

Genetische Algorithmen

Christian Borgelt

Arbeitsgruppe Neuronale Netze und Fuzzy-Systeme
Institut für Wissens- und Sprachverarbeitung
Otto-von-Guericke-Universität Magdeburg
Universitätsplatz 2, D-39106 Magdeburg
borgelt@ivs.cs.uni-magdeburg.de
<http://fuzzy.cs.uni-magdeburg.de/~borgelt/>
<http://fuzzy.cs.uni-magdeburg.de/studium/ga/>

Christian Borgelt

Genetische Algorithmen

Genetische Algorithmen: Einleitung

Einordnung: Soft Computing

Soft Computing = **neuronale Netze** (Parallelvorlesung)
+ **Fuzzy-Systeme** (im Wintersemester)
+ **genetische Algorithmen**

Soft Computing ist charakterisiert durch:

- Meist „modellfreie“ Ansätze
(d.h., es ist kein explizites Modell des zu beschreibenden Gegenstandsbereichs notwendig; „modellbasiert“ dagegen; z.B. Lösen von Differentialgleichungen)
- Approximation statt exakte Lösung (nicht immer ausreichend!)
- Schnelleres Finden einer brauchbaren Lösung,
u.U. auch ohne tiefgehende Problemanalyse

Christian Borgelt

Genetische Algorithmen

Genetische Algorithmen: Anwendungsgebiete

Allgemein: Lösen von Optimierungsproblemen

- Gegeben:
 - ein Suchraum S
 - eine zu optimierende Funktion $f : S \rightarrow \mathbb{R}$
 - ggf. einzuhaltende Nebenbedingungen
- Gesucht: Ein Element $s \in S$, das die Funktion f optimiert.
- **Prinzipielle Lösungsansätze:**
 - *analytische Lösung:* sehr effizient,
aber nur in seltenen Fällen anwendbar
 - *vollständige Durchforstung:* sehr ineffizient, daher nur bei sehr kleinen Suchräumen anwendbar
 - *blinde Zufallsuche:* immer anwendbar,
aber meist sehr ineffizient
 - *gesteuerte Suche:* Voraussetzung: Funktionswerte ähnlicher
Elemente des Suchraums sind ähnlich.

Christian Borgelt

Genetische Algorithmen

Genetische Algorithmen: Anwendungsgebiete

Beispiele für Optimierungsprobleme

- **Parameteroptimierung**
z.B. Krümmung von Rohren für minimalen Widerstand
Allgemein: Finden eines Parametersatzes, so daß eine gegebene reellwertige Funktion ein (möglichst globales) Optimum annimmt.
- **Packprobleme**
z.B. Füllen eines Rucksacks mit maximalem Wert oder
Packen möglichst weniger Kisten mit gegebenen Gütern
- **Wegeprobleme**
z.B. Problem des Handlungsreisenden (Anwendung: Bohren von Plattformen)
Reihenfolge von anzufahrenden Zielen, Fahrtroutenoptimierung
Verlegen von Leiterbahnen auf Platinen und in integrierten Schaltkreisen

Christian Borgelt

Genetische Algorithmen

4

Genetische Algorithmen: Anwendungsgebiete

- **Anordnungsprobleme**
z.B. Steinerproblem (facility allocation problem);
Positionierung von Verteilerknoten z.B. in einem Telekommnetz
- **Planungsprobleme**
z.B. Ablaufpläne (Scheduling), Arbeitspläne, Operationenfolgen
(auch z.B. zur Optimierung in Compilern — Umordnung der Befehle)
- **Strategieprobleme**
z.B. Gefangenendilemma und andere Modelle der Spieltheorie,
Verhaltensmodellierung von Akteuren im Wirtschaftslieben
- **biologische Modellbildung**
z.B. Netzsprinter (regelbasiertes Modell, das beschreibt, wie eine Spinne ihr Netz baut; Parameter werden durch einen genetischen Algorithmus optimiert und mit Beobachtungen verglichen; liefert recht gutes Modell)

Christian Borgelt

Genetische Algorithmen

5

Biologische Grundlagen

Christian Borgelt

Genetische Algorithmen

6

Genetische Algorithmen: Motivation

- Genetische Algorithmen basieren auf der **biologischen Evolutionstheorie** Charles R. Darwin: “On the Origin of Species by Means of Natural Selection” („Die Entstehung der Arten durch natürliche Zuchtwahl“), London 1859
Empfehlenswerte Literatur zur biologischen Evolutionstheorie sind speziell die Bücher von Richard Dawkins, z.B. “The Selfish Gene” („Das egoistische Gen“) und “The Blind Watchmaker” („Der blinde Uhrmacher“).
- Grundsätzliches Prinzip:
Durch zufällige Variation entstehende vorteilhafte Eigenschaften werden durch natürliche Auslese ausgewählt.
(Individuen mit vorteilhaften Eigenschaften haben bessere Fortpflanzungs- und Vermehrungschancen — „differentielle Reproduktion“.)
- Die Evolutionstheorie erklärt die Vielfalt und Komplexität der Lebewesen und erlaubt es, alle Disziplinen der Biologie zu vereinen.

Christian Borgelt

Genetische Algorithmen

7

Prinzipien der organismischen Evolution I

- **Diversität**
Alle Lebewesen, sogar solche innerhalb ein und derselben Art, sind voneinander *verschieden*, und zwar bereits in ihrem Erbgut (*Vielfalt des Lebens*). Gleichwohl bilden die tatsächlich existierenden Formen von Lebewesen nur einen winzigen Bruchteil der im Prinzip möglichen.
- **Variation**
Es entstehen, durch Mutation und genetische Rekombination (sexuelle Fortpflanzung), laufend *neue Varianten*.
- **Verehrbung**
Die Variationen sind, soweit sie in die Keimbahn gelangen, *erbtlich*, werden also genetisch an die nächste Generation weitergegeben.
(i.a. keine Verehrbung von erworbenen Eigenschaften — sog. *Lamarckismus*)

(nach Gerhard Volpert: „Der wissenschaftstheoretische Status der Evolutionstheorie — Einwände und Gegenargumente“ in: „Biophilosophie“, Reclam, Stuttgart 1995)

Christian Borgelt

Genetische Algorithmen

8

Prinzipien der organismischen Evolution II

- **Artbildung**
Es kommt zur *genetischen Divergenz* von Individuen und Populationen; es entstehen neue *Arten*, deren Vertreter nicht mehr fruchtbar miteinander kreuzbar sind. Die Artbildung verläuft dem phylogenetischen (stammesgeschichtlichen) „Stammbaum“ seine charakteristische Verzweigungsstruktur.
- **Überproduktion**
Fast alle Lebewesen erzeugen *mehr Nachkommen*, als jemals zur Reproduktionstrafe kommen können.
- **Anpassung / natürliche Auslese / differentielle Reproduktion**
Im Durchschnitt weisen die Überlebenden einer Population solche erblichen Variationen auf, die ihre *Anpassung* an die lokale Umgebung *erhöhen*.
Herbert Spencers Redewendung vom „Überleben der Tauglichsten“ (“survival of the fittest”) ist allerdings eher irreführend; besser spricht man von „unterschiedlicher Vermehrung aufgrund unterschiedlicher Tauglichkeit“.

Christian Bergelt

Genetische Algorithmen

9

Prinzipien der organismischen Evolution III

- **Zufälligkeit / blinde Variation**
Variationen sind *zufällig* zwar *ausgelöst*, *beeirkt*, *verursacht*, aber nicht *vorausgesetzt* auf bestimmte Mechanale oder günstige Anpassungen ausgerichtet (*nicht teleologisch*, von griech. *τέλος* — Ziel; Zweck).
- **Gradualismus**
Variationen erfolgen in vergleichsweise *kleinen Schufen*, gemessen am gesamten Informationsgehalt oder an der Komplexität des Organismus. Deshalb sind phylogenetische (stammesgeschichtliche) Veränderungen *graduell* und relativ langsam. (Gegensatz: Saltationismus — große Entwicklungssprünge)
- **Evolution / Transmutation / Vererbung mit Modifikation**
Wegen der Anpassung an die Umgebung sind Arten nicht unveränderlich, sondern *entwackeln* sich im Laufe der Zeit.
(Die Evolutionstheorie steht damit im Gegensatz zum Kreationismus, der die Unveränderlichkeit der Arten behauptet.)

Christian Bergelt

Genetische Algorithmen

10

Prinzipien der organismischen Evolution IV

- **diskrete genetische Einheiten**
Die Erbinformation wird in diskreten („atomaren“) Einheiten gespeichert; übertragen und geändert (keine kontinuierliche Verschmelzung von Erbmerkmalen), dann sonst kommt es durch Rekombination zum sogenannten *Jenkins nomenclature*, dem völligen Verschwinden jeglicher Verschiedenheit in einer Population.
- **Opportunismus**
Evolutive Prozesse sind *äußerst opportunistisch*: Sie arbeiten ausschließlich mit dem, was vorhanden ist, nicht mit dem, was es einmal gab oder geben könnte. Bessere oder optimale Lösungen werden nicht gefunden, wenn die erforderlichen evolutionen Zwischenstadien Tauglichkeitsnachteile mit sich bringen.
- **evolutionsstrategische Prinzipien**
Optimiert werden nicht nur die Organismen, sondern auch die Mechanismen der Evolution: Vermehrungs- und Sterberaten, Lebensdauern, Anfälligkeit gegenüber Mutationen, Mutationschrittwerten, Evolutionsgeschwindigkeit etc.

Christian Bergelt

Genetische Algorithmen

11

Prinzipien der organismischen Evolution V

- **ökologische Nischen**
Konkurrierende Arten können einander tolerieren, wenn sie unterschiedliche Ökoinischen („Lebensräume“ im weiten Sinne) besetzen, vielleicht sogar selbst schaffen. Nur so ist — trotz Konkurrenz und natürlicher Auslese — die beobachtete Artenvielfalt möglich.
- **Irreversibilität**
Der Gang der Evolution ist *irreversibel* und *unwiederholbar*.
- **Nichtvorhersagbarkeit**
Der Gang der Evolution ist nicht determiniert, nicht programmiert, nicht zielgerichtet und deshalb nicht vorhersagbar.
- **wachsende Komplexität**
Die biologische Evolution hat im allgemeinen zu immer komplexeren Systemen geführt. (Problem: Wie mißt man die Komplexität von Lebewesen?)

Christian Bergelt

Genetische Algorithmen

12

Grundlagen genetischer Algorithmen

Christian Bergelt

Genetische Algorithmen

13

Grundbegriffe und ihre Bedeutung I

Begriff	Biologie	Informatik
Individuum	Lebewesen	Lösungskandidat
Chromosom	DNS-Histon-Protein-Strang	Zeichenkette
	legt den „Bauplan“ bzw. (einen Teil der) Eigenschaften eines Individuums in kodierter Form fest	
	meist mehrere Chromosomen	meist nur ein Chromosom
	je Individuum	je Individuum
Gen	Teilstrick eines Chromosoms	ein Zeichen
	grundlegende Einheit der Vererbung, die eine (Teil-)Eigenschaft eines Individuums festlegt	
Allel	Ausprägung eines Gens	Wert eines Zeichens
(Allelomorph)	je Chromosom gibt es nur eine Ausprägung eines Gens	
Locus	Ort eines Gens	Position eines Zeichens
	in einem Chromosom gibt es an jedem Ort nur genau ein Gen	

Christian Bergelt

Genetische Algorithmen

14

Grundbegriffe und ihre Bedeutung II

Begriff	Biologie	Informatik
Phänotyp	äußeres Erscheinungsbild eines Lebewesens	Umsetzung / Implementierung eines Lösungskandidaten
Genotyp	genetische Konstitution eines Lebewesens	Kodierung eines Lösungskandidaten
Population	Menge von Lebewesen	Familie / Multimenge von Chromosomen
Generation	Population zu einem Zeitpunkt	Population zu einem Zeitpunkt
Reproduktion	Erzeugen von Nachkommen aus einem oder mehreren (weib, Eltern-)Chromosomen	Erzeugen von (Kind-)Chromosomen aus einem oder mehreren (Eltern-)Chromosomen
Fitness	Tauglichkeit / Anzugaftigkeit eines Lebewesens	Güte / Tauglichkeit eines Lösungskandidaten
	bestimmt Überlebens- und Fortpflanzungschancen	

Christian Bergelt

Genetische Algorithmen

15

Elemente eines genetischen Algorithmus I

Ein genetischer Algorithmus besteht aus:

- einer **Kodierungsvorschrift** für die Lösungskandidaten

Wie die Lösungskandidaten kodiert werden, ist problempezifisch; es gibt keine allgemeinen Regeln. Wir werden später jedoch einige Aspekte besprechen, die man bei der Wahl einer Kodierung beachten sollte.

- einer Methode, eine **Anfangspopulation** zu erzeugen

Meist werden einfach zufällige Zeichenketten erzeugt; je nach gewählter Kodierung können aber auch komplexere Verfahren nötig sein.

- einer **Bewertungsfunktion** (Fitnessfunktion) für die Individuen

Die Bewertungsfunktion 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.

- einer **Auswahlmethode** auf der Grundlage der Fitnessfunktion

Die Auswahlmethode bestimmt, welche Individuen zur Erzeugung von Nachkommen herangezogen werden oder auch unverändert in die nächste Generation gelangen.

Christian Bergelt

Genetische Algorithmen

16

Elemente eines genetischen Algorithmus II

Ein genetischer Algorithmus besteht weiter aus:

- **genetischen Operatoren**, die die Lösungskandidaten ändern
 - *Mutation* — zufällige Veränderung einzelner Gene
 - *Crossover* — Rekombination von Chromosomen
(eigentlich "crossing over", nach einem Vorgang in der Meiose (Phase der Zellteilung), bei dem Chromosomen zerfallen und überkreuzt wieder zusammengefügt werden)
- Werten für verschiedene **Parameter**
(z.B. Populationsgröße, Mutationswahrscheinlichkeit etc.)
- einem **Abbruchkriterium**, z.B.
 - eine festgelegte Anzahl von Generationen wurde berechnet
 - eine festgelegte Anzahl von Generationen lang gab es keine Verbesserung
 - eine vorgegebene Mindestfitness wurde erreicht

Christiane Bergelt

Genetische Algorithmen

17

Grundstruktur eines genetischen Algorithmus

procedure evolution-program;

begin

- $t \leftarrow 0$; (* initialisiere den Generationenzähler *)
- initialize pop(t); (* erzeuge die Anfangspopulation *)
- evaluate pop(t); (* und bewerte sie (berechne Fitness) *)
- while** not termination criterion **do** (* solange Abbruchkriterium nicht erfüllt *)
 - $t \leftarrow t + 1$;
 - select pop(t) from pop($t - 1$); (* zähle die erzeugte Generation *)
 - alter pop(t); (* wähle Individuen nach Fitness aus *)
 - evaluate pop(t); (* wende genetische Operatoren an *)
 - end (* bewerte die neue Population *)
 - end (* (berechne neue Fitness) *)

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

- Nur die Individuen der Zwischenpopulation können Nachkommen bekommen.

Christiane Bergelt

Genetische Algorithmen

18

Einführendes Beispiel: Das n-Damen-Problem

Christiane Bergelt

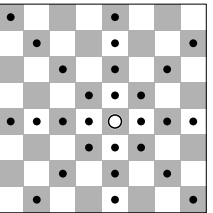
Genetische Algorithmen

19

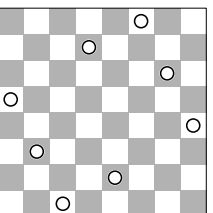
Einführendes Beispiel: Das n-Damen-Problem

Plaziere n Damen auf einem $n \times n$ Schachbrett, so daß in keiner Reihe (Zeile), keiner Linie (Spalte) und keiner Diagonale mehr als eine Dame steht.
Oder: Plaziere die Damen so, daß keine einer anderen im Weg steht.

Zugmöglichkeiten einer Dame



Lösung des 8-Damen-Problems



Christiane Bergelt

Genetische Algorithmen

20

n-Damen-Problem: Backtracking

- Wie aus anderen Vorlesungen wahrscheinlich bekannt, kann das n -Damen-Problem leicht mit Hilfe des Backtracking gelöst werden:
- Die Damen werden zeilenweise, von unten nach oben, platziert. (oder auch spaltenweise, von links nach rechts, o.ä.)
- Jede Zeile wird wie folgt bearbeitet:
 - In einer Zeile wird eine Dame der Reihe nach, von links nach rechts, auf die Felder der Zeile gesetzt.
 - Für jede Platzierung der Dame wird geprüft, ob es zu einer Kollision mit Damen in tieferliegenden Zeilen kommt.
 - Ist dies nicht der Fall, wird rekursiv die nächste Zeile bearbeitet.
 - Anschließend wird die Dame ein Feld weiter nach rechts gerückt.
- Kann eine Dame in der obersten Zeile des Brettes platziert werden, ohne daß es zu einer Kollision kommt, wird die gefundene Lösung ausgegeben.

Christina Bergelt

Genetische Algorithmen

21

n-Damen-Problem: Backtracking

```
int search (int y)
{
    /* --- depth first search */
    int x, i, d;
    int sol = 0;

    if (y >= size) {
        show(); return 1; }
    for (x = 0; x < size; x++) { /* traverse fields of the current row */
        for (i = y; --i >= 0; ) { /* traverse the preceding rows */
            d = abs(apos[i] - x);
            if ((d == 0) || (d == y - i)) break;
        }
        if (i >= 0) continue;
        apos[y] = x;
        sol += search(y+1);
    }
    return sol;
} /* search() */
```

```
/* if a solution has been found, */
/* show it and abort the function */
/* traverse fields of the current row */
/* traverse the preceding rows */
/* and check for collisions */
/* if there is a colliding queen, */
/* skip the current field */
/* otherwise place the queen */
/* and search recursively */
/* return the number of */
/* solutions found */
```

Christina Bergelt

Genetische Algorithmen

22

n-Damen-Problem: Direkte Berechnung

- Ist man nur an *einer* Lösung (einer Platzierung der Damen) interessiert, so kann man die Positionen für alle $n > 3$ wie folgt berechnen:
- Falls n ungerade ist, setze eine Dame auf $(n - 1, n - 1)$ und verringere n um 1
- Falls n mod 6 $\neq 2$: Setze die Damen
in den Zeilen $y = 0, \dots, \frac{n}{2} - 1$ in die Spalten $x = 2y + 1$,
in den Zeilen $y = \frac{n}{2}, \dots, n - 1$ in die Spalten $x = 2y - n$.
- Falls n mod 6 = 2: Setze die Damen
in den Zeilen $y = 0, \dots, \frac{n}{2} - 1$ in die Spalten $x = (2y + \frac{n}{2})$ mod n ,
in den Zeilen $y = \frac{n}{2}, \dots, n - 1$ in die Spalten $x = (2y - \frac{n}{2} + 2)$ mod n .
- Es ist daher eigentlich nicht zweckmäßig, einen genetischen Algorithmus zu verwenden, um das n -Damen-Problem zu lösen. Allerdings lassen sich an diesem Problem einige Aspekte genetischer Algorithmen gut verdeutlichen.

