 92

- 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))))))
```

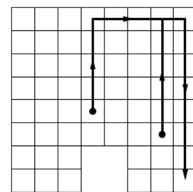
Bestes Individuum in Generation 0:



(Bewegung von zwei Startpunkten aus)

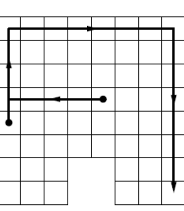
Bestes Individuum in Generation 2:

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



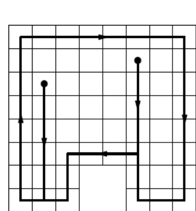
Bestes Individuum in Generation 6:

```
(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

fitness: 0, 50, 100, 150, 200, 250, 300, 350

generation number: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

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

Übersicht

1. Motivation

2. Initialisierung

3. Genetische Operatoren

4. Beispiele

5. Zusammenfassung und Ausblick
 Problem der Introns
 Erweiterungen

Problem der Introns

Mit zunehmender Generationenzahl wachen Individuen

- Grund: sogenannten **Introns**:
  - Biologie: Teile der DNA ohne Information
  - inaktive (evtl. veraltete) Abschnitte innerhalb eines Gens, welches funktionslos ist (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

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 $\Rightarrow$ 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

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.