Prof. V. Claus
N. Weicker
Abteilung Formale Konzepte

Seminar Softcomputing

Genetische Algorithmen

Ausarbeitung zum
Vortrag am 12. Mai 2000
von Philipp Cimiano Lavin,
Matrikelnummer 1849668

Inhaltsverzeichnis

Genetische Algorithmen

I. Einleitung

Die Ursprünge genetischer Algorithmen (GAs) liegen in der Biologie und wurden verwendet um biologische Vorgänge (wie z.B. Evolutionsvorgänge) zu simulieren. Ihre heutige Bedeutung verdanken die GAs dem Informatiker und Psychologen Holland, der diese erstmals auf Probleme aus praktischen Anwendungsfeldern anwandte.
GAs sind problemunabhängige Optimierungsalgorithmen und werden dementsprechend zur Lösung von beliebigen Parameter−Optimierungsproblemen eingesetzt.
Probleme dieser Art sind durch einen *Suchraum S* und einer *Fitness−Funktion f* gegeben, die jedem Punkt im Suchraum einen *Fitness−Wert* zuordnet, nach dem optimiert werden soll.
O.b.d.A. wird angenommen, dass nur positive Fitness−Werte vorkommen, d.h.

$$f : S \rightarrow R^+$$

In Anlehnung an die natürliche Evolution wird das Problem bzw. der Suchraum durch eine *Population*, d.h. eine Menge von *Individuen*, die Punkte in diesem Raum repräsentieren, dargestellt, wobei jedes Individuum als potentielle Lösung des Optimierungsproblems verstanden wird.
Durch Simulation der natürlichen Auslese wird versucht, die "fitesten" Individuen und damit das Maximum eines gegebenen Problems zu finden.
Dementsprechend ist ein GA ein zyklischer Algorithmus, der in jedem Zyklus seine Population durch *genetische Operatoren* (*Mutation, Rekombination*) manipuliert bzw. modifiziert und durch einen Selektionsmechanismus gewisse Individuen auswählt, die in die Nachfolgepopulation aufgenommen werden.
Voraussetzung dafür ist allerdings ein Evaluationsmechanismus, der es erlaubt, die Individuen aus einer gegebenen Population zu bewerten .

Das abstrakte Grundgerüst eines genetischen Algorithmus sieht nun folgendermaßen aus:

```
Time:= 0
Initialize Population(0)
Evaluate Population(0)
While Termination_Criterion # TRUE do
        Population´(t) =Recombine Population(t)
        Population´´(t)= Mutate Population ´(t)
        Evaluate Population´´(t)
        Population(t+1)=Select Population´´(t)
        Time++
od
```

II. Der GA im Detail

1. Repräsentation und Evaluation

Die Population eines GAs besteht aus einer Menge von Individuen, die in Form von Bitstrings fester Länge dargestellt werden, d.h. $\vec{a} = (a_1, ..., a_l) \in B^l$.

Falls das gegebene Problem nicht binärer Art ist $(S^n \in B^n)$, wird für die Evaluation von Individuen eine Dekodierungsfunktion benötigt, die den Bitvektor auf den für das Problem relevanten Suchraum abbildet.

Dementsprechend wird ein Bitstring $\vec{a} = (a_1, ..., a_l) \in B^l$ als Kodierung eines Vektors $\vec{x} \in S^n$ des Suchraums interpretiert, wobei der Bitstring in n logische Abschnitte gleicher Länge l_x unterteilt wird, so dass l = n?l_x und jeder Abschnitt $(a_{i,1}, ..., a_{i,l_x})$ die entsprechende Variable x_i kodiert.
Nun kann man jeden Abschnitt durch eine Funktion:

$$\Upsilon^i(a_{il}, ..., a_{il_x}) : B^{l_x} \rightarrow S$$

dekodieren, die jedem Abschnitt des Vektors einen Wert aus der entsprechenden Dimension des Suchraumes abbildet.

Nun kann die gesamt–dekodierende Funktion aus den abschnittsweise–dekodierenden Funktionen zusammengesetzt werden:

$$\Upsilon = \Upsilon^i \times .... \times \Upsilon^n$$

Es ist auch üblich keine binäre Kodierung, sondern eine Gray–Code–Interpretation der Bitvektoren zu verwenden.