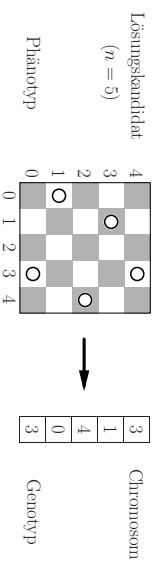
Christina Bergelt

Genetische Algorithmen

23

Genetischer Algorithmus: Kodierung

- Darstellung eines Lösungskandidaten durch ein Chromosom mit n Genen.
- Jedes Gen beschreibt eine Zeile des Schachbrettes und hat n mögliche Allele. Die Genausprägung gibt die Position der Dame in der zugehörigen Zeile an.



- Man beachte, daß durch die Kodierung bereits Lösungskandidaten mit mehr als einer Dame je Zeile ausgeschlossen werden → kleinerer Suchraum.

Christina Bergelt

Genetische Algorithmen

24

Genetischer Algorithmus: Datentypen

- Datentyp für ein Chromosom, der auch die Fitnß speichert.
- Datentyp für eine Population mit einem Puffer für die „Zwischenpopulation“ und einem Member für das beste Individuum.

```
typedef struct {
    int fitness;
    int cnt;
    int genes[1];
} IND;

typedef struct {
    int size;
    IND **buf;
    IND *best;
} POP;

/* --- an individual --- */
/* fitness (number of collisions) */
/* number of genes (number of rows) */
/* genes (queen positions in rows) */
/* (individual) */

/* --- a population --- */
/* number of individuals */
/* vector of individuals */
/* buffer for individuals */
/* best individual */
/* (population) */
```

Christian Borgelt

Genetische Algorithmen

25

Genetischer Algorithmus: Hauptschleife

Die Hauptschleife zeigt die Grundform eines genetischen Algorithmus:

```
pop_init(pop);
while (pop_eval(pop) < 0)
    && (--genct >= 0) {
    pop_select(pop, tmsize, elitist);
    pop_cross (pop, frac);
    pop_mutate(pop, prob);
}

/* initialize the population */
/* while no solution found and */
/* not all generations computed */
/* select individuals, */
/* do crossover, and */
/* mutate individuals */
```

Parameter:

- genct maximale Anzahl (noch) zu berechnender Generationen
- tmsize Größe des Turniers für die Individuenauswahl
- elitist zeigt an, ob das beste Individuum immer übernommen werden soll
- frac Anteil der Individuen, die Crossover unterworfen werden
- prob Mutationswahrscheinlichkeit

Christian Borgelt

Genetische Algorithmen

26

Genetischer Algorithmus: Initialisierung

- Es werden zufällige Folgen von n Zahlen aus $\{0, 1, \dots, n - 1\}$ erzeugt.

```
void ind_init (IND *ind)
{
    int i;

    for (i = ind->n; --i >= 0; ) /* initialize the genes randomly */
        ind->genes[i] = (int)(ind->n *rand());
    ind->fitness = 1;
} /* ind_init() */

void pop_init (POP *pop)
{
    int i;

    for (i = pop->size; --i >= 0; )
        ind_init(pop->inds[i]);
} /* pop_init() */

/* --- initialize a population */
/* loop variable */
```

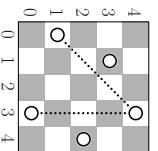
Christian Borgelt

Genetische Algorithmen

27

Genetischer Algorithmus: Bewertung

- Fitnß: negierte Zahl der Spalten und Diagonalen mit mehr als einer Dame. (Negierte Zahl, damit wir eine zu maximierende Fitnß erhalten.)



2 Kollisionen \rightarrow Fitnß = -2

- Bei mehr als zwei Damen in einer Spalte/Diagonale wird jedes Paar gezählt (einfacher zu implementieren).
- Aus dieser Fitnefunktion ergibt sich unmittelbar das Abbruchkriterium: Eine Lösung hat die (maximal mögliche) Fitnß 0.
- Zusätzlich: Maximale Generationenzahl, um Terminierung zu garantieren.

Christian Borgelt

Genetische Algorithmen

28

Genetischer Algorithmus: Bewertung

- Zähle Kollisionen durch Berechnungen auf den Chromosomen.

```
int ind_eval (IND *ind)
{
    int i, k;
    int d;
    int n;

    if (ind->fitness <= 0)
        return ind->fitness;
    for (n = 0, i = ind->n; --i > 0; ) {
        for (k = i; --k > 0; ) { /* traverse all pairs of queens */
            d = abs(ind->genes[i] - ind->genes[k]);
            if ((d == 0) || (d == i-k)) n++;
        }
        /* count the number of pairs of queens */
        /* in the same column or diagonal */
        return ind->fitness = -n;
    } /* ind_eval() */
}
```

Christian Borgelt

Genetische Algorithmen

20

Genetischer Algorithmus: Bewertung

- Es wird die Fitness aller Individuen der Population berechnet.
- Gleichzeitig wird das beste Individuum bestimmt.
- Hat das beste Individuum die Fitness 0, so ist eine Lösung gefunden.

```
int pop_eval (POP *pop)
{
    int i;
    IND *best;

    ind_eval(best = pop->inds[0]);
    for (i = pop->size; --i > 0; )
        if (ind_eval(pop->inds[i]) >= best->fitness)
            best = pop->inds[i];
    pop->best = best;
    /* note the best individual */
    return best->fitness;
} /* pop_eval() */
```

Christian Borgelt

Genetische Algorithmen

30

Genetischer Algorithmus: Auswahl von Individuen

Turnierauswahl:

- Es werden **tmsize** zufällig bestimmte Individuen betrachtet.
- Das beste dieser Individuen „gewinnt“ das Turnier und wird ausgewählt.
- Je höher die Fitness ist, um so größer ist die Chance, ausgewählt zu werden.

```
IND* pop_tmsel (POP *pop, int tmsize)
{
    IND *ind, *best;
    /* --- tournament selection */
    /* competing/best individual */

    best = pop->inds[(int)(pop->size *drand())];
    while (--tmsize > 0) {
        ind = pop->inds[(int)(pop->size *drand())];
        if (ind->fitness > best->fitness) best = ind;
    }
    return best;
} /* pop_tmsel() */
```

Christian Borgelt

Genetische Algorithmen

31

Genetischer Algorithmus: Auswahl von Individuen

- Die Individuen, aus der die Individuen der nächsten Generation erzeugt werden, werden durch Turnierauswahl bestimmt.
- Eventuell wird das beste Individuum übernommen (und nicht verändert).

```
void pop_select (POP *pop, int tmsize, int elitist)
{
    int i;
    IND **p;

    i = pop->size;
    if (elitist)
        ind_copy(pop->buf[--i], pop->best);
    while (--i >= 0)
        ind_copy(pop->buf[i], pop_tmsel(pop, tmsize));
    p = pop->inds; pop->inds = pop->buf;
    pop->buf = p;
    pop->best = NULL;
    /* set selected individuals */
    /* best individual is not known yet */
} /* pop_select() */
```

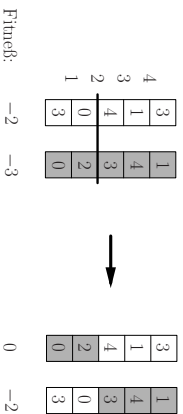
Christian Borgelt

Genetische Algorithmen

32

Genetischer Algorithmus: Crossover

- Austausch eines Chromosomenstücks (oder auch einer in anderer Weise ausgewählter Teilmenge der Gene) zwischen zwei Individuen.
- Hier: sogenanntes **Ein-Punkt-Crossover**
 - Wähle zufällig eine Trennstelle zwischen zwei Genen.
 - Tausche die Gensequenzen auf der einen Seite dieser Trennstelle aus.
 - Beispiel: Wähle Trennstelle 2



Fitness: -2 -3

0 -2

Christian Bergelt

Genetische Algorithmen

35

Genetischer Algorithmus: Crossover

- Austausch eines Chromosomenstücks zwischen zwei Individuen.

```
void ind_cross (IND *ind1, IND *ind2)
{
    /* --- crossover of two chromosomes */
    int i;
    /* loop variable */
    int k;
    /* gene index of crossover point */
    int t;
    /* exchange buffer */

    k = (int)(drand() *(ind1->n-1) +1); /* choose a crossover point */
    if (k > (ind1->n >> 1)) { i = ind1->n; }
    else { i = k; k = 0; }
    while (--i >= k) {
        /* traverse smaller section */
        t = ind1->genes[i];
        ind1->genes[i] = ind2->genes[i];
        ind2->genes[i] = t;
        /* exchange genes */
    }
    /* of the chromosomes */
    ind1->fitness = 1;
    /* invalidate the fitness */
    ind2->fitness = 1;
    /* of the changed individuals */
} /* ind_cross() */
```

Christian Bergelt

Genetische Algorithmen

36

Genetischer Algorithmus: Crossover

- Ein gewisser Anteil der Individuen wird Crossover unterworfen.
- Beide Crossoverprodukte werden in die neue Population aufgenommen, die „Elternindividuen“ gehen verloren.
- Das beste Individuum (falls übernommen) wird keinem Crossover unterzogen.

```
void pop_cross (POP *pop, double frac)
{
    /* --- crossover in a population */
    int i, k;
    /* loop variables */

    k = (int)((pop->size -1) *frac) & ~1;
    for (i = 0; i < k; i += 2) /* crossover of pairs of individuals */
        ind_cross(pop->inds [i], pop->inds[i+1]);
} /* pop_cross() */
```

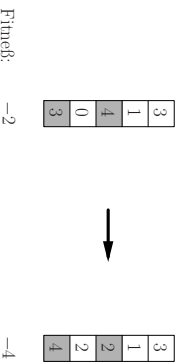
Christian Bergelt

Genetische Algorithmen

37

Genetischer Algorithmus: Mutation

- Zufällig gewählte Gene werden zufällig ersetzt (Allele werden geändert).
- Wie viele Gene ersetzt werden, kann auch zufällig gewählt werden. (Die Zahl ersetzter Gene sollte allerdings klein sein.)



Fitness: -2

-4

- Die meisten Mutationen sind schädlich (verschlechtern die Fitness).
- Anfangs nicht vorhandene Allele können (nur) durch Mutationen entstehen.

Christian Bergelt

Genetische Algorithmen

38

Genetischer Algorithmus: Mutation

- Es wird entschieden, ob eine (weitere) Mutation durchgeführt werden soll.
- Das beste Individuum (falls übernommen) wird nicht mutiert.

```
void ind_mutate (IND *ind, double prob)
{
    /* --- mutate an individual */
    if (drand() >= prob) return; /* det. whether to change individual */
    do ind->genes[ind->n * drand()] = (int)(ind->n * drand());
    while (drand() < prob); /* randomly change random genes */
    ind->fitness = 1; /* fitness is no longer known */
} /* ind_mutate() */

void pop_mutate (Pop *pop, double prob)
{
    /* --- mutate a population */
    int i; /* loop variable */
    for (i = pop->size - 1; --i >= 0; )
        ind_mutate(pop->inds[i], prob);
} /* pop_mutate() */
```

Christian Borgelt

Genetische Algorithmen

37

n-Damen-Problem: Programme

- Die besprochenen Verfahren zur Lösung des n -Damen-Problems,
 - Backtracking,
 - direkte Berechnung,
 - genetischer Algorithmus,stehen auf der Vorlesungsseite als Kommandozeilenprogramme zur Verfügung. (C-Programme **queens.c** und **qga.c**)
- Ruft man diese Programme ohne Parameter auf, erhält man eine Liste von Programmoptionen.
- Man beachte, daß mit Hilfe des genetischen Algorithmus nicht immer eine Lösung gefunden wird (ausgegebenen Lösungskandidat hat eine Fitness < 0).
- Die Eigenschaften der genannten Verfahren werden anhand dieser Programme in einigen Übungsaufgaben noch genauer betrachtet.

Christian Borgelt

Genetische Algorithmen

38

Verwandte Optimierungsverfahren

Christian Borgelt

Genetische Algorithmen

39

Verwandte Optimierungsverfahren

- **Allgemeine Problemstellung:**
Gegeben sei eine Funktion $f : S \rightarrow \mathbb{R}$ (S : Suchraum).
Finde ein Element $s \in S$, das f optimiert (maximiert oder minimiert).
- o.B.d.A.: Finde ein Element $s \in S$, das f maximiert.
(Ist f zu minimieren, kann man stattdessen $f' \equiv -f$ betrachten.)
- **Voraussetzung:**
Für ähnliche Elemente $s_1, s_2 \in S$ unterscheiden sich die Funktionswerte $f'(s_1)$ und $f'(s_2)$ nicht zu sehr (keine großen Sprünge in den Funktionswerten).
- **Verfahren:**
 - Gradientenverfahren
 - Zufallsstufstieg
 - Simuliertes Ausgählen
 - Akzeptieren mit Schwellenwert
 - Simultur-Algorithmus

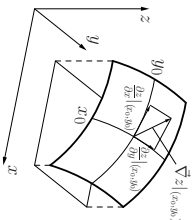
Christian Borgelt

Genetische Algorithmen

40

Gradientenverfahren

- **Voraussetzung:** $S \subseteq \mathbb{R}^n$, $f : S \rightarrow \mathbb{R}$ ist differenzierbar
- **Gradient:** Differentialoperation, die ein Vektorfeld erzeugt. Liefert Vektor in Richtung der stärksten Steigung einer Funktion.



- Illustration des Gradienten einer Funktion $z = f(x, y)$ am Punkt (x_0, y_0) . Es ist $\vec{\nabla} z |_{(x_0, y_0)} = \left(\frac{\partial z}{\partial x} |_{(x_0, y_0)}, \frac{\partial z}{\partial y} |_{(x_0, y_0)} \right)$.

Christian Bergelt

Gradientenverfahren

14

Gradientenverfahren

Idee: Mache, ausgehend von einem zufälligen Startpunkt, kleine Schritte im Suchraum, jeweils in Richtung des stärksten Anstiegs der Funktion, bis ein (lokales) Maximum erreicht ist.

1. Wähle einen (zufälligen) Startpunkt $\vec{x}^{(0)} = (x_1^{(0)}, \dots, x_n^{(0)})$
2. Bestimme den Gradienten am aktuellen Punkt $\vec{x}^{(i)}$:

$$\nabla_{\vec{x}} f(\vec{x}^{(i)}) = \left(\frac{\partial}{\partial x_1} f(\vec{x}^{(i)}), \dots, \frac{\partial}{\partial x_n} f(\vec{x}^{(i)}) \right)$$
3. Gehe ein kleines Stück in Richtung des Gradienten:

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} + \eta \nabla_{\vec{x}} f(\vec{x}^{(i)}).$$
 η ist ein Schrittweitenparameter („Lernrate“ in neuronalen Netzen)
4. Wiederhole Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist. (Vorgegebene Anzahl Schritte ausgeführt, aktueller Gradient sehr klein)

Christian Bergelt

Gradientenverfahren

14

Gradientenverfahren: Probleme

- **Wahl des Schrittweitenparameters**
 - Bei einem zu kleinen Wert kann es sehr lange dauern, bis das Maximum erreicht ist (Schritte zu klein).
 - Bei einem zu großen Wert kann es zu Oszillationen (Hin- und Herspringen im Suchraum) kommen (Schritte zu groß).
 - Lösungsmöglichkeiten: Momentum, adaptiver Schrittweitenparameter (Details: siehe Vorlesung „Neuronale Netze“)
- **Hängenbleiben in lokalen Maxima**
 - Da nur lokale Steigungsinformation genutzt wird, wird eventuell nur ein lokales Maximum erreicht.
 - Dieses Problem kann *nicht* prinzipiell behoben werden.
 - Chancenverbesserung für Finden des globalen Optimums: Mehrfaches Ausführen des Gradientenaufstieg von verschiedenen Startwerten aus.

Christian Bergelt

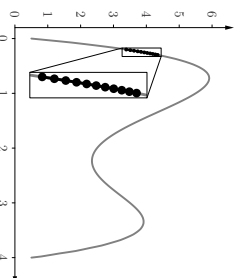
Gradientenverfahren

14

Gradientenverfahren: Beispiele

Beispielfunktion: $f(x) = -\frac{5}{2}x^4 + 7x^3 - \frac{115}{6}x^2 + 18x + \frac{1}{2}$

i	x_i	$f(x_i)$	Δx_i
0	0.200	3.388	11.147
1	0.211	3.510	10.811
2	0.222	3.626	10.490
3	0.232	3.734	10.182
4	0.243	3.836	9.888
5	0.253	3.932	9.606
6	0.262	4.023	9.335
7	0.271	4.109	9.075
8	0.281	4.191	8.825
9	0.289	4.267	8.585
10	0.298	4.340	0.009



Gradientenaufstieg mit Startwert 0.2 und Schrittweitenparameter 0.001.

Christian Bergelt

Gradientenverfahren

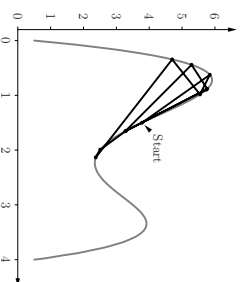
14

Gradientenverfahren: Beispiele

Beispielfunktion:

$$f(x) = -\frac{5}{6}x^4 + 7x^3 - \frac{115}{6}x^2 + 18x + \frac{1}{2}$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	1,500	3,781	-3,500	-0,875
1	0,625	5,845	1,431	0,358
2	0,983	5,545	-2,554	-0,639
3	0,344	4,689	7,157	1,789
4	2,134	2,373	-0,567	-0,142
5	1,992	2,511	-1,380	-0,345
6	1,647	3,297	-3,063	-0,766
7	0,881	5,766	-1,753	-0,438
8	0,443	5,289	4,851	1,213
9	1,656	3,269	-3,029	-0,757
10	0,898	5,734		



Gradientenaufstieg mit Startwert 1,5 und Schrittweiteparameter 0,25.

Christina Bergelt

Genetische Algorithmen

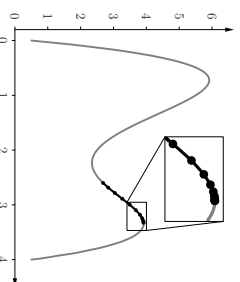
15

Gradientenverfahren: Beispiele

Beispielfunktion:

$$f(x) = -\frac{5}{6}x^4 + 7x^3 - \frac{115}{6}x^2 + 18x + \frac{1}{2}$$

i	x_i	$f(x_i)$	$f'(x_i)$	Δx_i
0	2,600	2,684	1,707	0,085
1	2,685	2,840	1,947	0,097
2	2,783	3,039	2,116	0,106
3	2,888	3,267	2,153	0,108
4	2,996	3,492	2,009	0,100
5	3,097	3,680	1,688	0,084
6	3,181	3,805	1,263	0,063
7	3,244	3,872	0,845	0,042
8	3,286	3,901	0,515	0,026
9	3,312	3,911	0,293	0,015
10	3,327	3,915		



Gradientenaufstieg mit Startwert 2,6 und Schrittweiteparameter 0,05.

Christina Bergelt

Genetische Algorithmen

16

Zufallsaufstieg

• **Idee:** Wenn die Funktion f nicht differenzierbar ist, kann man versuchen, eine Richtung, in der die Funktion f ansteigt, durch Auswerten zufälliger Punkte in der Umgebung des aktuellen Punktes zu bestimmen.

1. Wähle einen (zufälligen) Startpunkt $s_0 \in S$.
 2. Wähle einen Punkt $s' \in S$, in der Nähe^{er} des aktuellen Punktes s_i . (z.B. durch zufällige, aber nicht zu große Veränderung von s_i)
 3. Setze

$$s_{i+1} = \begin{cases} s', & \text{falls } f(s') \geq f(s_i), \\ s_i, & \text{sonst.} \end{cases}$$
 4. Wiederhole Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist.
- **Problem:** Hängenbleiben in lokalen Maxima.
- Alle folgenden Verfahren versuchen, dieses Problem zu verringern.

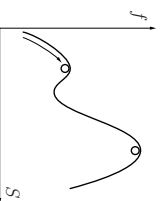
Christina Bergelt

Genetische Algorithmen

17

Simuliertes Ausgüthen

- Kann als Erweiterung des Zufalls- und Gradientenaufstiegs gesehen werden, die ein Hängenbleiben vermeidet.
- **Idee:** Übergänge von niedrigeren zu höheren (lokalen) Maxima sollen wahrscheinlicher sein als umgekehrt.



Prinzip des simulierten Ausgüthens:

- Zufällige Varianten der aktuellen Lösung werden erzeugt.
- Bessere Lösungen werden immer übernommen.
- Schlechtere Lösungen werden mit einer bestimmten Wahrscheinlichkeit übernommen, die abhängt von
 - der Qualitätsdifferenz der aktuellen und der neuen Lösung und
 - einem Temperaturparameter, der im Laufe der Zeit verringert wird.

Christina Bergelt

Genetische Algorithmen

18

Simuliertes Ausgüthen

- **Motivation:** (Minimierung statt Maximierung)
Physikalische Minimierung der Energie (genauer: der Atomgitterenergie), wenn ein erhitztes Stück Metall langsam abgekühlt wird.
Dieser Prozess wird **Ausgüthen** (engl.: annealing) genannt. Er dient dazu, ein Metall weicher zu machen, indem innere Spannungen und Instabilitäten aufgehoben werden, um es dann leichter bearbeiten zu können.
- **Alternative Motivation:** (ebenfalls Minimierung)
Eine Kugel rollt auf einer unregelmäßig gewellten Oberfläche.
Die zu minimierende Funktion ist die potentielle Energie der Kugel.
Am Anfang hat die Kugel eine gewisse kinetische Energie, die es ihr erlaubt, Ausbege zu überwinden. Durch Reibung sinkt die Energie der Kugel, so daß sie schließlich in einem Tal zur Ruhe kommt.
- **Achtung:** Es gibt keine Garantie, daß das globale Optimum gefunden wird.

Christian Bergelt

Genetische Algorithmen

40

Simuliertes Ausgüthen

1. Wähle einen (zufälligen) Startpunkt $s_0 \in S$.
 2. Wähle einen Punkt $s' \in S$, in der Nähe“ des aktuellen Punktes s_i .
(z.B. durch zufällige, aber nicht zu große Veränderung von s_i)
 3. Setze
$$s_{i+1} = \begin{cases} s', & \text{falls } f(s') \geq f(s_i), \\ s' & \text{mit Wahrscheinlichkeit } p = e^{-\frac{\Delta f}{kT}}, \\ s_i & \text{mit Wahrscheinlichkeit } 1 - p, \end{cases}$$
sonst.
- $\Delta f = f(s_i) - f(s')$ Qualitätsverringern der Lösung
(Schätzung der) Spannwerte der Funktionswerte
 $k = \Delta f_{\max}$ Temperaturparameter; wird im Laufe der Zeit gesenkt
 T
4. Wiederhole Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist.
- Für kleine T geht das Verfahren nahezu in einen Zufallsanstrich über.

Christian Bergelt

Genetische Algorithmen

50

Akzeptieren mit Schwellenwert

- **Idee:** Ähnlich wie beim simulierten Ausgüthen werden auch schlechtere Lösungen akzeptiert, allerdings mit einer oberen Schranke für die Verschlechterung.
1. Wähle einen (zufälligen) Startpunkt $s_0 \in S$.
 2. Wähle einen Punkt $s' \in S$, in der Nähe“ des aktuellen Punktes s_i .
(z.B. durch zufällige, aber nicht zu große Veränderung von s_i)
 3. Setze
$$s_{i+1} = \begin{cases} s', & \text{falls } f(s') \geq f(s_i) - \theta, \\ s_i, & \text{sonst.} \end{cases}$$
- θ Schwellenwert für das Akzeptieren schlechterer Lösungen;
wird im Laufe der Zeit (langsam) gesenkt.
($\theta = 0 \rightarrow$ normaler Zufallsanstrich)
4. Wiederhole Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist.

Christian Bergelt

Genetische Algorithmen

51

Sintflut-Algorithmus

- **Idee:** Ähnlich wie beim simulierten Ausgüthen werden auch schlechtere Lösungen akzeptiert, allerdings mit einer unteren Schranke.
1. Wähle einen (zufälligen) Startpunkt $s_0 \in S$.
 2. Wähle einen Punkt $s' \in S$, in der Nähe“ des aktuellen Punktes s_i .
(z.B. durch zufällige, aber nicht zu große Veränderung von s_i)
 3. Setze
$$s_{i+1} = \begin{cases} s', & \text{falls } f(s') \geq \theta, \\ s_i, & \text{sonst.} \end{cases}$$
- θ Untere Schranke für die Lösungsgüte;
wird im Laufe der Zeit (langsam) erhöht.
(„Fluten“ des Funktionsgebietes; θ entspricht dem „Wasserstand“)
4. Wiederhole Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist.

Christian Bergelt

Genetische Algorithmen

52

Beispiel: Problem des Handlungsreisenden

- **Gegeben:**
 - Eine Menge von n Städten (als Punkte in einer Ebene)
 - Abstände/Kosten der Wege zwischen den Städten
- **Gesucht:**
 - Rundreise minimaler Länge/Kosten durch alle n Städte, auf der keine Stadt mehr als einmal besucht wird
- **Mathematisch:** Suche eines Hamiltonkreises (enthält jeden Knoten genau einmal) mit minimalem Gewicht in einem Graphen mit gewichteten Kanten.
- **Bekannt:** Dieses Problem ist NP-vollständig, d.h., man kennt keinen Algorithmus, der dieses Problem in polynomialer Zeit löst.
- **Daher:** Für großes n ist in annehmbarer Zeit nur eine Näherungslösung berechenbar (die beste Lösung kann gefunden werden, dies ist aber nicht garantiert).
- **Hier:** Betrachte Ansatz mit simuliertem Ausglühen und Zufallsaufstieg.

Christian Borgelt

Genetische Algorithmen

55

Beispiel: Problem des Handlungsreisenden

1. Bringe die Städte in eine zufällige Reihenfolge (zufällige Rundreise).
 2. Wähle zufällig zweimal zwei Städte, die in der aktuellen Rundreise aufeinander folgen (alle vier Städte verschieden). Tausche die Rundreise zwischen den Städten jedes Paares auf und drehe den dazwischenliegenden Teil um.
 3. Wenn die so entstehende Rundreise besser (kürzer, billiger) ist als die alte, ersetze die alte Rundreise auf jeden Fall durch die neue, sonst ersetze die alte Rundreise nur mit Wahrscheinlichkeit $p = e^{-\frac{\Delta Q}{T}}$.
- ΔQ Qualitätsunterschied zwischen alter und neuer Rundreise
- k Spannweite der Rundreisesqualitäten
- (muß ggf. geschätzt werden, z.B. $k_T = \frac{t+1}{t} \max_{j=1}^t \Delta Q_j$, wobei ΔQ_j der Qualitätsunterschied im j -ten Schritt und t der aktuelle Schritt ist.)
- T Temperaturparameter, der im Laufe der Zeit abnimmt, z.B. $T = \frac{1}{t}$
4. Wiederhole die Schritte 2 und 3, bis ein Abbruchkriterium erfüllt ist.

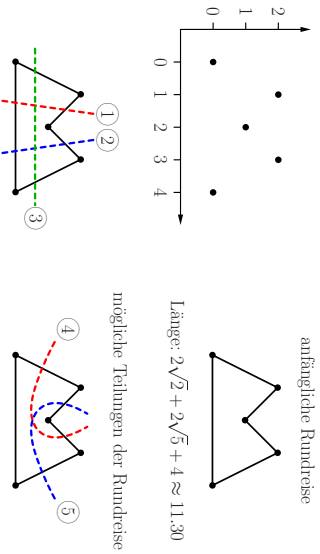
Christian Borgelt

Genetische Algorithmen

56

Beispiel: Problem des Handlungsreisenden

- Reiner Zufallsaufstieg kann in einem lokalen Minimum hängenbleiben. Dazu ein einfaches Beispiel mit 5 Städten:



Christian Borgelt

Genetische Algorithmen

56

Beispiel: Problem des Handlungsreisenden

- ① Länge: $\sqrt{2} + 3\sqrt{5} + \sqrt{13} \approx 11.73$
- ② Länge: $\sqrt{2} + 2\sqrt{13} + 4 \approx 14.04$
- ③ Länge: $\sqrt{2} + 2\sqrt{5} + 2 + 4 \approx 11.89$
- ④ Länge: $4\sqrt{5} + 2 \approx 10.94$
- ⑤ Beste Rundreise:
(globales Optimum)

Christian Borgelt

Genetische Algorithmen

56

Beispiel: Problem des Handlungsreisenden

- Alle Modifikationen der Aufgangsroute führen zu Rundreisen, die schlechter sind. Das globale Optimum kann daher, ausgehend von dieser Rundreise, mit einem reinen Zufallsaufstieg nicht gefunden werden.
 - Beim simulierten Ausglichen werden dagegen mitunter auch schlechtere Lösungen akzeptiert, so daß man zum globalen Optimum gelangen kann. (*aber*: Es gibt keine Garantie, daß dies geschieht!)
 - **Beachte**: Es kann von den erhaltbaren Operationen abhängen, ob die Suche in einem lokalen Optimum hängenbleiben kann:
 - Läßt man als weitere Operation zu, daß die Position einer Stadt in der Rundreise geändert wird (Entfernen von der aktuellen Position und Einfügen an einer anderen), so tritt im betrachteten Beispiel kein Hängenbleiben mehr auf.
- Auch für diese Operationenmenge läßt sich jedoch ein Beispiel konstruieren, in dem die Suche in einem lokalen Minimum hängenbleibt.

Christian Borgelt

Genetische Algorithmen

57

Verwandte Optimierungsverfahren: Probleme

- Alle bisher betrachteten Verfahren suchen i.w. **lokal**:
 - Es wird stets nur ein aktueller Lösungskandidat betrachtet.
 - Der aktuelle Lösungskandidat wird nur geringfügig verändert.
- **Nachteile**: Es wird u.U. nur ein kleiner Teil des Suchraums betrachtet.
- **Abhilfe**: Mehrere Läufe des Verfahrens mit verschiedenen Startpunkten.
- **Nachteile**: Keine Informationsübertragung von einem Lauf zum nächsten.
- **Beachte**: Große Veränderungen des Lösungskandidaten, bis hin zu völliger Neuberechnung, sind nicht sinnvoll, da dann zu wenig/keine Information von einem Lösungskandidaten zum nächsten weitergegeben wird.
 - (Man könnte im Extremfall auch einfach eine Menge von Lösungskandidaten zufällig erzeugen und den besten auswählen.)
- → wichtig sind: Zusammenhang der Lösungskandidaten, großräumigere Abdeckung des Suchraums

Christian Borgelt

Genetische Algorithmen

58

Teilenschwarmoptimierung

- (engl.: **Particle Swarm Optimization**) [Kennedy and Eberhart 1995]
- Teilenschwarmoptimierung kann gesehen werden als ein Verfahren, das Elemente der bahnorientierten Suche (z.B. Gradientenverfahren) und populärere beobachteter Suche (z.B. genetische Algorithmen) zusammenbringt.
 - **Ansatz**: Statt nur einem einzelnen aktuellen Lösungskandidaten wird ein „Schwarm“ von m Lösungskandidaten verwendet.
 - **Motivation**: Verhalten von z.B. Fischeschwärmen bei der Futtersuche: Zufälliges Ausschwärmen, aber stets auch Rückkehr zum Schwarm. Informationsaustausch zwischen den Schwarmmitgliedern.
 - **Voraussetzung**: Der Suchraum ist reellwertig, d.h. $S \subseteq \mathbb{R}^n$ und folglich die zu optimierende (o.B.d.A.: zu maximierende) Funktion $f: \mathbb{R}^n \rightarrow \mathbb{R}$.
 - **Vorgehen**: Jeder Lösungskandidat wird als „Teilchen“ aufgefaßt, das einen Ort \vec{x}_i im Suchraum und eine Geschwindigkeit \vec{v}_i^t hat, $i = 1, \dots, m$.

Christian Borgelt

Genetische Algorithmen

59

Teilenschwarmoptimierung

- **Aktualisierungsformeln** für Ort und Geschwindigkeit des i -ten Teilchens:
$$\vec{x}_i(t+1) = \vec{x}_i(t) + \vec{v}_i^t(t)$$
$$\vec{v}_i^t(t+1) = \alpha \cdot \vec{v}_i^t(t) + \beta_1 \cdot (\vec{x}_i^{(local)}(t) - \vec{x}_i(t)) + \beta_2 \cdot (\vec{x}_i^{(global)}(t) - \vec{x}_i(t))$$
- $\vec{x}_i^{(local)}$ ist das **lokale Gedächtnis** des Individuums (Teilchens). Es ist der beste Ort im Suchraum, den ein Individuum des Schwarms bisher besucht hat (bester bisher gefundener Lösungskandidat), d.h.
$$\vec{x}_i^{(local)}(t) = \vec{x}_i^{(local)}(t) \quad \text{mit} \quad j = \text{argmax}_{k=1}^m f(\vec{x}_k^{(local)}(t)).$$
- Parameterwahl: β_1, β_2 zufällig in jedem Schritt, α mit der Zeit abnehmend.

Christian Borgelt

Genetische Algorithmen

60

Formen genetischer Algorithmen

Christian Bergelt

Genetische Algorithmen

61

Grundstruktur eines genetischen Algorithmus

procedure evolution_program;

begin

$t \leftarrow 0$;

initialize pop(t);

evaluate pop(t);

while not termination_criterion **do**

$t \leftarrow t + 1$;

select pop(t) from pop($t - 1$);

alter pop(t);

evaluate pop(t);

end

end

(* initialisiere den Generationenzähler *)

(* erzeuge die Anfangspopulation *)

(* und bewerte sie (berechne Fitness) *)

(* solange Abbruchkriterium nicht erfüllt *)

(* zähle die erzeugte Generation *)

(* wähle Individuen nach Fitness aus *)

(* wende genetische Operatoren an *)

(* bewerte die neue Population *)

(* berechne neue Fitness) *)

- Durch die Auswahl wird eine Art „Zwischenpopulation“ von Individuen mit (im Durchschnitt) hoher Fitness erzeugt.

- Nur die Individuen der Zwischenpopulation können Nachkommen bekommen.

Christian Bergelt

Genetische Algorithmen

62

Genetische Algorithmen: Formen

• Standardform:

- Auswahl einer „Zwischenpopulation“ mit einem Selektionsverfahren
- Anwendung der genetischen Operatoren auf die Zwischenpopulation, wobei alle Individuen mutiert werden können.

- **Modifizierte Form:** (Parameter r , $0 < r < p_{\text{size}}$, und p_c , $0 < p_c < 1$)

- Wähle aus pop(t) zufällig (aber unter Berücksichtigung der Fitness) r Chromosomen (**mit** Zurücklegen).

- Bestimme für jedes dieser r Chromosomen, ob es am Crossover teilnimmt (Wahrscheinlichkeit p_c) oder mutiert wird (Wahrscheinlichkeit $1 - p_c$) und wende die genetischen Operatoren an.

- Wähle aus pop(t) zufällig (aber unter Berücksichtigung der Fitness) $p_{\text{size}} - r$ Chromosomen (**ohne** Zurücklegen).

- Diese $p_{\text{size}} - r$ Chromosomen werden unverändert übernommen.

Christian Bergelt

Genetische Algorithmen

63

Genetische Algorithmen: Formen

Eigenschaften der modifizierten Form:

- Ein Chromosom nimmt **entweder** am Crossover teil **oder** wird mutiert.
- Die unveränderten Chromosomen werden nicht vervielfacht (da sie *ohne* Zurücklegen gewählt werden).

Vorteile:

- Die Population enthält so gut wie keine identischen Individuen. (Prinzip der *Diversität* — Verschiedenheit der Individuen)
- Vorteilhafte Crossoverprodukte haben bessere Chancen, da sie nicht durch eventuelle Mutationen beeinträchtigt werden.

Nachteile:

- Größere Gefahr des „Aussterbens“ guter Lösungsstandarten, da es nur eine Kopie jedes Individuums in einer Population gibt.

Christian Bergelt

Genetische Algorithmen

64

Genetische Algorithmen: Formen

Steady-State genetischer Algorithmus (Parameter r , $1 \leq r \leq \text{popsize}$)

- Zufällige Auswahl von zwei Eltern (unter Berücksichtigung der Fitness) und Anwendung genetischer Operatoren auf diese Eltern.
- Verwerfen von Duplikaten (in der Population bereits vorhandene Individuen).
- Insgesamt werden so r Nachkommen erzeugt, die die r schlechtesten Nachkommen der Population ersetzen ($1 \leq r \leq \text{popsize}$, oft $r = 2$).

Vorteile:

- Duplikate werden explizit vermieden \rightarrow bessere Abdeckung des Suchraums
- kein Aussterben sehr guter Individuen (wenn $q < \text{popsize}$)

Nachteile:

- Größerer Aufwand bei der Individuenauswahl.
- Etwas größere Gefahr des Hängenbleibens in lokalen Optima.