Es soll darauf hingewiesen werden, dass nicht ausschließlich diskrete Optimierungsprobleme durch den Einsatz genetischer Algorithmen gelöst werden können, sondern auch kontinuierliche Suchräume, d.h. $S = \prod_{i=1}^{n} [u_i, v_i] \subseteq R^n$ betrachtet werden können.
Das setzt allerdings eine Diskretisierung des ursprünglichen kontinuierlichen Problemintervalls voraus, so dass das Ergebnis des GA immer nur eine Approximation des eigentlichen gesuchten Minimums bzw. Maximums ist.
Die abschnittsweise–dekodierende Funktion sieht in diesem Fall dann folgendermaßen aus:

$$\gamma^i(a_{il}, ..., a_{il_x}) = u_i + \frac{v_i - u_i}{2^{l_x} - 1} \cdot \left(\sum_{0}^{l_x - 1} a_{i(l_x - j)} \cdot 2^j \right)$$

Der Abstand zwischen zwei "diskreten" Punkten wird durch die Länge l_x eines

Bitvektors und durch das Intervall $[u_i,v_i]$ bestimmt:

$$\Delta x_i = \frac{v_i - u_i}{2^{l_x} - 1}$$

Die Genauigkeit kann also durch Erhöhen von l_x verbessert werden.

GAs eignen sich also optimal zur Lösung von binären Problemen, da dazu keine Kodierung notwendig ist, sie können aber durch eine entsprechende Kodierungsfunktion auch zur Lösung diskreter und durch eine Diskretisierung des Suchraumes sogar kontinuerlicher Parameter–Optimierungsproblemen eingesetzt werden.

2.Mutation

Der Mutationsoperator wurde bei GAs vorgesehen, um "zufällig" gewisse Bits eines Individuums zu invertieren und damit die Artenvielfalt zu erhalten. Die Mutationswahrscheinlichkeit p_m wird dabei oft sehr klein gewählt und liegt häufig im Intervall [0.005,0.01].
Formal gesehen ist der Mutationsoperator eine Funktion, die auf der gesamten Population operiert:

$$M'_{\{p_m\}}: I^{\mu} \rightarrow I^{\mu}$$

$$M'_{\{p_m\}}(\vec{a}_1,\ldots,\vec{a}_l) = (m'_{\{p_m\}}(\vec{a}_1),\ldots,m'_{\{p_m\}}(\vec{a}_l)),$$

wobei $m'_{\{p_m\}}: I \rightarrow I$ eine Funktion ist, die ein Individuum mutiert indem sie einen neuen Bitvektor $\vec{a}' = (a_1',\ldots,a_l') = m'_{\{p_m\}}(a_1,\ldots,a_l) = m'_{\{p_m\}}(\vec{a})$ erzeugt, und zwar nach folgender Vorschrift:

$$\forall i \in \{1,\ldots,l\}:$$

$$a_i' = \begin{cases} a_i, & \text{if } x_i > p_m \\ 1-a_i, & \text{if } x_i \leq p_m \end{cases}$$

wobei x_i eine Zufallsvariable ist, die im Intervall [0,1] liegt und die für jedes Bit neu erzeugt wird.

3.Rekombination

Die Idee, die sich hinter dem Rekombinationsprinzip verbirgt, ist, dass nützliche Abschnitte verschiedener Elternindividuen kombiniert werden, um ein neues Individuum zu erzeugen, das von der vorteilhaften Bitkombination seiner Eltern profitiert.
Der entsprechende Operator heißt *Crossover−Operator* und hat folgende formale Definition:

$$
\begin{aligned}
&R_{\{p_c\}} : I^{\mu} \to I^{\mu} \\
&R'_{\{p_c\}}(\vec{a}_1,\ldots,\vec{a}_{\mu}) = (r'_{\{p_c\}}(\vec{a}_{x_1},\vec{a}_{x_2}),\ldots, r'_{\{p_c\}}(\vec{a}_{x_{2\mu-1}},\vec{a}_{x_{2\mu}})),
\end{aligned}
$$

wobei $x_i \in \{1,\ldots,\mu\}$ zufallsverteilt ist und bei der Anwendung des Operators 2μ Elternindividuen zufällig aus der Population ausgewählt und jeweils paarweise mit Wahrscheinlichkeit p_c durch die *Crossover−Funktion* $r'_{\{p_c\}} : I^2 \to I$ zu einem neuen Individuum rekombiniert werden.
Die Crossover−Wahrscheinlichkeit liegt dabei sehr häufig im Intervall [0.75,0.95].
Man unterscheidet hauptsächlich folgende Arten von Crossover:

a.One−point−crossover:

Es wird zufällig eine Crossover−Position

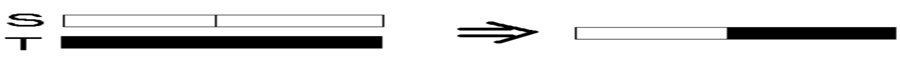
$$X \in \{1,\ldots,l\}$$

ermittelt und die Bits rechts von der Position zwischen den beiden Individuen vertauscht, wobei eines der beiden entstehenden Individuen verworfen wird. Die Crossover−Funktion sieht folgendermaßen aus:

$$
r'_{\{p_c\}}(\vec{a}_S, \vec{a}_T) = \begin{cases} (a_{S,1},\ldots,a_{S,X-1},a_{T,X},\ldots,a_{T,l}), & x \le p_c \\ \vec{a}_S, & x > p_c \end{cases},
$$

wobei $x \in [0,1]$ eine Zufallsvariable ist, die für jede Anwendung der Crossover−Funktion erzeugt wird.