Christian Borgelt

Genetische Algorithmen

65

Elemente genetischer Algorithmen

1. Kodierung der Lösungskandidaten

Christian Borgelt

Genetische Algorithmen

66

Genetische Algorithmen: Kodierung

Im folgenden betrachten wir die Elemente eines genetischen Algorithmus genauer.

Zunächst: **Kodierung der Lösungskandidaten**

- Wie bereits erwähnt, ist die Kodierung problemspezifisch zu wählen.
- Es gibt kein allgemeines „Kochrezept“, um eine (gute) Kodierung zu finden.
- Aber man kann einige Prinzipien angeben, die beachtet werden sollten.

Wünschenswerte Eigenschaften einer Kodierung:

- Ähnliche Phänotypen sollten durch ähnliche Genotypen dargestellt werden.
- Ähnlich kodierte Lösungskandidaten sollten eine ähnliche Fitness haben.
- Der Suchraum (die Menge möglicher Lösungskandidaten) sollte, soweit möglich, unter den verwendeten genetischen Operatoren abgeschlossen sein.

Christian Borgelt

Genetische Algorithmen

67

Genetische Algorithmen: Kodierung

Ähnliche Phänotypen sollten durch ähnliche Genotypen dargestellt werden.

- Mutationen einzelner Gene führen zu ähnlichen Genotypen (einzelne Alleländerungen \rightarrow kleine Änderung des Chromosoms).
- Werden ähnliche Phänotypen nicht durch ähnliche Genotypen dargestellt, können naheliegende Verbesserungen u. U. nicht erzeugt werden. (Es ist dann eine große Änderung des Genotyps erforderlich, um zu einem ähnlichen (und vielleicht besseren) Phänotyp zu gelangen.)

Beispiel zur Verdeutlichung:

- Optimierung einer reellen Funktion $y = f(x_1, \dots, x_n)$.
- Die (reellen) Argumente sollen durch Binärcodes dargestellt werden.
- Problem: einfache Kodierung als Binärzahl führt zu „Hamming-Klippen“.

Christian Borgelt

Genetische Algorithmen

68

Binärkodierung reeller Zahlen

- **Gegeben:** ein reelles Intervall $[a, b]$ und eine Kodierungsgenauigkeit ϵ .
- **Gesucht:** eine Kodierungsvorschrift für Zahlen $x \in [a, b]$ als Binärzahl z , so daß die kodierte Zahl um weniger als ϵ von ihrem tatsächlichen Wert abweicht.
- **Vorgehen:**
 - Teile das Intervall $[a, b]$ in gleich große Abschnitte mit einer Länge $\leq \epsilon$.
 $\rightarrow 2^k$ Abschnitte mit $k = \lceil \log_2 \frac{b-a}{\epsilon} \rceil$ kodiert durch Zahlen $0, \dots, 2^k - 1$.
 - **Kodierung:** $z = \lfloor \frac{x-a}{b-a}(2^k - 1) \rfloor$ (alternativ: $\lfloor \frac{x-a}{b-a}(2^k - 1) + \frac{1}{2} \rfloor$).
 - **Dekodierung:** $x = a + z \cdot \frac{b-a}{2^k - 1}$ dann reicht $k = \lceil \log_2 \frac{b-a}{\epsilon} \rceil$
- **Beispiel:** Intervall $[-1, 2]$, Genauigkeit $\epsilon = 10^{-6}$, $x = 0.637197$.
 - $k = \lceil \log_2 \frac{2-(-1)}{10^{-6}} \rceil = \lceil \log_2 3 \cdot 10^6 \rceil = 22$
 - $z = \lfloor \frac{0.637197-(-1)}{2-(-1)}(2^{22} - 1) \rfloor = 2288966_{10} = 1000101110110101000110_2$

Christian Bergelt

Grundlegende Algorithmen

60

Vermmeidung von Hamming-Klippen: Gray-Kodes

Problem: Benachbarte Zahlen können sehr verschieden kodiert sein, d.h. die Kodierungen haben einen großen Hamming-Abstand (Anzahl verschiedener Bits). Große Hamming-Abstände können durch Mutationen und Crossover nur sehr schwer überwinden werden (sogenannte „Hamming-Klippen“).

Beispiel: Der Bereich der Zahlen von 0 bis 1 werde durch 4-Bit-Zahlen dargestellt, d.h. Abbildung $\frac{x}{5} \rightarrow k$. Dann haben die Kodierungen von $\frac{1}{5}$ (0111) und $\frac{2}{5}$ (1000) den Hamming-Abstand 4, denn jedes Bit ist verschieden.

Lösung: Gray-Kodes — benachbarte Zahlen unterscheiden sich nur in einem Bit.

binär	Gray
0000	0000
0001	0001
0010	0011
0011	0010

binär	Gray
0100	0110
0101	0111
0110	0101
0111	0100

binär	Gray
1000	1100
1001	1101
1010	1111
1011	1110

binär	Gray
1100	1010
1101	1011
1110	1001
1111	1000

Christian Bergelt

Grundlegende Algorithmen

70

Gray-Kodes: Berechnung

- **Gray-Kodes sind nicht eindeutig:** Jeder Kode, in dem sich die Kodierungen benachbarter Zahlen nur in einem Bit unterscheiden, heißt Gray-Kode.
- Gray-Kodes werden meist aus einer Binärzahlkodierung berechnet.
- **Häufigste Form:**
 - **Kodierung:** $g = z \oplus \lfloor \frac{z}{2} \rfloor$ (\oplus : Exklusiv-Oder der Binärdarstellung)
 - **Dekodierung:** $z = \bigoplus_{i=0}^{k-1} \lfloor \frac{z}{2^i} \rfloor$
- **Beispiel:** Intervall $[-1, 2]$, Genauigkeit $\epsilon = 10^{-6}$, $x = 0.637197$.
 - $z = \lfloor \frac{0.637197-(-1)}{2-(-1)}(2^{22} - 1) \rfloor = 2288966_{10} = 1000101110110101000110_2$
 - $g = 1000101110110101000110_2$
 $\oplus 100010111011010100011_2$
 $= 1100111001101111100101_2$

Christian Bergelt

Grundlegende Algorithmen

71

Gray-Kodes: Implementierung

```

unsigned int num2gray (unsigned int x)
{
    /* --- convert number to Gray code */
    return x ^ (x >> 1);
} /* num2gray() */

unsigned int gray2num (unsigned int x)
{
    /* --- convert Gray code to number */
    unsigned int y = x;
    while (x >>= 1) y ^= x;
    return y;
} /* gray2num() */

unsigned int gray2num (unsigned int x)
{
    /* --- convert Gray code to number */
    x ^= x >> 16; x ^= x >> 8;
    x ^= x >> 4; x ^= x >> 2;
    return x ^ (x >> 1);
} /* gray2num() */

```

Christian Bergelt

Grundlegende Algorithmen

72

Genetische Algorithmen: Kodierung

Der Suchraum (Menge der kodierten Lösungskandidaten) sollte, soweit möglich, unter den verwendeten genetischen Operatoren abgeschlossen sein.

- Was als Verlassen des Suchraums gilt, ist i. U. eine Definitionsfrage.
- Allgemein: Der **Suchraum wird verlassen**, wenn
 - das neue Chromosom nicht sinnvoll interpretiert/dekodiert werden kann,
 - der Lösungskandidat bestimmte prinzipielle Anforderungen nicht erfüllt,
 - der Lösungskandidat durch die Fitnessfunktion falsch bewertet wird.
- Problem der **Abstimmung** von Kodierung und genetischen Operatoren:
 - Verwenden kodierungsspezifischer genetischer Operatoren.
 - Einsatz von Mechanismen, die Chromosomen „reparieren“.
 - Einführen eines Strafterms, der die Fitness von Chromosomen außerhalb des Suchraums (deutlich) verringert.

Christina Bergelt

Genetische Algorithmen

77

Genetische Algorithmen: Verlassen des Suchraums

Beispiel zum Verlassen des Suchraums:

- n -Damen-Problem (Platzierung von n -Damen auf einem $n \times n$ -Schachbrett).

- **Kodierung 1:**

Chromosom der Länge n , das die Spaltenpositionen der Damen je Zeile angibt (Allele $0, \dots, n - 1$, siehe einführendes Beispiel).

Operatoren: Ein-Punkt-Crossover, Standardmutation

Es entstehen stets wieder gültige Vektoren von Spaltenpositionen.

→ Suchraum wird nicht verlassen.

- **Kodierung 2:**

Chromosom der Länge n , das die Nummern der Felder (Allele $0, \dots, n^2 - 1$) angibt, auf denen Damen stehen.

Operatoren: Ein-Punkt-Crossover, Standardmutation

Es entstehen Chromosomen, die mehrere Damen auf das gleiche Feld setzen.
→ Suchraum wird verlassen.

Christina Bergelt

Genetische Algorithmen

78

Genetische Algorithmen: Verlassen des Suchraums

Mögliche Lösungsansätze am Beispiel des n -Damen-Problems:

- **Andere Kodierung verwenden:** Da die erste Kodierung das Problem des Verlassens des Suchraums vermeidet, außerdem der Suchraum deutlich kleiner ist, ist sie vorzuziehen. (Dies ist, wenn durchführbar, die beste Variante!)
- **Kodierungsspezifische genetische Operatoren**
 - *Mutation:* Schließe bereits vorhandene Allele bei der zufälligen Wahl aus.
 - *Crossover:* Stelle zunächst die Feldnummern je Chromosom zusammen, die in jeweils anderen Chromosomen nicht vorkommen, und wende auf die so verknüpften Chromosomen das Ein-Punkt-Crossover an.
- **Reparaturmechanismus:** Finde und ersetze doppelt bzw. mehrfach auftretende Feldnummern, so daß alle Feldnummern verschieden werden.
- **Strafterm:** Verringere die Fitness um die Anzahl der Doppel-/Mehrfachbelegungen von Feldern, ggf. multipliziert mit einem Gewichtungsfaktor.

Christina Bergelt

Genetische Algorithmen

79

Genetische Algorithmen: Verlassen des Suchraums

Weiteres Beispiel: **Problem des Handlungsreisenden**

- Eine Rundreise wird durch eine Permutation der Städte dargestellt. (Stadt an k -ter Position wird im k -ten Schritt besucht.)
- Ein-Punkt-Crossover kann den Raum der Permutationen verlassen:

3	5	2	8	1	7	6	4
1	2	3	4	5	6	7	8

3	5	2	4	5	6	7	8
1	2	3	8	1	7	6	4

- **Kodierungsspezifische genetische Operatoren:**
 - *Mutation:* z. B. Zweiertausch, Verschieben eines Teilstücks, Inversion
 - *Crossover:* Kantentrennkombination (wird später besprochen)
- **Reparaturmechanismus:** Entferne doppelt auftretende Städte und hänge die fehlenden Städte am Ende an:

3	5	2	4	X	6	7	8	1
---	---	---	---	---	---	---	---	---
- **Strafterm:** Verringern der Fitness um eine Konstante für jede fehlende Stadt.

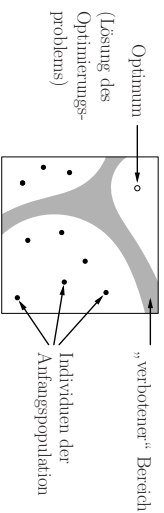
Christina Bergelt

Genetische Algorithmen

80

Genetische Algorithmen: Verlassen des Suchraums

- Bei **nicht zusammenhängenden Suchräumen** können **Reparaturmechanismen** die Suche erschweren, da Lösungskandidaten in den „verbotenen“ Bereichen sofort wieder in den erlaubten Bereich zurückgeführt werden.



- In diesen Fällen ist es angebrachter, einen **Straferraum** einzuführen, durch den Lösungskandidaten in „verbotenen“ Bereich zwar bestraft, aber nicht entfernt werden. Dieser Straferraum sollte im Laufe der Zeit wachsen, um Lösungskandidaten in den „verbotenen“ Bereichen in späteren Generationen zu unterdrücken.

Christian Bergelt

Genetische Algorithmen

81

Elemente genetischer Algorithmen

2. Fitneßfunktion und Selektionsverfahren

Christian Bergelt

Genetische Algorithmen

82

Genetische Algorithmen: Selektion

- **Prinzip der Selektion:** Bessere Individuen (bessere Fitneß) sollen größere Chancen haben, Nachkommen zu haben (differenzielle Reproduktion).

Die Stärke der Bevorzugung guter Individuen heißt **Selektionsdruck**

Bei der Wahl des Selektionsdrucks gibt es einen Gegensatz von

- **Durchforstung des Suchraums (exploration):**
Die Individuen sollten möglichst breit über den Suchraum gestreut sein, damit die Chancen, daß das globale Optimum gefunden wird, möglichst groß sind.
→ geringer Selektionsdruck wünschenswerter
- **Ausbeutung guter Individuen (exploitation):**
Es sollte das (u. U. lokale) Optimum in der Nähe guter Individuen angestrebt werden (Konvergenz zum Optimum).
→ hoher Selektionsdruck wünschenswerter

Christian Bergelt

Genetische Algorithmen

83

Genetische Algorithmen: Selektion

Wahl des Selektionsdrucks:

- **Beste Strategie:** Zeitabhängiger Selektionsdruck
 - geringerer Selektionsdruck in früheren Generationen
 - höherer Selektionsdruck in späteren Generationen→ zuerst gute Durchforstung des Suchraums, dann Ausbeutung der erfolgversprechendsten Region
- Der Selektionsdruck wird über eine Skalierung der Fitneßfunktion oder über Parameter des Selektionsverfahrens gesteuert.
- Wichtige **Selektionsverfahren** und **Skalierungsmethoden:**
 - Glücksradwahl
 - Rangauswahl
 - Turnirauswahl
 - Anpassung der Fitneßvariation
 - linear dynamische Skalierung
 - σ -Skalierung

Christian Bergelt

Genetische Algorithmen

84

Selektion: Glücksradauswahl

- **Glücksradauswahl** (roulette wheel selection) ist das bekannteste Verfahren.
- Berechne die relative Fitness der Individuen:

$$f_{rel}(s) = \frac{f_{abs}(s)}{\sum s' \in pop(t) f_{abs}(s')}$$

und interpretiere sie als Auswahlwahrscheinlichkeit eines Individuums (sogenannte **fitnepproportionale Selektion**).

- **Beachte:** Die absolute Fitness $f_{abs}(s)$ darf in diesem Fall nicht negativ sein; ggf. ist ein positiver Wert zu addieren oder negative Werte sind Null zu setzen.
- **Beachte:** Die Fitness muß zu maximieren sein.
(Sonst würden schlechte Individuen mit hoher Wahrscheinlichkeit gewählt).
- **Voraussetzung:** Glücksrad mit einem Sektor je Individuum s . Die Sektorgrößen entsprechen den relativen Fitnesswerten $f_{rel}(s)$.

Christina Bergelt

Genetische Algorithmen

86

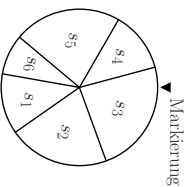
Selektion: Glücksradauswahl

Auswahl eines Individuums:

- Drehe Glücksrad.
- Wähle Chromosom, dessen Sektor an der Markierung liegt.

Auswahl der Zwischengeneration:

- Wiederhole die Auswahl so oft, wie es Individuen in der Population gibt.



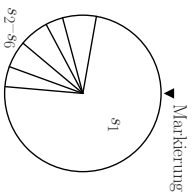
- **Technischer Nachteil der Glücksradauswahl:**
Zur Berechnung der relativen Fitness müssen die Fitnesswerte aller Individuen summiert werden (Normierungsfaktor).
 - Ausgangspopulation muß während der Auswahl konstant bleiben.
 - Parallelisierung der Implementierung wird erschwert.

Christina Bergelt

Genetische Algorithmen

86

Glücksradauswahl: Dominanzproblem



- Hat ein Individuum eine sehr hohe Fitness, kann es die Auswahl **dominieren**.
- In den Folgegenerationen wird diese Dominanz noch verstärkt, da dann Kopien und sehr ähnliche Individuen vorliegen.
- Als Ergebnis kommt es zum **Crowding**: Die Population besteht aus gleichen und sehr ähnlichen Individuen.
 - Crowding führt dazu, daß das (lokale) Optimum sehr schnell gefunden wird.
 - **Nachteil:** Die Diversität der Population geht verloren.
 - Ausbeutung eines oder weniger guter Individuen.
 - Keine Durchforstung des Suchraums; sondern lokale Optimierung. (in späten Generationen erwünscht, am Anfang unerwünscht.)

Christina Bergelt

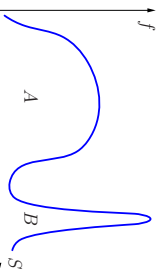
Genetische Algorithmen

87

Selektion: Einfluß der Fitnessfunktion

Das Dominanzproblem weist auf den starken Einfluß der Fitnessfunktion auf die Wirkung der fitnepproportionalen Selektion hin.

- **Problem der vorzeitigen Konvergenz:**
Nimmt die zu maximierende Funktion stark unterschiedliche Werte an, so kann es zu vorzeitiger Konvergenz kommen.
- **Beispiel:** Ist am Anfang im Bereich B kein Chromosom, so bleibt die Population durch die Selektion in der Nähe des (lokalen) Maximums im Bereich A .



Individuen, die sich dem Übergangsbereich zwischen A und B nähern, haben nur sehr schlechte Chancen auf Nachkommen.

Christina Bergelt

Genetische Algorithmen

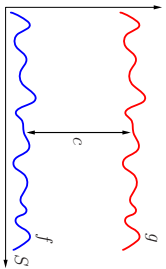
88

Selektion: Einfluß der Fitneßfunktion

Allgemein stellt sich das Problem der **absoluten Höhe** der Fitneßwerte im Vergleich zu ihrer **Variation** (Spannweite); denn es gibt umgekehrt auch das

- Problem des **verschwindenden Selektionsdrucks**:

- Die Maximierung einer Funktion $f : S \rightarrow \mathbb{R}$ ist äquivalent zur Maximierung einer Funktion $g : S \rightarrow \mathbb{R}$ mit $g(s) \equiv f(s) + c$, $c \in \mathbb{R}$.
- Falls $c \gg \sup_{s \in S} f(s)$, so folgt $\forall s \in S : g_{\text{rel}}(s) \approx \frac{1}{\text{popsize}}$
 \rightarrow (zu) geringer Selektionsdruck



Obwohl die Maxima an den gleichen Stellen liegen, sind sie mit einem genetischen Algorithmus unterschiedlich leicht zu finden: mit g gibt es nur (zu) geringe Unterschiede der relativen Fitneß.

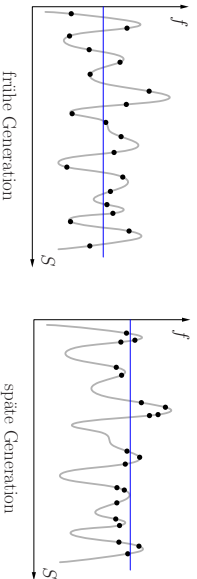
Christian Borgelt

Genetische Algorithmen

80

Selektion: Abnehmender Selektionsdruck

- Da ein genetischer Algorithmus die (durchschnittliche) Fitneß der Individuen tendenziell von Generation zu Generation steigert, erzeugt er u. U. selbst das Problem des verschwindenden Selektionsdrucks.
- Höherer Selektionsdruck am Anfang, da die Fitneßwerte zufällig verteilt sind, geringerer Selektionsdruck in späteren Generationen (umgekehrt wäre besser).
- Beispiel: Die Punkte zeigen die Individuen der Generation.



Christian Borgelt

Genetische Algorithmen

90

Selektion: Anpassung der Fitneßfunktion

Lösungsansatz für die angesprochenen Probleme: **Skalierung der Fitneß**

- **Linear dynamische Skalierung**

$$f_{\text{lin}}(s) = \alpha f(s) - \min\{f(s') \mid s' \in \text{pop}(t)\}, \quad \alpha > 0.$$

Statt des Minimums der aktuellen Population $\text{pop}(t)$ wird auch das Minimum der letzten k Generationen benutzt. Gewöhnlich ist $\alpha > 1$.

- **σ -Skalierung**

$$f_{\sigma}(s) = f(s) - (\mu_f(t) - \beta \cdot \sigma_f(t)), \quad \beta > 0.$$

$$\mu_f(t) = \frac{1}{\text{popsize}} \sum_{s \in \text{pop}(t)} f(s) \quad (\text{Mittelwert der Fitneß})$$

$$\sigma_f(t) = \sqrt{\frac{1}{\text{popsize} - 1} \sum_{s \in \text{pop}(t)} (f(s) - \mu_f(t))^2} \quad (\text{Standardabweichung})$$

- Problem: Wahl der Parameter α und β .

Christian Borgelt

Genetische Algorithmen

91

Selektion: Anpassung der Fitneßfunktion

- Betrachte den **Variationskoeffizienten** der Fitneßfunktion

$$v = \frac{\sigma_f}{\mu_f} = \frac{\sqrt{\frac{1}{|S|-1} \sum_{s \in S} (f(s) - \frac{1}{|S|} \sum_{s \in S} f(s))^2}}{\frac{1}{|S|} \sum_{s \in S} f(s)} \quad \text{bzw. } v(t) = \frac{\sigma_f(t)}{\mu_f(t)}.$$

- Empirisch wurde festgestellt, daß ein Variationskoeffizient $v \approx 0.1$ ein gutes Verhältnis von Durchforstung und Ausbreitung liefert.
- Weicht v von diesem Wert ab, so versucht man, diesen Wert durch Anpassung der Fitneßfunktion f zu erreichen (z.B. durch Skalierung, Exponentiation).
- Für die praktische Berechnung von v wird der Lösungsraum S durch die aktuelle Population $\text{pop}(t)$ ersetzt (v ist nicht berechen-, sondern nur schätzbar).
- Es werden dann in jeder Generation der Wert von $v(t)$ berechnet und die Fitneßwerte entsprechend angepaßt (σ -Skalierung mit $\beta = \frac{1}{v^*}$, $v^* = 0.1$).

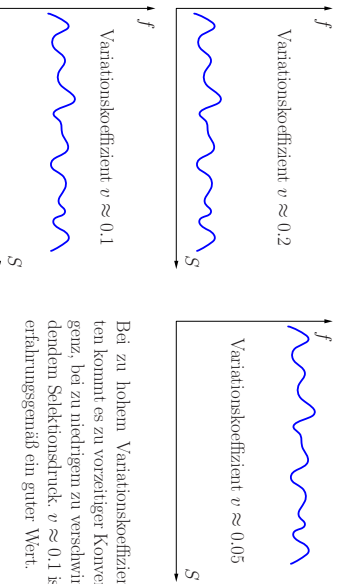
Christian Borgelt

Genetische Algorithmen

92

Selektion: Anpassung der Fitneffunktion

Illustration des Variationskoeffizienten:



Christian Borgelt

Genetische Algorithmen

95

Selektion: Anpassung der Fitneffunktion

• Zeitabhängige Fitneffunktion:

Bestimme die relative Fitneß nicht direkt aus der zu optimierenden Funktion $f(s)$, sondern aus $g(s) \equiv (f(s))^{k(t)}$.

Der zeitabhängige Exponent $k(t)$ steuert den Selektionsdruck.

- **Verfahren zur Bestimmung von $k(t)$** [Michalewicz, 1996]: (soll den Variationskoeffizienten v in der Nähe von $v^* \approx 0.1$ halten)

$$k(t) = \left(\frac{v^*}{v}\right)^{\beta_1} \left(\tan\left(\frac{t}{T+1} \cdot \frac{\pi}{2}\right)\right)^{\beta_2} \left(\frac{v}{v^*}\right)^\alpha$$

- v^* , β_1 , β_2 , α Parameter des Verfahrens
- v Variationskoeffizient (z.B. aus Anfangspopulation geschätzt)
- T maximale Anzahl zu berechnender Generationen
- t aktueller Zeitschritt (Nummer der Generation)
- Empfehlung: $v^* = 0.1$, $\beta_1 = 0.05$, $\beta_2 = 0.1$, $\alpha = 0.1$

Christian Borgelt

Genetische Algorithmen

96

Selektion: Anpassung der Fitneffunktion

- **Boltzmann-Selektion** (andere zeitabhängige Fitneffunktion):

Bestimme die relative Fitneß nicht direkt aus der zu optimierenden Funktion $f(s)$, sondern aus $g(s) \equiv \exp\left(\frac{f(s)}{kT}\right)$.

Der zeitabhängige **Temperaturparameter T** steuert den Selektionsdruck. k ist eine Normierungskonstante.

Die Temperatur kann z.B. linear bis zu einer vorher festgelegten Maximalzahl an Generationen abnehmen.

- Die Idee dieses Auswahlverfahrens ähnelt der des **simulierten Ausglühens**:
 - In frühen Generationen ist der Temperaturparameter hoch, die relativen Unterschiede zwischen den Fitneßwerten daher gering.
 - In späteren Generationen wird der Temperaturparameter gesenkt, wodurch die Fitneßunterschiede größer werden.
 - **Folger:** Im Laufe der Generationen steigt der Selektionsdruck.

Christian Borgelt

Genetische Algorithmen

97

Glückradsauswahl: Varianzproblem

- Die Auswahl der Individuen ist zwar fitneßproportional, aber dennoch zufällig.
- Es gibt keine Garantie, daß gute Individuen in die nächste Generation kommen, nicht einmal für das beste Individuum.
- Allgemein: Die Zahl der Nachkommen eines Individuums kann stark vom Erwartungswert abweichen (**hohe Varianz der Nachkommenzahl**). (Berechnung des Erwartungswertes: siehe Übungsaufgabe)

- Sehr einfache, aber nicht unbedingt empfehlenswerte Lösung:

Diskretisierung des Fitneßwertebereichs

- Berechne Mittelwert $\mu_f(t)$ und Standardabweichung $\sigma_f(t)$ der Population.
- Wenn $\mu_f(t) - \sigma_f(t) > f(s)$: 0 Nachkommen
- Wenn $\mu_f(t) - \sigma_f(t) \leq f(s) \leq \mu_f(t) + \sigma_f(t)$: 1 Nachkomme
- Wenn $f(s) > \mu_f(t) + \sigma_f(t)$: 2 Nachkommen

Christian Borgelt

Genetische Algorithmen

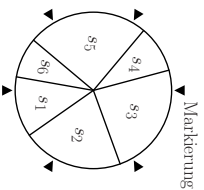
98

Selektion: Erwartungswertmodell

Lösungsmöglichkeit für das Varianzproblem: **Erwartungswertmodell**

- Erzeuge für jeden Lösungskandidaten $[f_n(s) \cdot \text{popsize} \mid \text{Individuen}]$.
- Fülle die (Zwischen-)Population durch Glücksradauswahl auf.

Alternative: **Stochastic Universal Sampling**



Markierung

Auswahl der Zwischenpopulation:

- Drehe Glücksrad einmal.
- Wähle ein Chromosom je Markierung.
- Hier: $1 \times s_1, 1 \times s_2, 2 \times s_3, 2 \times s_4$.
- Überdurchschnittlich gute Individuen kommen sicher in die Zwischenpopulation.

Christian Borgelt

Genetische Algorithmen

97

Selektion: Erwartungswertmodell

Varianten des Erwartungswertmodells:

Erzeugen der restlichen Individuen der (Zwischen-)Population durch

- Verfahren, die aus der **Wahlauswertung** bekannt sind (Mandats-/Sitzverteilung; z.B. größte Reste, Hare-Niemeyer, d'Hondt etc.)
- **Glücksradauswahl**, aber:
 - Für jedes Individuum, daß einen Nachkommen erhält, wird seine Fitness um einen bestimmten Betrag Δf verringert.
 - Wird die Fitness eines Individuums dadurch negativ, so erhält es keine weiteren Nachkommen.
 - Prinzip für die Wahl Δf : Das beste Individuum erhält höchstens eine festgelegte Zahl k von Nachkommen:

$$\Delta f = \frac{1}{k} \max\{f(s) \mid s \in \text{pop}(t)\}.$$

Christian Borgelt

Genetische Algorithmen

98

Selektion: Rangauswahl

- Die Individuen werden nach ihrer Fitness absteigend sortiert. So erhält jedes Individuum einen **Rang** in der Population. (Idee aus der Statistik: verteilungsfreie Verfahren; z.B. Rangkorrelation)
- Über der Rangskala wird eine Wahrscheinlichkeitsverteilung definiert: Je kleiner die Rangnummer, desto größer die Wahrscheinlichkeit.
- Anschließend wird mit dieser Verteilung eine Glücksradauswahl durchgeführt.
- **Vorteile:**
 - Das Dominanzproblem kann weitgehend vermieden werden, da der Wert der Fitnessfunktion nicht direkt die Auswahlwahrscheinlichkeit beeinflusst.
 - Über die Wahrscheinlichkeitsverteilung auf der Rangskala kann der Selektionsdruck sehr bequem gesteuert werden.
- **Nachteile:**
 - Die Individuen müssen sortiert werden (Aufwand: $\text{popsize} \cdot \log \text{popsize}$).

Christian Borgelt

Genetische Algorithmen

99

Selektion: Turnierauswahl

- Für die Auswahl eines Individuums werden k : $2 \leq k < \text{popsize}$; Individuen zufällig aus der Population gezogen (mit oder ohne Zurücklegen; Auswahl *ohne* Berücksichtigung der Fitness, k ist die **Turniergröße**).
- Die Individuen tragen ein Turnier aus; das beste Individuum gewinnt. Der Turniersieger erhält einen Nachkommen in der Zwischenpopulation.
- Anschließend werden *alle* Individuen des Turniers (auch der Sieger) in die aktuelle Population zurückgelegt.
- **Vorteile:**
 - Das Dominanzproblem kann weitgehend vermieden werden, da der Wert der Fitnessfunktion nicht direkt die Auswahlwahrscheinlichkeit beeinflusst.
 - Über die Turniergröße kann m.E. der Selektionsdruck gesteuert werden.
- **Modifikation:** Die relative Fitness der Turnierteilnehmer bestimmt die Gewinnwahrscheinlichkeit (Glücksradauswahl eines Individuums im Turnier).

Christian Borgelt

Genetische Algorithmen

100

Selektion: Elitismus

- Nur im Erwartungswertmodell (oder einer seiner Varianten) ist sichergestellt, daß das beste Individuum in die Zwischengeneration gelangt.
- Auch wenn das beste Individuum in die Zwischengeneration gelangt, ist es nicht vor Veränderungen durch genetische Operatoren geschützt. (Dies gilt auch für das Erwartungswertmodell.)
- **Folglich:** Die Fitness des besten Individuums kann von einer Generation zur nächsten auch wieder abnehmen. (Das ist natürlich unerwünscht.)
- **Lösung: Elitismus**
Das beste Individuum (oder die k , $1 \leq k < \text{popsize}$, besten Individuen) werden *unverändert* in die nächste Generation übernommen. (Die Elite einer Population bleibt erhalten, daher *Elitismus*.)
- **Beachte:** Die Elite wird *nicht* von der normalen Auswahl ausgenommen, da sie ja auch noch durch genetische Operatoren verbessert werden kann.

Christian Bergelt

Genetische Algorithmen

101

Selektion: Elitismus

- Meist ersetzen Nachkommen (Mutations-/Crossover-Produkte) ihre Eltern.
- **„lokaler“ Elitismus** (Elitismus zwischen Eltern und Nachkommen)
 - *Mutation:* Ein mutiertes Individuum ersetzt seinen Elter nur dann, wenn es eine mindestens ebenso gute Fitness besitzt.
 - *Crossover:* Die vier am Crossover beteiligten Individuen (zwei Eltern und zwei Nachkommen) werden nach Fitness sortiert. Die beiden besten Individuen werden in die nächste Generation übernommen.
- **Vorteil**
 - Bessere Konvergenzeigenschaften, da das lokale Optimum konsequenter angestrebt wird.
- **Nachteil**
 - Relativ große Gefahr des Hängenbleibens in lokalen Optima, da keine (lokalen) Verschlechterungen möglich sind.

Christian Bergelt

Genetische Algorithmen

102

Selektion: Nischentechniken

Ziel von Nischentechniken: **Explizites Vermeiden des Crowding**

- **Deterministisches Crowding**
 - *Idee:* Erzeugte Nachkommen ersetzen stets die ihnen ähnlichsten Individuen der Population → Suchraum wird lokal weniger dicht besetzt.
 - *Benötigt:* ein Ähnlichkeits- oder Abstandsmaß für die Individuen (bei binärikodierten Chromosomen z.B. der Hammingabstand)
- **Variante des deterministischen Crowding**
 - Beim Crossover werden zwei Paare von Individuen aus je einem Elter und einem Nachkommen gebildet, wobei ein Nachkomme dem ihm ähnlichsten Elter zugeordnet wird.
 - Aus jedem Paar wird das bessere Individuum übernommen.
 - *Vorteil:* Es sind deutlich weniger Vergleiche zwischen Individuen nötig, da nicht die ganze Population betrachtet wird.

Christian Bergelt

Genetische Algorithmen

103

Selektion: Nischentechniken

- **Sharing**
 - *Idee:* Die Fitness eines Individuums wird reduziert, wenn sich in seiner Nachbarschaft noch weitere Individuen befinden.
 - **Anschaulich: Die Individuen teilen sich die Fitness einer Nische.**
 - *Benötigt:* ein Ähnlichkeits- oder Abstandsmaß für die Individuen
 - *Beispiel:*
$$f_{\text{share}}(s) = \frac{f(s)}{\sum_{s' \in \text{pop}(t)} g(d(s, s'))}$$
 - d:* Abstandsmaß für die Individuen
 - g:* Wichtigkeitsfunktion, die die Form und Größe der Nische definiert, z.B. sogenanntes **power law sharing**

$$g(x) = \begin{cases} 1 - \left(\frac{x}{\sigma}\right)^\alpha, & \text{falls } x < \sigma, \\ 0, & \text{sonst.} \end{cases}$$

σ : Nischenradius, α : steuert die Einflußstärke innerhalb der Nische

Christian Bergelt

Genetische Algorithmen

104

Selektionsverfahren: Charakterisierung

Selektionsverfahren werden gern mit folgenden Begriffspaaren charakterisiert:

statisch Die Auswahlwahrscheinlichkeiten bleiben konstant.

dynamisch Die Auswahlwahrscheinlichkeiten ändern sich.

extinctive (auslöschend) Auswahlwahrscheinlichkeiten dürfen 0 sein.

preservative (erhaltend) Alle Auswahlwahrscheinlichkeiten müssen positiv sein.

rein Individuen dürfen nur in einer Generation Nachkommen haben.

unrein Individuen dürfen in mehreren Generationen Nachkommen haben.

rechts Alle Individuen einer Population dürfen sich vermehren.

links Die besten Individuen einer Population dürfen sich *nicht* vermehren (um vorzeitige Konvergenz zu vermeiden).

generational Die Elterngeneration ist fest, bis alle Nachkommen erzeugt sind.

on the fly Erzeugte Nachkommen ersetzen unmittelbar ihre Eltern.

Christian Borgelt

Genetische Algorithmen

106

Elemente genetischer Algorithmen

3. Genetische Operatoren

Christian Borgelt

Genetische Algorithmen

106

Genetische Operatoren

- Auf einen Teil der ausgewählten Individuen (Zwischengeneration) werden **genetische Operatoren** angewandt, um Varianten und Rekombinationen der bestehenden Lösungskandidaten zu erzeugen.
- Allgemeine Einteilung genetischer Operatoren nach der Zahl der Eltern:
 - Ein-Elter-Operatoren („Mutation“)
 - Zwei-Elter-Operatoren („Crossover“)
 - Mehr-Elter-Operatoren
- Je nach verwendeter Kodierung müssen die genetischen Operatoren bestimmte Eigenschaften haben, z.B.:
 - Sind die Lösungskandidaten durch Permutationen kodiert, so sollten die genetischen Operatoren permutationserhaltend sein.
 - Allgemein: Sind bestimmte Allelkombinationen unsinnig, sollten die genetischen Operatoren sie möglichst nicht erzeugen.

Christian Borgelt

Genetische Algorithmen

107

Genetische Ein-Elter-Operatoren I

Genetische Ein-Elter-Operatoren bezeichnen wir auch allgemein als **Mutation**.

- **Standardmutation**

Anstausch der Ausprägung eines Gens durch eine andere.

$\begin{bmatrix} 3 & 1 & 4 & 2 & 5 & 4 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 1 & 6 & 2 & 5 & 4 & 6 \end{bmatrix}$

- Gegebenfalls werden mehrere Gene mutiert (vgl. n -Damen-Problem).

- *Parameter*: Mutationswahrscheinlichkeit p_m , $0 < p_m \ll 1$
Für Bitstrings ist $p_m = \frac{1}{\text{length}(s)}$ annähernd optimal.

- **Zweiterausch**

Anstausch der Ausprägungen zweier Gene eines Chromosoms.

$\begin{bmatrix} 3 & 1 & 4 & 2 & 5 & 4 & 6 \end{bmatrix} \longrightarrow \begin{bmatrix} 3 & 5 & 4 & 2 & 1 & 4 & 6 \end{bmatrix}$

- *Voraussetzung*: gleiche Allelmenge der ausgetauschten Gene.
- *Verallgemeinerung*: zyklischer Tausch von $3, 4, \dots, k$ Genen.