Anschaulich:



b.Multi−point−crossover:

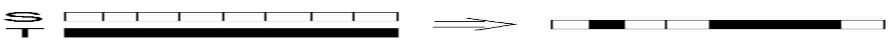
Der Multi−point−crossover−Operator ist eine Verallgemeinerung des One−point−crossover und erlaubt z Crossover−Positionen, wobei jeder zweite Abschnitt zwischen Positionen vertauscht

wird. Dazu werden Crossover–Punkte $X_1',...,X_z' \in \{1,...,l-1\}$ zufällig ausgewählt und, falls z ungerade ist, wird definitionsgemäß zusätzlich der Punkt $X_{z+1}=l$ eingeführt.
Die formale Definition für die Z–point–crossover–Funktion $r_{\{pc,z\}}$ lautet:

$$\begin{aligned}
&geg.:\ Zufallszahlen: X_1',...,X_z' \in \{1,...,l-1\},\ X_k' < X_{k+1}' \\
&und\ \ X_{z+1}' = l,\ falls\ \ z\ \ ungerade: \\
&\qquad \vec{a}' = r_{\{p_c,z\}}(\vec{a}_S, \vec{a}_T), \quad wobei \\
&\qquad a_i' = \begin{cases} a_{S,i},\ \ falls\ x > p_c \\ \begin{cases} a_{S,i}\ \ \forall k \leq z,\ k\ \ ungerade \\ \quad (X_k' < i \leq X_{k+1}') \\ \\ \quad a_{T,i}\ ,sonst \end{cases} ,falls\ x \leq p_c \end{cases}
\end{aligned}$$

wobei die Indices S und T die beiden zufällig ausgewählten Elternindividuen bezeichnen.
Für jede Anwendung des Multi–point–crossover–Funktion wird zufällig ein Wert für die Zufallsvariable x ermittelt und je nachdem ob ihr Wert größer bzw. kleiner als p_c ist, werden Segmente vertauscht oder nicht.
Für jede Anwendung des Multi–point–crossover–Funktion muß also ein Wert für die Variable x, genauso wie z Crossover–Positionen ermittelt werden.

Beispiel für z=4:



c.Uniform–crossover:

Der Uniform–crossover ist ein Sonderfall des Z–point–crossover für z=l. Hier funktioniert die Rekombination so, dass für jedes Bit des neuen Individuums zufällig eines der beiden Elternbits übernommen wird:

$$\begin{aligned}
&r_{\{pc\}}(\vec{a}_S, \vec{a}_T) = \vec{a}' = (a_1',...,a_l'), \\
&wobei\ \ a_i' = \begin{cases} a_{S,i},\ falls\ x > p_c \\ a_{S,i} \vee a_{T,i},\ falls\ x \leq p_c \end{cases}
\end{aligned}$$

wobei $x \in [0,1]$ wieder eine Zufallsvariable ist und die Entscheidung, ob $a_{S,i}$ oder $a_{T,i}$ gewählt zufällig getroffen wird ("Münzwurf").

Es sei darauf hingewiesen, dass es noch weitere Arten von Crossover gibt. (vergleiche [Bäck] S. 105–106

4.Selektion

Der wohl wichtigste Baustein eines GAs ist der Selektionsmechanismus, der für jedes Individuum eine Überlebenswahrscheinlichkeit berechnet und eine sozusagen "natürliche" Auslese vornimmt.

Dazu werden die Überlebenschancen p_S eines Individuums berechnet, und zwar durch folgende Formel:

$$p_S(\vec{a}_i(t)) = \frac{\Phi(\vec{a}_i(t))}{\sum_{j=1}^{\mu} \Phi(\vec{a}_j(t))},$$

$$\textit{wobei } \Phi(\vec{a}) = \delta(f(\Upsilon(\vec{a})), \Theta_\delta)$$
$$\textit{und } \delta : R^+ \times \Theta_\delta \to R^+$$

Dabei ist δ eine sogenannte skalierende Funktion, die dazu dient, die Fitness–Werte, die eventuell sehr dicht beieinander liegen können, auf einen größeren Bereich zu verteilen, um damit deutlicher die "schlechten" von den "guten" Individuen unterscheiden zu können; Θ_δ steht dabei für zusätzliche Parameter, die der skalierenden Funktion übergeben werden müssen, wie z.B. die beiden Konstanten c_0 und c_1 bei der linearen Skalierung (s. u.)
Es gibt eine ganze Vielfalt von skalierenden Funktionen, die z.B. bei [Bäck] S. 101–102 nachgeschlagen werden können.