Christian Borgelt

Genetische Algorithmen

108

Genetische Ein-Elter-Operatoren II

- Verschieben eines Teilstücks

$\begin{bmatrix} 3 & 1 & 4 & 2 & 5 & 4 & 6 \end{bmatrix}$	\longrightarrow	$\begin{bmatrix} 3 & 2 & 5 & 1 & 4 & 4 & 6 \end{bmatrix}$
---	-------------------	---
- Mischen/Permutation eines Teilstücks

$\begin{bmatrix} 3 & 1 & 4 & 2 & 5 & 4 & 6 \end{bmatrix}$	\longrightarrow	$\begin{bmatrix} 3 & 2 & 1 & 5 & 4 & 4 & 6 \end{bmatrix}$
---	-------------------	---
- Inversion (Umdrehen eines Teilstücks)

$\begin{bmatrix} 3 & 1 & 4 & 2 & 5 & 4 & 6 \end{bmatrix}$	\longrightarrow	$\begin{bmatrix} 3 & 5 & 2 & 4 & 1 & 4 & 6 \end{bmatrix}$
---	-------------------	---
- *Voraussetzung:* gleiche Allelhengen im betroffenen Bereich.
- *Parameter:* Ggf. Wahrscheinlichkeitsverteilung über Längen (und Verschiebungswerten für Verschieben eines Teilstücks).

Christina Bergelt

Genetische Algorithmen

109

Genetische Zwei-Elter-Operatoren I

Genetische Zwei-Elter-Operatoren bezeichnen wir auch allgemein als **Crossover**.

- **Ein-Punkt-Crossover**
 - Bestimmen eines zufälligen Schnittpunktes
 - Austausch der Gensequenzen auf einer Seite des Schnittpunktes

$\begin{bmatrix} 5 & 2 & 1 & 1 & 4 & 3 & 6 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$	\longrightarrow	$\begin{bmatrix} 5 & 2 & 1 & 1 & 4 & 5 & 4 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$
--	-------------------	--
- **Zwei-Punkt-Crossover**
 - Bestimmen zweier zufälliger Schnittpunkte
 - Austausch der Gensequenzen zwischen den beiden Schnittpunkten

$\begin{bmatrix} 5 & 2 & 1 & 1 & 4 & 3 & 6 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$	\longrightarrow	$\begin{bmatrix} 5 & 1 & 4 & 2 & 5 & 6 & 1 & 1 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$
--	-------------------	--

Christina Bergelt

Genetische Algorithmen

110

Genetische Zwei-Elter-Operatoren II

- **n-Punkt-Crossover**
 - Verallgemeinerung des Ein- und Zwei-Punkt-Crossover
 - Bestimmen von n zufälligen Schnittpunkten
 - Abwechselndes Austauschen / Nicht-Austauschen der Gensequenzen zwischen zwei aufeinanderfolgenden Schnittpunkten
- **Uniformes Crossover**
 - Für jedes Gen wird einzeln bestimmt, ob es ausgetauscht wird oder nicht (+: ja, -: nein, *Parameter:* Wahrscheinlichkeit p_x für Austausch).

$\begin{bmatrix} 5 & 2 & 1 & 1 & 4 & 3 & 6 & 1 & 1 \\ + & - & + & - & - & + & - & + & - \end{bmatrix}$	\longrightarrow	$\begin{bmatrix} 3 & 2 & 4 & 4 & 3 & 4 & 1 & 1 \\ 3 & 1 & 4 & 2 & 5 & 4 & 6 \end{bmatrix}$
--	-------------------	--
 - **Beachte:** Das uniforme Crossover entspricht **nicht** dem $(\text{length}(s) - 1)$ -Punkt-Crossover! Die Zahl der Crossoverpunkte wird zufällig bestimmt.

Christina Bergelt

Genetische Algorithmen

111

Genetische Zwei-Elter-Operatoren III

- **Shuffle Crossover**
 - Bevor Ein-Punkt-Crossover angewandt wird, werden die Gene zufällig gemischt, nach dem Crossover wieder entmischt.

Mischen	Crossover	Entmischen
$\begin{bmatrix} 5 & 2 & 1 & 1 & 4 & 3 & 6 \\ 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 4 & 2 & 6 & 3 & 5 & 1 & 1 \\ 4 & 2 & 6 & 5 & 1 & 3 \end{bmatrix}$	$\begin{bmatrix} 4 & 2 & 6 & 5 & 3 & 4 \\ 4 & 2 & 6 & 3 & 5 & 1 & 3 \end{bmatrix}$
- **Beachte:** Das Shuffle Crossover ist **nicht** äquivalent zum uniformen Crossover!
 - Beim Shuffle Crossover ist jede Anzahl von Vertauschungen von Genen zwischen den Chromosomen gleichwahrscheinlich, beim uniformen Crossover ist die Anzahl binomialverteilt mit dem Parameter p_x .
 - Das Shuffle Crossover ist eines der empfindlichsten Verfahren.

Christina Bergelt

Genetische Algorithmen

112

Genetische Zwei-Elter-Operatoren IV

Permutationserhaltende Crossover-Operatoren

- **Uniformes ordnungsbasiertes Crossover**
 - Ähnlich wie beim normalen uniformen Crossover wird für jedes Gen einzeln entschieden, ob es erhalten bleibt oder nicht (+: ja, -: nein, *Parameter*: Wahrscheinlichkeit p_k für Erhalt).
 - Die Lücken werden durch die fehlenden Allele aufgefüllt, und zwar in der Reihenfolge, in der sie im *anderen* Chromosom vorkommen.
- $$\begin{array}{cccccccc} \boxed{5} & \boxed{7} & \boxed{2} & \boxed{4} & \boxed{6} & \boxed{3} & \boxed{1} & \boxed{1} \\ + & - & + & + & - & - & + & + \\ \hline \boxed{4} & \boxed{2} & \boxed{3} & \boxed{1} & \boxed{5} & \boxed{7} & \boxed{6} & \boxed{6} \end{array} \longrightarrow \begin{array}{cccccccc} \boxed{5} & \boxed{2} & \boxed{4} & \boxed{6} & \boxed{6} & \boxed{1} & & \boxed{1} \\ + & - & + & + & - & - & + & + \\ \hline \boxed{4} & \boxed{3} & \boxed{1} & \boxed{6} & \boxed{6} & \boxed{6} & & \boxed{6} \end{array} \longrightarrow \begin{array}{cccccccc} \boxed{5} & \boxed{3} & \boxed{2} & \boxed{4} & \boxed{7} & \boxed{6} & \boxed{1} & \boxed{1} \\ + & - & + & + & - & - & + & + \\ \hline \boxed{4} & \boxed{5} & \boxed{3} & \boxed{1} & \boxed{7} & \boxed{2} & \boxed{6} & \boxed{6} \end{array}$$
- Dieses Verfahren erhält **Reihenfolgeinformation**.
 - *Alternativ*: In einem Chromosom werden die durch „+“ markierten, im anderen die durch „-“ markierten Gene erhalten.

Christian Borgelt

Genetische Algorithmen

113

Genetische Zwei-Elter-Operatoren V

Permutationserhaltende Crossover-Operatoren

- **Kantenrekombination** (speziell für Problem des Handlungsreisenden)
 - Chromosom wird als Graph (Kette oder Ring) aufgefaßt. Jedes Gen besitzt Kanten zu seinen Nachbarn im Chromosom.
 - Die Kanten der Graphen zweier Chromosomen werden gemischt, daher der Name *Kantenrekombination*.
 - Dieses Verfahren erhält **Nachbarschaftsinformation**.
- **Vorgehen**:
 1. *Aufbau einer Kantenabelle*:
 - Zu jedem Allel werden seine Nachbarn (in beiden Eltern) aufgelistet. (Gef: sind das erste und das letzte Gen des Chromosoms benachbart.)
 - Hat ein Allel in beiden Eltern den gleichen Nachbarn (Seite irrelevant), so wird dieser Nachbar nur einmal aufgefüllt, dafür aber markiert.

Christian Borgelt

Genetische Algorithmen

114

Zwei-Elter-Operatoren: Kantenrekombination

- **Vorgehen**:
 2. *Aufbau eines Nachkommen*:
 - Das erste Allel wird zufällig aus einem der beiden Eltern gewählt.
 - Ein ausgewähltes Allel wird aus der Kantenabelle gelöscht (aus den Listen der Nachbarn der Allele).
 - Das jeweils nächste Allel wird aus den noch nicht gelöschten Nachbarn des vorangehenden gewählt, wobei die folgende **Prioritätsliste** gilt:
 - a) markierte (d.h. doppelt auftretende) Nachbarn
 - b) Nachbarn mit kürzester Nachbarschaftsliste (wobei markierte Nachbarn einfach zählen)
 - c) zufällige Auswahl eines Nachbarn
 - Ein zweiter Nachkomme kann analog aus dem ersten Allel des anderen Elter erzeugt werden. (Dies wird jedoch meist nicht gemacht.)

Christian Borgelt

Genetische Algorithmen

115

Zwei-Elter-Operatoren: Kantenrekombination

Beispiel: A:

6	3	1	5	2	7	4
---	---	---	---	---	---	---

B:

3	7	2	5	6	1	4
---	---	---	---	---	---	---

Aufbau der Kantenabelle

Allel	Nachbarn in A	Nachbarn in B	zusammengefaßt
1	3, 5	6, 4	3, 4, 5, 6
2	5, 7	7, 5	5*, 7*
3	6, 1	4, 7	1, 4, 6, 7
4	7, 6	1, 3	1, 3, 6, 7
5	1, 2	2, 6	1, 2*, 6
6	4, 3	5, 1	1, 3, 4, 5
7	2, 4	3, 2	2*, 3, 4

- Beide Chromosomen werden als Ring aufgefaßt (erstes und letztes Gen sind benachbart):
In **A** ist der linke Nachbar der 6 die 4, der rechte Nachbar der 4 die 6; in **B** analog.
- In beiden Chromosomen stehen 5, 2 und 7 nebeneinander — das sollte erhalten werden (zeigt sich in Markierungen).

Christian Borgelt

Genetische Algorithmen

116

Zwei-Elter-Operatoren: Kantenrekombination

Aufbau eines Nachkommen

6 5 2 7 4 3 1

Allel	Nachbarn	Wahl: 6	5	2	7	4	3	1
1	3, 4, 5, 6	3, 4, 5	3, 4	3, 4	3, 4	3	—	—
2	5*, 7*	5*, 7*	7*	7*	—	—	—	—
3	1, 4, 6, 7	1, 4, 7	1, 4, 7	1, 4, 7	1, 4	1	1	—
4	1, 3, 6, 7	1, 3, 7	1, 3, 7	1, 3, 7	1, 3	1, 3	—	—
5	1, 2*, 6	1, 2*	1, 2*	—	—	—	—	—
6	1, 3, 4, 5	1, 3, 4, 5	—	—	—	—	—	—
7	2*, 3, 4	2*, 3, 4	2*, 3, 4	3, 4	3, 4	—	—	—

- Starte mit einem Allel des Chromosoms **A**, also der 6, und streiche die 6 aus allen Nachbarschaftslisten (dritte Spalte).
- Da unter den Nachbarn der 6 (das sind 1, 3, 4, 5) die 5 die kürzeste Liste hat, wird die 5 für das zweitens Gen gewählt. Dann folgt die 2, die 7 usw.

Christian Borgelt

Genetische Algorithmen

117

Zwei-Elter-Operatoren: Kantenrekombination

- Der Nachbarname hat meist eine neue Kante (vom letzten zum ersten Gen).
- Die Kantenrekombination kann auch angewendet werden, wenn das erste und das letzte Gen eines Chromosoms nicht als benachbart angesehen werden: Die entsprechenden Kanten werden dann nicht in die Kantenabelle aufgenommen.
- Werden erstes und letztes Gen als benachbart angesehen, kann das Startallel beliebig aus den Chromosomen gewählt werden. Werden sie dagegen nicht als benachbart angesehen, muß es ein am Anfang stehendes Allel sein.
- Beim Aufbau eines Nachkommen kann es vorkommen, daß die Nachbarschaftsliste des gerade ausgewählten Allels leer ist.
(Die Prioritätsregeln dienen dazu, die Wahrscheinlichkeit dafür gering zu halten; sie sind aber nicht perfekt.)
In diesem Fall wird die Konstruktion mit einem Allel fortgesetzt, das zufällig aus den noch übrigen Allelen gewählt wird.

Christian Borgelt

Genetische Algorithmen

118

Genetische Drei- und Mehr-Elter-Operatoren

- **Diagonal-Crossover**
 - Ähnlich wie Ein-, Zwei- und n -Punkt-Crossover, aber für mehr Eltern.
 - Bei drei Eltern werden zwei Crossover-Punkte gewählt.
 - An den Schnittstellen werden die Gensequenzen diagonal und zyklisch über die Chromosomen verschoben.



- Eine Verallgemeinerung auf mehr als drei Eltern ist naheliegend. Für k Eltern werden $k - 1$ Crossover-Punkte gewählt.
- Dieses Verfahren führt zu einer sehr guten Durchforstung des Suchraums, besonders bei großer Elternzahl (10–15 Eltern).

Christian Borgelt

Genetische Algorithmen

119

Charakterisierung von Crossover-Operatoren

- **Ortsabhängige Verzerrung (positional bias)**
 - Liegt vor, wenn die Wahrscheinlichkeit, daß zwei Gene zusammen vererbt werden (im gleichen Chromosom bleiben, zusammen in das andere Chromosom wandern), von ihrer relativen Lage im Chromosom abhängt.
 - Ist unwahrscheinlich, da dann die Anordnung der Gene im Chromosom entscheidenden Einfluß auf den Erfolg oder Mißerfolg des genetischen Algorithmus haben kann (bestimmte Anordnungen lassen sich schwerer erreichen).
- **Beispiel: Ein-Punkt-Crossover**
 - Zwei Gene werden voneinander getrennt (gelangen in verschiedene Nachkommen), wenn der Crossover-Punkt zwischen sie fällt.
 - Je näher zwei Gene im Chromosom beieinander liegen, desto so weniger mögliche Crossover-Punkte gibt es zwischen ihnen.
 - *Folglich:* Nebeneinander liegende Gene werden mit höherer Wahrscheinlichkeit als entfernt liegende in den gleichen Nachkommen gelangen.

Christian Borgelt

Genetische Algorithmen

120

Charakterisierung von Crossover-Operatoren

- **Verteilungsverzerrung (distributional bias)**
 - Liegt vor, wenn die Wahrscheinlichkeit, daß eine bestimmte Anzahl von Genen ausgetauscht wird, nicht für alle Anzahlen gleich ist.
 - Ist oft unerwünscht, da dann Teilösungen unterschiedlicher Größe unterschiedlich gute Chancen haben, in die nächste Generation zu gelangen.
 - Die Verteilungsverzerrung ist meist weniger kritisch (d.h. eher tolerierbar) als die ortalabhängige Verzerrung.
- **Beispiel: uniformes Crossover**
 - Da jedes Gen unabhängig von allen anderen mit der Wahrscheinlichkeit p_x ausgetauscht wird, ist die Anzahl k der ausgetauschten Gene binomialverteilt mit dem Parameter p_x :
$$P(K = k) = \binom{n}{k} p_x^k (1 - p_x)^{n-k} \quad \text{mit} \quad n \hat{=} \text{Gesamtzahl der Gene.}$$
 - *Folglich:* Sehr kleine und sehr große Anzahlen sind unwahrscheinlicher.

Christian Borgelt

Genetische Algorithmen

121

Theoretische Betrachtung: Das Schematheorem

Christian Borgelt

Genetische Algorithmen

122

Das Schematheorem

- Frage: **Warum funktionieren genetische Algorithmen?**
- **Ansatz** von [Holland 1975]:

Betrachte Chromosomenschemata (d.s. nur teilweise festgelegte Chromosomen) und untersuche, wie sich die Zahl der Chromosomen, die zu einem Schema passen, über die Generationen hinweg entwickelt.
- **Ziel:** Eine zumindest grobe stochastische Aussage darüber, wie genetische Algorithmen den Suchraum durchforsten.
- Zur **Vereinfachung** der Darstellung: Beschränkung auf
 - Bitfolgen (Chromosomen aus Nullen und Einsen) mit fester Länge L
 - Fitnessproportionale Selektion (Gitterradtauswahl)
 - Standardmutation (Änderung eines zufällig gewählten Bits)
 - Ein-Punkt-Crossover (Durchschneiden an einer Stelle und Vertauschen)

Christian Borgelt

Genetische Algorithmen

123

Das Schematheorem: Schemata

- **Definition: Schema**

Ein **Schema** h ist eine Zeichenkette der Länge L über dem Alphabet $\{0, 1, *\}$, d.h. $h \in \{0, 1, *\}^L$.
Das Zeichen $*$ heißt **Jokerzeichen** oder **Don't-Care-Symbol**.
- **Definition: Passung**

Ein Chromosom $c \in \{0, 1\}^L$ **paßt zu einem Schema** $h \in \{0, 1, *\}^L$, in Zeichen: $c \triangleleft h$, wenn es mit h an allen Stellen übereinstimmt, an denen h eine 0 oder eine 1 enthält. (Stellen, an denen ein $*$ steht, bleiben unberücksichtigt.)
- **Beispiel:**

$h =$	**0*1*10*	Schema der Länge 10
$c_1 =$	1100111100	paßt zu h , also $c_1 \triangleleft h$
$c_2 =$	1111111111	paßt nicht zu h , also $c_2 \not\triangleleft h$
- Es gibt 2^L Chromosomen und 3^L Schemata.
Jedes Chromosom paßt zu $\sum_{h=0}^L \binom{L}{h} = 2^L$ Schemata.

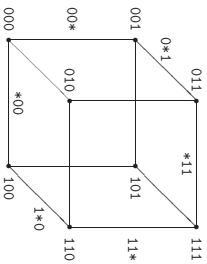
Christian Borgelt

Genetische Algorithmen

124

Schemata: Hyperebenen

- Jedes Schema beschreibt eine Hyperebene in einem Hyperinheitswürfel (allerdings nur Ebenen, die parallel oder senkrecht zu den Achsen stehen).



- Beispiele:** *00 $\hat{=}$ Kante von 000 nach 100 (vorne unten)
0** $\hat{=}$ linke Würfelfläche
*** $\hat{=}$ gesamter Würfel

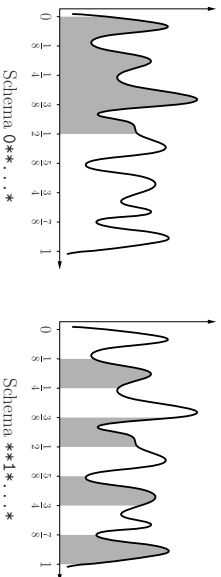
Christian Borgelt

Genetische Algorithmen

125

Schemata: Wertebereiche von Funktionen

- Gegeben: reelle Funktion $f : [0, 1] \rightarrow \mathbb{R}$
- Annahme: Binärikodierung der Funktionsargumente (kein Gray-Code)
- Jedem Schema entspricht ein „Streifenmuster“ im Definitionsbereich von f :



- Schemata mit Gray-Kodierung: siehe Übungsaufgabe

Christian Borgelt

Genetische Algorithmen

126

Das Schematheorem: Einfluß der Selektion

- Um die Verteilung von Chromosomen, die zu einem Schema passen, verfolgen zu können, müssen wir untersuchen, wie sich die **Selektion** und die **genetischen Operatoren** (Mutation und Crossover) auswirken.
- Für die Selektion müssen wir untersuchen, welche Fittest Chromosomen haben, die zum Schema h passen. Ansatz: Mittelung über alle Chromosomen.

- Definition: Mittlere Fittest**

Die **mittlere relative Fittest** der Chromosomen, die in der Generation $\text{pop}(t)$ zum Schema h passen, ist

$$f_{\text{rel}}(h) = \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} f_{\text{rel}}(c)}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|}$$

- Die durchschnittliche Anzahl Nachkommen eines zu einem Schema h passenden Chromosoms ist $f_{\text{rel}}(h) \cdot \text{popsize}$. Die zu erwartende Zahl Chromosomen, die nach der Auswahl zu einem Schema h passen, ist daher: (Zahl vorher passender Chromosomen) $\cdot f_{\text{rel}}(h) \cdot \text{popsize}$.

Christian Borgelt

Genetische Algorithmen

127

Das Schematheorem: Einfluß der Selektion

- Weitere Betrachtungen zur relativen Fittest eines Schemas:

$$\begin{aligned} f_{\text{rel}}(h) \cdot \text{popsize} &= \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} f_{\text{rel}}(c)}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|} \cdot \text{popsize} \\ &= \frac{\sum_{c \in \text{pop}(t), c \triangleleft h} \sum_{c' \in \text{pop}(t)} f(c')}{\sum_{c \in \text{pop}(t), c \triangleleft h} \sum_{c' \in \text{pop}(t)} f(c')} \cdot \text{popsize} \\ &= \frac{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|}{|\{c \in \text{pop}(t) \mid c \triangleleft h\}|} \cdot \frac{\sum_{c \in \text{pop}(t)} f(c)}{\sum_{c' \in \text{pop}(t)} f(c')} \\ &= \frac{f(h)}{f_i} \end{aligned}$$

- $\overline{f(h)}$ mittlere Fittest der in der t -ten Generation zum Schema h passenden Chromosomen
- $\overline{f_i}$ mittlere Fittest aller Chromosomen der t -ten Generation
- Die mittlere Anzahl Nachkommen kann so durch das Verhältnis der mittleren Güte eines Schemas zur Gesamtdurchschnittsgüte ausgedrückt werden.

Christian Borgelt

Genetische Algorithmen

128

Das Schematheorem: Einfluß der Mutation

- Für die genetischen Operatoren brauchen wir Maße, mit denen wir für ein Schema die Wahrscheinlichkeit angeben können, daß durch Anwendung eines Operators die Passung zu diesem Schema verlorengeht bzw. erhalten bleibt.
- **Definition: Ordnung** (für die Standardmutation)
Die **Ordnung** eines Schemas h ist die Anzahl der Nullen und Einsen in h , also $\text{ord}(h) = \#0 + \#1 = L - \#*$ ($\#$: Anzahl des Auftretens).
Beispiel: $\text{ord}(**0*1*1*10*) = 5$.
- Durch eine Standardmutation geht die Passung zu einem Schema verloren mit Wahrscheinlichkeit $p_{m_i}^*(h) = \frac{\text{ord}(h)}{L}$, falls das Bit umgekehrt wird, Wahrscheinlichkeit $p_{m_i}^*(h) = \frac{\text{ord}(h)}{2L}$, falls das neue Bit zufällig bestimmt wird.
- **Erläuterung:** Jedes der L Gene eines Chromosoms der Länge L wird mit gleicher Wahrscheinlichkeit für eine Mutation ausgewählt.

Christina Bergelt

Genetische Algorithmen

129

Das Schematheorem: Einfluß des Crossover

- **Definition: Definierende Länge** (für das Ein-Punkt-Crossover)
Die **definierende Länge** eines Schemas h ist die Differenz zwischen der Positionnummer der letzten 0/1 und der Positionnummer der ersten 0/1 in h .
Beispiel: $\text{dl}(**0*1*1*10*) = 9 - 3 = 6$.
- Beim Ein-Punkt-Crossover liegt der Schnittpunkt mit Wahrscheinlichkeit $p_c^*(h) = \frac{\text{dl}(h)}{L-1}$ so, daß zwei Nicht-Jokerzeichen voneinander getrennt werden.
- **Erläuterung:** Bei Chromosomen der Länge L gibt es $L - 1$ mögliche Schnittpunkte für das Ein-Punkt-Crossover, die alle gleichwahrscheinlich sind. $\text{dl}(h)$ dieser Schnittpunkte liegen so, daß im Schema festgelegte Gene in verschiedenen Nachkommen gelangen, wodurch die Passung verlorengehen kann.
- **Achtung:** Die Passung kann, muß jedoch nicht zwangsläufig verlorengehen.
→ Für die Rechnung sind weitere Überlegungen nötig (später).

Christina Bergelt

Genetische Algorithmen

130

Das Schematheorem: Definitionen

- **Erwartungswert passender Chromosomen**
 $N(h, t)$ ist der Erwartungswert der Anzahl Chromosomen, die in der t -ten Generation zum Schema h passen.
- **Erwartungswert nach Selektion**
 $N(h, t + \Delta t_s)$ ist der Erwartungswert der Anzahl Chromosomen, die in der t -ten Generation nach Selektion zum Schema h passen.
- **Erwartungswert nach Crossover**
 $N(h, t + \Delta t_c + \Delta t_d)$ ist der Erwartungswert der Anzahl Chromosomen, die in der t -ten Generation nach Selektion und Crossover zum Schema h passen.
- **Erwartungswert nach Mutation**
 $N(h, t + \Delta t_m + \Delta t_c + \Delta t_d) = N(h, t + 1)$ ist der Erwartungswert der Anzahl Chromosomen, die in der t -ten Generation nach Selektion, Crossover und Mutation (und damit in der $t + 1$ -ten Generation) zum Schema h passen.

Christina Bergelt

Genetische Algorithmen

131

Das Schematheorem: Selektion

- **Gesucht:** Zusammenhang zwischen $N(h, t)$ und $N(h, t + 1)$.
- **Vorgehen:** Wir betrachten schrittweise die Auswirkungen der Selektion, des Crossovers und der Mutation mit Hilfe der mittleren Fitness, der Ordnung und der dehnerenden Länge eines Schemas.
- **Selektion:**
Die Auswirkungen der Selektion werden durch die mittlere Fitness beschrieben:
$$N(h, t + \Delta t_s) = N(h, t) \cdot f_{\text{rel}}(h) \cdot \text{popsize}$$
$$N(h, t) \cdot f_{\text{rel}}(h)$$
 Wahrscheinlichkeit, daß ein zum Schema h passendes Chromosom ausgewählt wird
 $f_{\text{rel}}(h) \cdot \text{popsize}$ Durchschnittliche Anzahl Nachkommen eines zum Schema h passenden Chromosoms
- **Beachte:** Die relative Fitness $f_{\text{rel}}(h)$ ist nicht exakt bestimmt, da die zum Schema h passenden Chromosomen nur als Erwartungswert bekannt sind.

Christina Bergelt

Genetische Algorithmen

132

Das Schematheorem: Crossover

- **Crossover:**

Die Auswirkungen des Crossover werden beschrieben durch:

$$N(h, t + \Delta t_c) = (1 - p_c) \cdot \underbrace{N(h, t + \Delta t_s)}_A + \underbrace{p_c \cdot N(h, t + \Delta t_s)}_B \cdot (1 - p_{\text{loss}}) + C$$

- p_c Wahrscheinlichkeit eines Crossover
- p_{loss} Wahrscheinlichkeit, daß durch ein Ein-Punkt-Crossover die Passung eines Chromosoms zum Schema h verlorengeht
- A Erwartungswert der Anzahl Chromosomen, die zum Schema h passen und *nicht* am Crossover teilnehmen
- B Erwartungswert der Anzahl Chromosomen, die am Crossover teilnehmen und deren Passung zum Schema h dadurch nicht verlorengeht
- C Gewinne an Chromosomen, die zum Schema h passen, durch ... (siehe Übungsaufgabe)

Christian Bergelt

Genetische Algorithmen

135

Das Schematheorem: Crossover

Betrachtungen zur Wahrscheinlichkeit p_{loss} :

- **Beispiele:**

$h = **0*1*1*$	\rightarrow	$**0*1*1*$	$= h$
$h \triangleright c_1 = 0000 1111$	\rightarrow	00000000	$= c_1^c \neq h$
$h \not\triangleright c_2 = 1111 0000$	\rightarrow	11111111	$= c_2^c \neq h$
$h = **0*1*1*$	$=$	$**0*1*1*$	$= h$
$h \triangleright c_1 = 0000 1111$	\rightarrow	00001010	$= c_1^c \triangleleft h$
$h \triangleright c_2 = 1101 1010$	\rightarrow	11011111	$= c_2^c \triangleleft h$

- **Folglich:**

$$p_{\text{loss}} \leq \underbrace{\frac{\text{dl}(h)}{L-1}}_{A=\chi_c^2(h)} \cdot \left(1 - \underbrace{\frac{N(h, t + \Delta t_s)}{\text{popsize}}}_B\right)$$

- A Wahrscheinlichkeit, daß der Schnittpunkt zwischen festgelegte Gene fällt
- B Wahrscheinlichkeit, daß das zweite Chromosom zum Schema h paßt

- **Frage:** Warum gilt nur \leq und nicht $=$? (siehe Übungsaufgabe)

Christian Bergelt

Genetische Algorithmen

135

Das Schematheorem: Crossover

- Einsetzen des Ausdrucks für p_{loss} liefert:

$$\begin{aligned} & N(h, t + \Delta t_s + \Delta t_c) \\ & \geq (1 - p_c) \cdot N(h, t + \Delta t_s) \\ & \quad + p_c \cdot N(h, t + \Delta t_s) \cdot \left(1 - \frac{\text{dl}(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\text{popsize}}\right)\right) \\ & = N(h, t + \Delta t_s) \left(1 - p_c + p_c \cdot \left(1 - \frac{\text{dl}(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\text{popsize}}\right)\right)\right) \\ & = N(h, t + \Delta t_s) \cdot \left(1 - p_c \frac{\text{dl}(h)}{L-1} \cdot \left(1 - \frac{N(h, t + \Delta t_s)}{\text{popsize}}\right)\right) \\ & \quad (*) \quad N(h, t) \cdot f_{\text{rel}}(h) \cdot \text{popsize} \cdot \left(1 - p_c \frac{\text{dl}(h)}{L-1} \cdot (1 - N(h, t) \cdot f_{\text{rel}}(h))\right) \end{aligned}$$

- Im Schritt (*) wurde zweimal die vorher abgeleitete Beziehung $N(h, t + \Delta t_s) = N(h, t) \cdot f_{\text{rel}}(h) \cdot \text{popsize}$ ausgenutzt.

Christian Bergelt

Genetische Algorithmen

135

Das Schematheorem: Mutation

- **Mutation:**

Die Auswirkungen der Mutation werden durch die Ordnung beschrieben:

$$N(h, t+1) = N(h, t + \Delta t_s + \Delta t_c + \Delta t_m) = N(h, t + \Delta t_s + \Delta t_c) \cdot (1 - p_m)^{\text{ord}(h)}$$

p_m Mutationswahrscheinlichkeit für jedes Bit,

d.h. jedes Bit wird mit Wahrscheinlichkeit p_m mutiert (umgedreht) und mit Wahrscheinlichkeit $(1 - p_m)$ unverändert gelassen.

Erklärung: Damit die Passung nicht verlorengeht, darf keines der $\text{ord}(h)$ Gene verändert werden, die im Schema h festgelegt sind.

- **Alternative Modelle** sind möglich, z.B.:

Jedes Chromosom wird genau einer Bitmutation unterworfen.

$$N(h, t+1) = N(h, t + \Delta t_s + \Delta t_c + \Delta t_m) = N(h, t + \Delta t_s + \Delta t_c) \cdot \frac{\text{ord}(h)}{L}$$

Wahrscheinlichkeit, daß durch die Bitmutation ein im Schema h festgelegtes Gen verändert wird (gleichwahrscheinliche Auswahl des Bits).

Christian Bergelt

Genetische Algorithmen

136

Das Schematheorem

- Insgesamt (mit erstem Mutationsmodell):
$$N(h, t + 1) = f_{\text{rel}}(h) \cdot \text{popsize} \cdot \left(1 - p_c \frac{L}{L-1}\right) \cdot (1 - N(h, t) \cdot f_{\text{rel}}(h)) \cdot (1 - p_m)^{\text{ord}(h)} \cdot N(h, t)$$
- Einsetzen des Fitneßverhältnisses liefert das **Schematheorem**:
$$N(h, t + 1) = \frac{\overline{f}(h)}{\overline{f}} \left(1 - p_c \frac{L}{L-1}\right) \left(1 - \frac{N(h, t)}{\text{popsize}} \cdot \frac{\overline{f}(h)}{\overline{f}}\right) (1 - p_m)^{\text{ord}(h)} \cdot N(h, t)$$
- **Interpretation:** Schemata mit
 - überdurchschnittlicher mittlerer Bewertung,
 - kurzer definierender Länge und
 - geringer Ordnungvermehren sich besonders stark (etwa exponentiall).

Christian Borgelt

Genetische Algorithmen

137

Das Schematheorem: Baustein-Hypothese

- Das Schematheorem besagt, daß der Suchraum besonders gut in Hyperplanen (Regionen) durchstreicht wird, die Schemata mit hoher mittlerer Fitneß, kleiner definierender Länge und geringer Ordnung entsprechen.
(Dem in diesen Regionen vermehren sich die Chromosomen am stärksten)
- Solche Schemata heißen **Bausteine** (engl.: **building blocks**); die obige Aussage heißt daher auch **Baustein-Hypothese**.
- **Beachte:** Die obige Form der Bausteinhypothese gilt nur für Bitfolgen, fitnessproportionale Selektion, Standardmutation und Ein-Punkt-Crossover.
Bei Verwendung z.B. anderer genetischer Operatoren werden Bausteine n.U. durch andere Eigenschaften charakterisiert (siehe Übungsaufgabe).
(Die hohe mittlere Fitneß ist jedoch stets eine Eigenschaft, da jedes Selektionsverfahren Chromosomen mit hoher Fitneß bevorzugt, wenn auch unterschiedlich stark und nicht immer in direkter Relation zum Fitneßwert.)

Christian Borgelt

Genetische Algorithmen

138

Das Schematheorem: Kritik

- Das Schematheorem gilt in Strenge nur bei unendlicher Populationsgröße. (Sonst kann es Abweichungen von den betrachteten Erwartungswerten geben, die nicht vernachlässigbar sind.)
Diese Annahme kann natürlich in der Praxis nie erfüllt werden. Es kommt daher immer zu Abweichungen vom idealen Verhalten (*stochastische Drift*).
- Es wird implizit angenommen, daß es kaum Wechselwirkungen zwischen den Genen gibt (*geringe Epistasie*), also die Fitneß von Chromosomen, die zu einem Schema passen, sehr ähnlich ist.
- Ein dritter, oft angeführter Einwand ist:
Es wird implizit angenommen, daß interessierende Gene im Chromosom eng zusammen liegen, um möglichst kleine Bausteine (*building blocks*) zu bilden. Dieser Einwand betrifft jedoch eigentlich die Beschränkung auf das Ein-Punkt-Crossover und nicht den Ansatz an sich. Man kann durch andere Maße als die definierende Länge begegnet werden, die operationenspezifisch sind.

Christian Borgelt

Genetische Algorithmen

139

Genetische Algorithmen zur Verhaltenssimulation

Christian Borgelt

Genetische Algorithmen

140

Genetische Algorithmen zur Verhaltenssimulation

- Bisher: Verwendung genetischer Algorithmen um Optimierungsprobleme (numerische oder diskrete) zu lösen.
- Jetzt: Verwendung genetischer Algorithmen um Verhalten zu simulieren (Populationsdynamik) und Verhaltensstrategien zu finden.
- **Grundlagentheorie**
 - Dient der Analyse sozialer und wirtschaftlicher Situationen.
 - Modellierung von Handlungen als Spielzüge in einem festgelegten Rahmen.
 - Wichtigste theoretische Grundlage der Wirtschaftswissenschaften.
- **Allgemeiner Ansatz:**
 - Kodiere die Verhaltensstrategie eines Akteurs in einem Chromosom.
 - Lasse Akteure miteinander interagieren und bewerte ihren Erfolg.
 - Akteure vermehren sich oder sterben aus, je nach erzieltem Erfolg.

Christian Bergelt

Genetische Algorithmen

111

Das Gefangenendilemma

Das bekannteste Problem der Spieltheorie ist das **Gefangenendilemma** (engl.: prisoner's dilemma)

- Zwei Personen haben einen Bankdiebstahl begangen und werden verhört.
- Die Beweise reichen jedoch nicht aus, um sie in einem Indizienprozess wegen des Bankdiebstahls zu verurteilen.
- Die Beweise reichen jedoch aus, um sie wegen eines geringfügigeren Deliktes (z.B. unerlaubter Waffensbesitz) zu verurteilen (Strafmaß: 1 Jahr Gefängnis).
- Angebot des Staatsanwaltes: Kronzeugenregelung
 - Gesteht einer der beiden die Tat, wird er Kronzeuge und nicht verurteilt.
 - Der andere dagegen wird mit voller Härte bestraft (10 Jahre Gefängnis)
 - Problem: Gestehen beide, gilt die Kronzeugenregelung nicht. Da sie jedoch beide geständig sind, erhalten sie mildernde Umstände zugesprochen (Strafe: je 5 Jahre Gefängnis)

Christian Bergelt

Genetische Algorithmen

112

Das Gefangenendilemma

Zur Analyse des Gefangenendilemmas benutzt man eine **Anzahlungsmatrix**:

	A	B	
A	schweigt	gesteht	
schweigt	-1	-10	0
gesteht	-10	-5	-5
B	schweigt	gesteht	
schweigt	-1	-10	0
gesteht	0	-5	-5

- Kooperation (beide schweigen) ist insgesamt am günstigsten.
 - **Aber:** Doppeltres Geständnis ist das **Nash-Gleichgewicht**.
Nash-Gleichgewicht: Keine der beiden Seiten kann ihre Auszahlung erhöhen, wenn nur sie ihre Aktion ändert.
- Jede Auszahlungsmatrix hat mindestens ein Nash-Gleichgewicht [Nash 1950].