a.Linear static scaling:

$$\delta(f(\Upsilon(\vec{a}), \{c_0, c_1\}) = c_0 \cdot f(\Upsilon(\vec{a})) + c_1,$$
$$\textit{wobei } c_0 \in R - \{0\},\ c_1 \in R$$

b.Logarithmic scaling:

$$\delta(f(\Upsilon(\vec{a})), \{c_0\}) = c_0 - \log(f(\Upsilon(\vec{a})),$$
$$\textit{wobei } c_0 > \log(f(\vec{x}))\ \forall \vec{x} \in \prod_{i=1}^{n} [u_i, v_i]$$

c.Exponential scaling:

$$\delta(f(\Upsilon(\vec{a}), \{c_0, c_1, c_2\}) = (c_0 \cdot f(\Upsilon(\vec{a}) + c_1)^{c_2}$$

d.Linear dynamic scaling with scaling window:

$$\delta(f(\Upsilon(\vec{a}), \{P(t-\omega)\}) = -f(\Upsilon(\vec{a})) + \max\{f(\Upsilon(\vec{a}_j)) \mid \vec{a}_j \in P(t-\omega)\},$$

diese Skalierungsfunktion zieht also auch noch die Population ω *Generationen vorher in Betracht*; *derParameter* ω *wird scaling window genannt*.

Anhand der Überlebenschancen eines Individuums kann man nun die Anzahl der erwarteten Kopien η in der nächsten Generation (auch *Erwartungswert* genannt) berechnen:

$$\eta(\vec{a_i}(t)) = \mu \cdot p_S(\vec{a_i}(t))$$

Dabei ist zu beachten, dass η einen reellen Wert annimmt, dass aber eine diskrete Anzahl von Kopien eines jeden Individuums in die Nachfolgegeneration übernommen werden muss.
Dazu braucht man ein Verfahren, das die Erwartungswerte auf ganze Zahlen abbildet, wobei die Diskrepanz zwischen Erwartungswert und übernommenen Kopien möglichst gering gehalten werden soll.
Ein verbreitetes Verfahren ist die *Roulette−wheel−Methode,* bei der jedes Individuum auf einen Abschnitt abgebildet wird, dessen Größe seinem Erwartungswert entspricht.
Dann werden μ Zufallszahlen aus dem Intervall [0,μ) generiert und das Individuum, in dessen Abschnitt die Zufallszahl liegt, ausgewählt.

Beispiel:

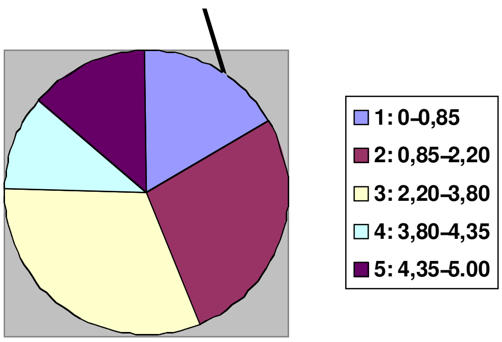
Nehmen wir an, wir hätten zu einem beliebigen Zeitpunkt t eine Population bestehend aus 5 Individuen, deren Fitness−Werte der unteren Tabelle zu entnehmen sind.
Da diese so nahe beieinander liegen werden sie linear skaliert, und zwar durch folgende Funktion:

$$\delta(\vec{a}) = 100 \cdot f(\Upsilon(\vec{a})) - 2080$$

Nun kann man die

	f (Fitness)	Φ(skalierte Fitness)	p_S(Überlebenschancen)	η(Erwartungswert)
I_1	20.91	11	0,17	0,85
I_2	20.98	18	0,27	1,35
I_3	21.01	21	0,32	1,60
I_4	20.87	7	0,11	0,55
I_5	20.89	9	0,14	0,70

entsprechende Roulette−wheel−Verteilung angeben:



5. Initialisierung der Population

Die Startpopulation P(0) eines GA ist eine Menge von μ Bitvektoren der Länge l, d.h.

$$P(0) = \{\vec{a}_1(0),...,\vec{a}_\mu(0)\}$$

wobei jedes Bit dieser Bitvektoren zufällig erzeugt wird.
Dabei ist die Randbedingung zu beachten, dass sich die dekodierten Variablen x_i auch wirklich im Suchraum S befinden.
Nehmen wir z.B. an, unserer Suchraum S wäre {1,...,10}. Für die Kodierung dieses Suchraumes bräuchten wir mindestens einen Bitvektor der Länge 4. Wenn wir nun die bei der Initialisierung zufällig erzeugten Bitvektoren dadurch dekodieren, dass wir sie einfach in die Basis 10 umwandeln, könnte auch z.B. der Wert $(1111)_2=(15)_{10}=15$ herauskommen, der offensichtlich nicht im Suchraum S liegt.
Aus diesem Grund ist die oben aufgeführte Randbedingung von großer Wichtigkeit.

6. Terminierung

Die Terminierung eines GA wird sehr oft durch ein Vielfältigkeitsmaß bestimmt, das die Durchschnittskonvergenz b(P(t)) aller Bitpositionen innerhalb der gesamten Population berechnet:

$$b(P(t)) = \frac{1}{l \cdot \mu} \sum_{j=1}^{l} \max\left\{ \sum_{i=1}^{\mu} (1 - a_{i,j}), \sum_{i=1}^{\mu} a_{i,j} \right\} \in [0.5, 1.0],$$

wobei $a_{i,j}$ das j–te Bit des i–ten Individuums der Population bezeichnet.
b(P(t)) gibt in diesem Sinne die Durchschnittsanzahl der an allen Position der gesamten Individuen der Population am meisten vorkommenden Wertes an. Große Werte von b stehen dementsprechend für eine geringe, kleinere Werte dagegen für eine große "Artenvielfalt".
Nun ergibt sich das Terminierungskriterium $\tau(P(t))$ eines GAs folgendermaßen:

$$\tau(P(t)) = \begin{cases} true, & if\, b(P(t)) > b_{\max} \\ false, & otherwise \end{cases},$$

wobei b_{max} häufig einen Wert aus dem Intervall [0.95,0.99] einnimmt.

Oftmals wird aber auch die Anzahl der Generationen, die ein GA durchlaufen soll, fest gewählt, so dass ein weiteres Abbruchkriterium entfällt.

7. Formale Definition eines GA und Skizze des Grundalgorithmus

Ein GA ist formal ein 7–Tupel:

$GA = (I, \Phi, \Omega, \Psi, s, \iota, \mu)$:

1) $I = B^l$

2) $\Phi(\vec{a}) = \delta\,(f(\Upsilon(\vec{a})), \Theta_\delta)$,

$\quad$ *wobei* $\delta : R^+ \times \Theta_\delta \rightarrow R^+$

3) $\Omega = \{M_{\{p_m\}} : I^\mu \rightarrow I^\mu, R_{\{p_c, z\}} : I^\mu \rightarrow I^\mu\}$

4) $\Psi(P) = s(M_{p_m}(R_{\{p_c, z\}}(P)))$

5) $s : I^\mu \rightarrow I^\mu$

6) $\tau : I^\mu \rightarrow \{true, false\}$

7) μ : *Anzahl der Individuen*

Diese formale Beschreibung von GAs erlaubt es, genetische Algorithmen auf einfache und abstrakte Weise (unabhängig von einer Programmiersprache) zu formulieren, so dass sie einfach nachvollzogen und ausprogrammiert werden können.

Der genetische Grundalgorithmus sieht nun wie folgt aus:

$$
\begin{array}{l}
t := 0; \\
\textit{initialize} : P(0) := (\vec{a}_1(0), \ldots, \vec{a}_\mu(0)) \in I^\mu; \\
\textit{while}\ (\tau(P(t)) \neq \textit{true})\ \textit{do} \\
\quad \textit{recombine} : P'(t) := R'_{\{p_c, z\}}(P(t)); \\
\quad \textit{mutate} : P''(t) := M'_{\{p_m\}}(P'(t)); \\
\quad \textit{evaluate} : P''(t) := (\vec{a}_1''(t), \ldots, \vec{a}_\mu''(t) : (\Phi(\vec{a}_1''(0)), \ldots, \Phi(\vec{a}_\mu''(0))), \\
\qquad\qquad \textit{wobei}\ \Phi(\vec{a}_k''(t)) = \delta\,(f(\Upsilon(\vec{a}_k''(t))), P(t - \omega)); \\
\quad \textit{select} : P(t+1) := s(P''(t)) \\
\quad t := t + 1 \\
\textit{od}
\end{array}
$$

Dabei geht die Evaluierung $\Phi(\vec{a}_k)$ eines Individuums in die Berechnung der Überlebenswahrscheinlichkeit p_s und damit in den Erwartungswert μ eines Individuums ein und bestimmt dadurch die Selektion.

Es sei darauf hingewiesen, dass der in [Bäck] präsentierte Formalismus nur im Wesentlichen übernommen wurde.
Bäck stellt eine Population P als eine Menge und nicht als ein Tupel dar, schreibt aber widersprüchlicherweise ebenso $P(t) \in I^\mu$.
Außerdem erfolgt bei ihm die erste Evaluierung vor der while–Schleife, ein Vorgehen, welches vollkommen sinnlos erscheint, da noch nicht rekombiniert und mutiert worden ist.
Hinzu kommt, dass seine genetischen Operatoren zwar laut Definition (siehe [Bäck] 2.32 u. 2.33) nicht auf der gesamten Population operieren, sondern auf Individuen (Mutation) bzw. auf

Individuenpaaren (Rekombination), bei seiner Formulierung des obigen GAs hingegen auf die gesamte Population angewendet werden.

8. Analyse genetischer Algorithmen

Zur Analyse von GAs wurde der Begriff des Schemas eingeführt. Ein Schema beschreibt eine Teilmenge von Bitvektoren, die an bestimmten Stellen gleich sind und ist als Bitvektor der Länge l über dem Alphabet {0,1,*} definiert.
Weiter ist ein Bitvektor $\vec{a} \in B^l$ eine Instanz eines Schemas H, wenn jedes "definierte" Bit (d.h. 0 oder 1) von H mit dem Bit an der gleichen Stelle in a übereinstimmt.
Nun kann man die Menge I(H) aller Instanzen eines Schemas H definieren:

$$I : \{0,1,*\}^l \rightarrow 2^l \;\; mit$$
$$I(H) = \{(a_1,...,a_l) \mid \forall i \in \{1,...,l\} : h_i \in \{0,1\} => a_i = h_i\}$$

Außerdem wurden drei weitere Maße für Schemen eingeführt:

1) Die *Ordnung o(H)* eines Schemas:

$$o : \{0,1,*\} \rightarrow \{0,...,l\} mit$$
$$o(H) = | \{ i \mid h_i \in \{0,1\} \} |$$

2) Die *definierende Länge* eines Schemas:

$$\Delta H : \{0,1,*\}^l \rightarrow \{0,...,l\} \;\; mit$$
$$\Delta H = \max\{ i \mid h_i \in \{0,1\} \} - \min\{ i \mid h_i \in \{0,1\} \}$$

3) und die *durchschnittliche Schema−Fitness* $\Phi(H(t))$ bezüglich einer Population P(t):

$$\Phi(H(t)) = \frac{1}{N(H(t))} \sum_{\vec{a}(t) \in I(H(t)) \cap P(t)} \Phi(\vec{a}(t))$$

(N(H(t)) bezeichnet die Anzahl der Instanzen eines Schemas H zur Generation t)

im Vergleich zur *durchschnittlichen Fitness* $\overline{\Phi}(t)$ der Population:

$$\overline{\Phi}(t) = \frac{1}{\mu} \sum_{i=1}^{\mu} \Phi(\vec{a}_i(t))$$

<u>Satz: (**Schema Theorem**)</u>

$$N(H(t+1)) = N(H(t)) \cdot \frac{\Phi(H(t))}{\Phi(t)} \cdot \left[1 - p_c \cdot \frac{\Delta(H(t))}{l-1} - o(H(t)) \cdot p_m \right]$$

Die Kernaussage dieses Satzes ist, dass die Mächtigkeit der Menge der Instanzen **kurzer Schemen** (d.h. mit einer kleinen definierenden Länge), **überdurchschnittlicher Fitness und kleiner Ordnung** exponentiell in den Nachfolgergenerationen wächst.

III. Anwendung eines GA zur Lösung des ROP

1. Beschreibung und formale Definition des Rucksackoptimierungsproblems (ROP):

Das ROP stellt informell gesehen das Problem dar, eine bestimmte Anzahl von "Gegenständen" mit einer bestimmten Masse bzw. Volumen in einem Behälter (Rucksack) mit einer gegebenen Kapazität so zu verstauen, dass dieser optimal ausgenutzt wird.
Dabei müssen i.A. natürlich nicht alle Gegenstände in den Rucksack "gepackt" werden; Ziel ist es vielmehr eine sinnvolle Auswahl von Gegenständen zu treffen, die den Platz im Rucksack möglichst optimal in dem Sinne ausnutzt, dass die Gesamtmasse bzw. das Gesamtvolumen der "gepackten" Gegenstände maximiert und der freie Platz im Rucksack dementsprechend minimiert wird.
Nun folgt eine vollständige, formale Definition des Problems; es sei darauf hingewiesen, dass das hier präsentierte ROP eine Abwandlung des Rucksackproblems ist, das in [Schön] S. 164 vorgestellt wird.

Definition: 1) Gegeben: eine Menge $M=\{m_1,...,m_l\}$ l natürlicher Zahlen, die die Größe, Volumen oder Gewicht von Gegenständen darstellen.

2) die Kapazität bzw. Größe des Rucksacks

Gesucht: $T \subseteq \{1,2,...,l\}$ mit Summe= $\sum_{i\in T} a_i \le b \wedge$ Summe maximal

2. Anwendung eines GA zur Lösung des ROP:

a. Suchraum und Fitness–Funktion:

Um das ROP mit Hilfe von einem GA lösen zu können, ist als aller erstes ein eindeutig bestimmter Suchraum erforderlich, der intuitiv gesehen aus allen Möglichkeiten besteht,
wie der "Rucksack gepackt" werden kann, d.h. die Potenzmenge P(M).
Außerdem brauchen wir noch eine Fitness–Funktion f, die jeder Teilmenge $T \in P(M)$ eine bestimmte "Bewertung" zuordnet. Am einfachsten wird jeder Teilmenge T sie Summe über der "Größe" der in ihr enthaltenen Gegenstände zugeordnet, und falls damit die Kapazität des Rucksacks überschritten wird, dementsprechend der negative Überschuss, d.h.

$$f : T \subseteq \{1,...,l\} \rightarrow N \ mit$$

$$f(T) = \begin{cases} \sum_{i \in T} a \ , \ \ falls \sum_{i \in T} a_i \leq b \\ \quad b - \sum_{i \in T} a_i, \ \ sonst \end{cases}$$

b.<u>Repräsentation und Kodierung:</u>

Wesentlich ist die Repräsentation und die Kodierung bzw. die Dekodierung des Suchraumes. Die Repräsentation eines Punktes im Suchraum scheint intuitiv klar; man betrachtet einfach Individuen der Länge l, d.h. $\vec{a} = (\vec{a}_1,...,\vec{a}_l) \in B^l$, wobei jedes Bit angibt, ob sich ein bestimmter Gegenstand in der Teilmenge T befindet.

Damit ergibt sich folgende Dekodierungsfunktion:

$$\Upsilon : I \rightarrow T \subseteq \{1,...,l\} \ mit$$
$$i \in T \Leftrightarrow b_i = 1$$

c.<u>Evaluierung:</u>

Die oben angegebene Fitness–Funktion f(T) kann auch ins Negative gehen, so dass wir eine skalierende Funktion benötigen, die uns den Wertebereich von f(T) nach R^+ abbildet, z.B. lineare Skalierung:

$$\delta : R \rightarrow R^+ \ mit$$
$$\delta(\vec{b}) = f(\Upsilon(\vec{b})) + l \cdot \max\{a_i \mid a_i \in M\}$$

Nun können wir die Evaluierungsfunktion angeben, die wiederum die Selektion festlegt:

$$\Phi(\vec{b}) = \delta(f(\Upsilon(\vec{b})))$$

d.<u>Spezifikation des GAs:</u>

Bisher wurde die Repräsentation, Kodierung und die entsprechende Dekodierung des Suchraumes festgelegt. Ebenso wurde die Evaluierungsfunktion angegeben. Zur vollständigen Spezifikation des GAs zur Lösung des ROP müsste noch ein Selektionsmechanismus genauso wie der Mutations– und Rekombinationsoperator definiert werden. Vermutlich wäre eine Entscheidung zugunsten von Uniform–crossover sinnvoll, da dies eher dem intuitiven Verfahren zur Rucksackoptimierung entspricht, d.h. man vertauscht nur einzelne Gegenstände und nicht ganze Gruppen von Gegenständen.
Außerdem müsste eine Populationsgröße und die Mutations– und Crossover–Wahrscheinlichkeiten gewählt werden. Dadurch wäre der GA vollständig bestimmt.

IV. Diskussion

Bei der Diskussionsrunde im Anschluss an den Vortrag tauchen einige Fragen zum ROP auf.
Es wird die Frage gestellt, ob das Problem so abgewandelt werden kann, dass mehrere Rucksäcke in Betracht gezogen werden können.

Das ist ohne größeren Aufwand möglich: gegeben sei eine Menge $M = \{a_1,...,a_k\}$ von Gegenständen und eine Anzahl n von Rucksäcken der Kapazität c_i. Ein Individuum $\vec{b}$ ist dann ein Bitvektor der Länge $l = k \cdot n$, wobei jeder Abschnitt der Länge k einen Rucksack kodiert.
Nun braucht man eine abschnittsweise–dekodierende Funktion Υ^i der Form:

$$\Upsilon^i : B^k \rightarrow T_i \subseteq \{1,...,k\} \text{ mit}$$

$$j \in T_i \Leftrightarrow b_{k \cdot i + j} = 1$$

Die Fitness–Funktion könnte dann folgendermaßen aussehen:

$$f : T = T_1 \times ... \times T_n \rightarrow R \text{ mit}$$

$$f(T) = \frac{\sum_{i=1}^{n} k_i}{n}, wobei$$

$$k_i = \begin{cases} \sum_{j \in T_i} b_i, \ falls \ \sum_{j \in T_i} b_i \leq c_i \\ c - \sum_{j \in T_i} b_i, \ sonst \end{cases}$$

Sie gibt die Durchschnittsauslastung aller Rucksäcke an.

Anschließend tauchen Zweifel darüber auf, inwieweit GAs wirklich als universelle Problemlösungsalgorithmen betrachtet werden können und ob es immer möglich ist, den Suchraum eines gegebenen Problems sinnvoll und erfolgreich zu kodieren.
Diese Zweifel sind durchaus berechtigt, denn die wohl schwerste und heikelste Aufgabe beim Entwurf eines GAs zur Lösung eines Problems ist die Kodierung des Suchraumes, denn von ihr hängt direkt die Wirkung der genetischen Operatoren und damit das Konvergenzverhalten der Population ab.
Theoretisch ist es möglich, jedes Problem zu kodieren, doch in der Praxis liegt die Schwierigkeit darin, eine möglichst günstige Kodierung zu finden, die die Population konvergieren lässt. Diese Aufgabe kann meistens nur intuitiv gelöst werden, da es dazu keine theoretische Basis gibt.

Im weiteren Gespräch wird der Vorschlag gemacht, die Rekombination auf die komponentenkodierenden Abschnitte zu begrenzen statt einen abschnittsübergreifenden Crossover zu erlauben. Dazu hat es schon Versuche gegeben, aber leider keine Aussage darüber, ob es wirklich sinnvoll ist. (siehe [Bäck] S. 106).

Von der Seminarleiterin N. Weicker werden noch zwei weitere Problemstellungen angesprochen.
Zum einen sollen Maßnahmen gefunden werden, um die Artenvielfalt zu erhalten und um zu vermeiden, dass eine Population zu einseitig wird, d.h. zum größten Teil aus Individuen mit dicht beieinander liegenden Fitness–Werten besteht.
Folgende Maßnahmen wurden genannt:

1) Population groß genug wählen

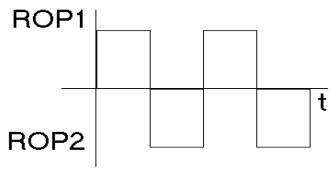
2) mehr Mutation bzw. Makromutation (z.B. durch Erhöhen der Mutationswahrscheinlichkeit p_m)

3) nichtlineare Selektion (z.B. logarithmische oder sigmoide Selektion)

4) durchschnittliche Fittness berücksichtigen, nicht nur Individuen mit besten Fitness–Werten selektieren

Zum anderen sollte das ROP so modifiziert werden, dass ein Rucksack nach zwei verschiedenen Kriterien, d.h. nach zwei verschiedenen Fitness–Funktionen, "gepackt" werden soll und dass die Population des GAs ständig zwischen den zwei Lösungen für die beiden Rucksackprobleme hin– und herschwingt.

Anschaulich:



Zu dieser Problemstellung wird vorgeschlagen, beide ROPs in einem Individuen zu kodieren und beide Fitnessfunktionen so zu einer zu kombinieren, dass jedes Individuum eine Lösung beider Probleme darstellt. Alternativ dazu wird vorgeschlagen, die Population so in zwei Klassen zu teilen, dass eine Klasse von Individuen die Lösung eines der beiden Probleme darstellt.
Eine andere Idee besteht darin, Individuen, die bereits eine gute Lösung für eines der beiden Probleme darstellen, zu erhalten und vom Mutations/Rekombinations–Zyklus des GAs auszuschließen.
Insgesamt sollte eine schnelle Optimierung zum "guten" Individuum angestrebt werden, was durch eine hohe "Flexibilität" der Individuen und durch eine hohe Artenvielfalt begünstigt werden kann.

V. Zusammenfassung

GAs sind universelle und problemunabhängige Algorithmen zur Lösung binärer, allgemeiner diskreter und sogar kontinuierlicher Parameter−Optimierungsproblemen, die durch einen Suchraum und einer Fitness−Funktion gegeben sind.
Jedes Individuum seiner Population stellt einen Punkt in diesem Suchraum und damit eine potentielle Lösung des Problems dar.
Auf dieser Population wird eine natürliche Auslese simuliert indem sie immer wieder einen Zyklus durchläuft, in dem sie mutiert und rekombiniert wird und gewisse Individuen entsprechend ihrer Fitness−Werte selektiert bzw. vermehrt und in die Nachfolgepopulation aufgenommen werden. Das Ziel dieser Simulation ist, dass am Ende des Zyklus, das entweder durch ein Abbruchkriterium reguliert oder nach einer festgelegten Anzahl von Iterationen erreicht wird, die Population zu einer guten Lösung des Optimierungs−Problems konvergiert.
Die Theorie, die den GAs zugrunde liegt, besagt sogar, dass unter günstigen Anforderungen an eine Lösung des Problems, sich die Individuen, die diese Anforderungen erfüllen, exponentiell vermehren werden und damit die Population auch konvergieren wird.

VI. Literaturverzeichnis

[Bäck] Bäck, Thomas, *Evolutionary Algorithms in Theory and Practice*
Universität Dortmund, Fachbereich Informatik, 1994

[Schön] Schöning, Uwe, *Theoretische Informatik, kurzgefaßt*
Spektrum , Akad. Verl., 1997