Christian Bergelt

Genetische Algorithmen

113

Das Gefangenendilemma

Allgemeine **Anzahlungsmatrix** des Gefangenendilemmas:

	A	B	
A	cooperat	defect	
cooperat	R	S	T
defect	T	P	P
B	cooperat	defect	
cooperat	R	S	T
defect	T	P	P

- **R:** Reward for mutual cooperation
 - **T:** Temptation to defect
 - **P:** Punishment for mutual defection
 - **S:** Sucker's payoff
- Die genauen Werte für **R**, **P**, **T** und **S** sind nicht wichtig.
- Es muß aber gelten $T > R > P > S$ und $2R > T + S$
(Wenn die 2. Bedingung nicht erfüllt ist, ist wechselseitiges Ausweichen besser.)

Christian Bergelt

Genetische Algorithmen

114

Das Gefangenendilemma

- Viele Alltagssituation kann man mit dem Gefangenendilemma beschreiben.
- **Aber:** Obwohl der doppelte Defekt das Nash-Gleichgewicht ist, beobachten wir auch anderes (kooperatives) Verhalten.
- Fragestellung (nach [Axelrod 1980]):
Unter welchen Bedingungen entsteht Kooperation in einer Welt von Egoisten ohne zentrale Autorität?
- Antwort von [Hobbes 1651] (Leviathan):
 - **Gar nicht!**
 - Eine staatliche Ordnung existierte, wurde der Naturzustand dominiert von egoistischen Individuen, die so rücksichtslos gegeneinander weiterfeierten, daß das Leben „solitary, poor, nasty, and short“ war.
 - **Aber:** Auf internationaler Ebene gibt es *de facto* keine zentrale Autorität, aber dennoch (wirtschaftliche und politische) Kooperation von Staaten.

Christian Borgelt

Genetische Algorithmen

116

Das Gefangenendilemma

- **Ansatz** von [Axelrod 1980]:
Betrachte das **iterierte Gefangenendilemma**
(Das Gefangenendilemma wird von zwei Spielern mehrfach hintereinander gespielt, wobei sie die vergangenen Züge des jeweils anderen Spielers kennen.)
- **Idee** dieses Ansatzes:
 - Wird das Gefangenendilemma nur *einmal* gespielt, ist es am günstigsten, das Nash-Gleichgewicht zu wählen.
 - Wird das Gefangenendilemma *mehrfach* gespielt: kann ein Spieler auf unkooperatives Verhalten des anderen reagieren.
(Möglichkeit der *Vergeltung* für erlittene Nachteile)
- Fragestellungen:
Entsteht im iterierten Gefangenendilemma Kooperation?
Was ist die beste Strategie im iterierten Gefangenendilemma?

Christian Borgelt

Genetische Algorithmen

116

Das Gefangenendilemma

[Axelrod 1980] legte folgende **Anzahlungsmatrix** fest:

	A	B	
A	cooperate	defect	
cooperate	3	0	5
defect	5	0	1

(Satz von kleinsten nicht-negativen ganzen Zahlen, die die Bedingungen erfüllen.)

- Wissenschaftler verschiedener Disziplinen (Psychologie, Sozial- und Politikwissenschaften, Wirtschaftswissenschaften, Mathematik) wurden eingeladen, Programme zu schreiben, die das iterierte Gefangenendilemma spielen.
- Ein Programm kann sich die eigenen und die gegnerischen Züge merken.

Christian Borgelt

Genetische Algorithmen

116

Das Gefangenendilemma: Turniere

Zur Beantwortung der genannten Fragen führte Axelrod zwei Turniere durch:

- **1. Turnier:**
 - 14 Programme plus ein Zufallsspieler (Fortran)
 - Rundenturnier mit 200 Spielen je Paarung
 - Sieger: A. Rapoport mit 'Tit-for-Tat' (Wie du mir, so ich dir.)
- Die Programme und Ergebnisse des ersten Turniers wurden veröffentlicht, dann wurde zu einem zweiten Turnier eingeladen.
Idee: Ergebnisanalyse ermöglicht es ggf., bessere Programme zu schreiben.
- **2. Turnier:**
 - 62 Programme plus ein Zufallsspieler (Fortran und Basic)
 - Rundenturnier mit 200 Spielen je Paarung
 - Sieger: A. Rapoport mit 'Tit-for-Tat' (Wie du mir, so ich dir.)

Christian Borgelt

Genetische Algorithmen

116

Das Gefangenendilemma: Tit-for-Tat

- Die Spielstrategie von **Tit-for-Tat** ist *sehr* einfach:
 - Kooperiere im ersten Spiel (spiele C).
 - Mache in allen folgenden Spielen den Zug des Gegners aus dem direkt vorangehenden Spiel.
- **Beachte:** Reines Tit-for-Tat ist nicht unbedingt die beste Strategie, wenn gegen *einzelne* andere Strategien gespielt wird.
 - Nur wenn es in einer Population Individuen gibt, mit denen Tit-for-Tat kooperieren kann, schnidet es insgesamt sehr gut ab.
- **Problem** von Tit-for-Tat: Es ist **anfällig für Fehler**.
 - Spiel: Tit-for-Tat gegen Tit-for-Tat und spielt einer der beiden Spieler „aus Versehen“ Defekt, so kommt es zu wechselseitigen Vergeltungsschlägen.
- Eine wichtige Alternative ist **Tit-for-Two-Tat**:
 - Schläge erst nach zweimaligen Defekt des Gegners zurück.

Christian Borgelt

Genetische Algorithmen

149

Das Gefangenendilemma: Genetischer Algorithmus

Kodierung der Spielstrategien: [Axelrod 1987]

- Betrachte alle möglichen Spielverläufe der Länge 3 ($2^3 = 64$ Möglichkeiten).
 - Speichere für jeden Spielverlauf den im nächsten Spiel auszuführenden Zug (C — kooperate oder D — defekt, kodiert in einem Bit):
 1. Spiel
 2. Spiel
 3. Spiel
- | | |
|--|---|
| | C |
| 1. Bit: Antwort auf (C,C), (C,C), (C,C): | D |
| 2. Bit: Antwort auf (C,C), (C,C), (C,D): | C |
| 3. Bit: Antwort auf (C,C), (C,C), (D,C): | C |
| ⋮ | ⋮ |
| 64. Bit: Antwort auf (D,D), (D,D): | D |
- Zusätzlich: 6 Bit zur Kodierung des Spielverlaufs vor dem ersten Zug.
→ jedes Chromosom hat 70 binäre Gene (jeweils C oder D)

Christian Borgelt

Genetische Algorithmen

150

Genetischer Algorithmus: Ablauf

- Initialisierung einer Anfangspopulation mit zufälligen Bitfolgen (70 Bit).
- Aus der aktuellen Population werden Paare von Individuen zufällig ausgewählt. Sie spielen 200-mal das Gefangenendilemma gegeneinander.
 - Für die ersten drei Spiele wird (ein Teil des) im Chromosom abgespeicherten Anfangsspielverlaufs benutzt, um den Zug zu bestimmen. (Die fehlende/zu kurze Historie wird ersetzt/aufgefüllt.)
- Jedes Individuum spielt gegen die gleiche Anzahl von Gegnern. (aus Rechenzeitgründen — 1987! — kein volles Rundenturnier).
- Auswahl von Individuen für die nächste Generation:
 - überdurchschnittliches Ergebnis ($x \geq \mu + \sigma$): 2 Nachkommen
 - durchschnittliches Ergebnis ($\mu - \sigma < x < \mu + \sigma$): 1 Nachkomme
 - unterdurchschnittliches Ergebnis ($\mu - \sigma \geq x$): 0 Nachkommen
- Genetische Operatoren: Standardmutation, Ein-Punkt-Crossover

Christian Borgelt

Genetische Algorithmen

151

Genetischer Algorithmus: Ergebnis

- Die sich ergebenden Strategien sind **Tit-for-Tat** sehr ähnlich.
- [Axelrod 1987] identifizierte die folgenden allgemeinen Muster:
 - **Don't rock the boat.** Kooperiere nach drei Kooperationen. (C,C), (C,C), (C,C) → C
 - **Be provokable.** Spiele Defekt nach plötzlichen Defekt des Gegners. (C,C), (C,C), (C,D) → D
 - **Accept an apology.** Kooperiere, nach wechselseitiger Ausbeutung. (C,C), (C,D), (D,C) → C
 - **Forgel.** (Sei nicht nachtragend.) Kooperiere, nachdem Kooperation nach einer Ausbeutung wiederhergestellt wurde (auch ohne Vergeltung). (C,C), (C,D), (C,C) → C
 - **Accept a rut.** (mit: *fig.* ausgefahrenes Gleis, alter Trost) Spiele Defekt nach dreimaligem Defekt des Gegners. (D,D), (D,D), (D,D) → D

Christian Borgelt

Genetische Algorithmen

152

Das Gefangenendilemma: Erweiterungen

Das Gefangenendilemma läßt sich in verschiedener Weise erweitern, um es realistischer zu machen und weitere Situationen zu erfassen:

- In der Praxis sind die Auswirkungen von Handlungen nicht immer perfekt beobachtbar. Daher: Nicht der genaue Zug des Gegners, sondern nur Wahrscheinlichkeiten sind bekannt.
- Oft sind mehr als zwei Akteure beteiligt: Mehr-Personen-Gefangenendilemma

Ebenso lassen sich die Beschreibungen der Strategien erweitern:

- Berücksichtigung längerer Spielverläufe (mehr als drei Spiele).
- Hinzunahme einer Zufallskomponente für die Wahl des Spielzuges: Wahrscheinlichkeiten für die Wahl von C und D statt eines festen Zuges.

- Beschreibung der Spielstrategie durch Moore-Automaten oder allgemeine Programme, die dann in einem genetischen Algorithmus verändert werden.

Christian Borgelt

Genetische Algorithmen

153

Genetische Programmierung

Christian Borgelt

Genetische Algorithmen

154

Genetische Programmierung

- **Bisher:** Darstellung von Lösungskandidaten / Spielstrategien durch Chromosomen fester Länge (Vektoren von Genen).
- **Jetzt:** Darstellung durch Funktionsausdrücke oder 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)
- 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.

Christian Borgelt

Genetische Algorithmen

155

Genetische Programmierung

- **Beispiel 1:** Erernen 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 gegebenen Daten unter Minimierung der Fehlerquadratsumme — *Methode der kleinsten Quadrate*)
 - $\mathcal{F} = \{+, -, *, /, \sqrt{\quad}, \sin, \cos, \log, \exp, \dots\}$
 - $\mathcal{T} = \{x_1, \dots, x_m\} \cup \mathbb{R}$
- **Beachte:** Alle Chromosomen müssen auswertbar sein.
Daher: Ggf. Vervollständigung des Definitionsbereichs einer Funktion, z.B.
 - $\forall x \leq 0 : \log(x) = 0$ oder = kleinste darstellbare Fließkommazahl
 - $\tan\left(\frac{\pi}{2}\right) = 0$ oder = größte darstellbare Fließkommazahl

Christian Borgelt

Genetische Algorithmen

156

Genetische Programmierung

- 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):
 - Konstanter- und Variablen symbole sind symbolische Ausdrücke.
 - Sind t_1, \dots, t_n symbolische Ausdrücke und ist $f \in \mathcal{F}$ ein (n -stelliges) Funktionsymbol, so ist $(f t_1 \dots t_n)$ 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.

Christian Bergelt

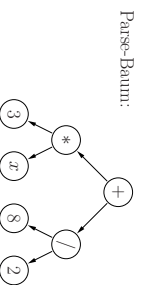
Genetische Algorithmen

157

Genetische Programmierung

- 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: $(+ (* 3 x) (/ 8 2))$



- In Lisp / Scheme werden Ausdrücke durch (verschachtelte) Listen dargestellt:
 - Das erste Element dieser Liste ist das Funktionsymbol bzw. der Operator.
 - Die folgenden Elemente sind die Argumente bzw. Operanden.

Christian Bergelt

Genetische Algorithmen

158

Genetische Programmierung: Ablauf

- 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:
 - Erhitzen Boolescher Funktionen:
 - Anteil korrekter Ausgaben für alle Eingaben bzw. in einer Stichprobe.
- Symbolische Regression:
 - Summe der Fehlerquadrate über die gegebenen Meßpunkte.
 - eindimensional: 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.

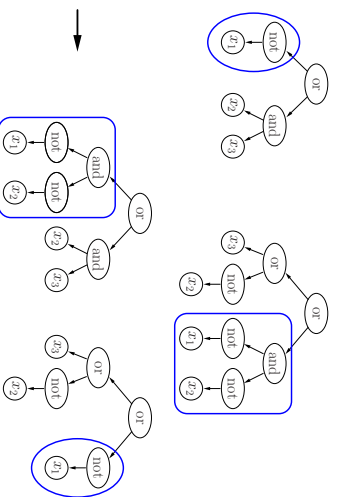
Christian Bergelt

Genetische Algorithmen

159

Genetische Programmierung: Crossover

- Crossover besteht im Austausch zweier Teilausdrücke (Teilbäume)



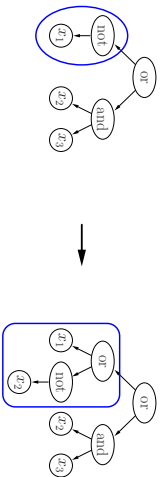
Christian Bergelt

Genetische Algorithmen

160

Genetische Programmierung

- 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 Lösungsandidaten rekombiniert werden können.

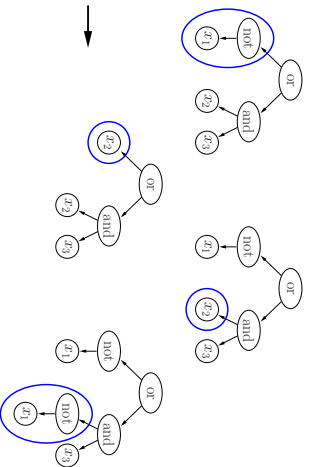
Christian Bergelt

Genetische Algorithmen

161

Genetische Programmierung

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



Christian Bergelt

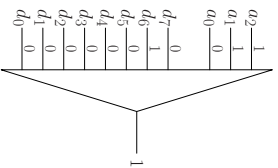
Genetische Algorithmen

162

Genetische Programmierung: 11-Multiplexer

Beispiel: Erlernen eines Booleschen 11-Multiplexers [Koza 1992]

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



Christian Bergelt

Genetische Algorithmen

163

Genetische Programmierung: 11-Multiplexer

Beispiel: Erlernen eines Booleschen 11-Multiplexers [Koza 1992]

- Populationsgröße popsize = 4000
- Anhängstiefe der Parse-Bäume: 6, maximale Tiefe: 17
- Die Fitnesswerte in der Anfangspopulation 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: $(\text{if } a_0 \ d_1 \ d_2)$
- Glücksradauswahl (fitnessproportionale Selektion)
- 90% (3600) der Individuen werden Crossover unterworfen, die restlichen unverändert übernommen.

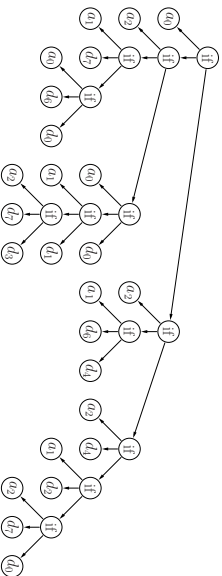
Christian Bergelt

Genetische Algorithmen

164

Genetische Programmierung: 11-Multiplexer

- Nach nur 9 Generationen ist folgende Lösung gefunden (Finnøf 2018):



- Dieses Ergebnis ist erstaunlich einfach und daher nicht plausibel. Es steht zu vermuten, daß die Einzelschritte der genetischen Programmierung übertrieben wird, was sich z.B. durch Auswahl des besten von vielen Läufen erreichen läßt.

Christian Bergelt

Genetische Algorithmen

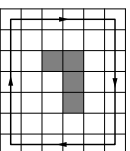
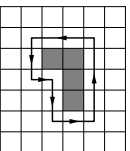
165

Genetische Programmierung: Stimulus-Response-Agent

Beispiel: Erlernen eines Robotersteuerprogramms [Nilsson 1998]

- Betrachte einen Stimulus-Response-Agenten in einer Gitterwelt:
 - 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 den Sensoreingaben, kein Gedächtnis
- Aufgabe:** Umlaufe ein im Raum stehendes Hindernis oder handle die Begegnung des Raumes ab.

s_1	s_2	s_3
s_8	s_4	s_4
s_7	s_6	s_5



Christian Bergelt

Genetische Algorithmen

166

Genetische Programmierung: 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 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.

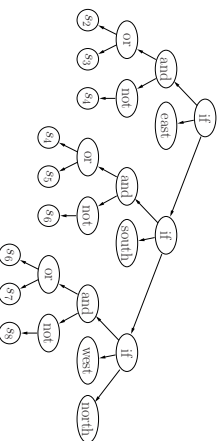
Christian Bergelt

Genetische Algorithmen

167

Genetische Programmierung: 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.

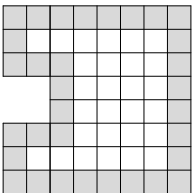
Christian Bergelt

Genetische Algorithmen

168

Genetische Programmierung: Stimulus-Response-Agent

- Bewertung einer Kandidatenlösung anhand eines Testraumes:



- Ein perfekt arbeitendes Steuerprogramm läßt den Agenten die grau gezeichneten Felder abtaufen.
 - 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: $10 \cdot 32 = 320$)

Christian Bergelt

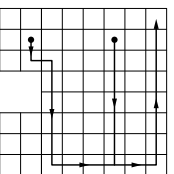
Genetische Algorithmen

169

Genetische Programmierung: Stimulus-Response-Agent

```
(and (not (not (if (if (not s1)
                  (if s4 north east)
                  (if (not 0 south))
                  (or (if s1 s3 s8) (not s7)))
                    (not (not north))))))
(if (or (not (not (and (if s7 north s3)
                      (and south 1)))
        (or (not s6) (or s4 s4))
          (and (if west s3 s5)
               (if 1 s4 s4))))))
(or (not (and (not s3)
              (if east s6 s2)))
    (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 s9))))))
```

Bestes Individuum
in Generation 0:



(Bewegung von zwei
Startpunkten aus)

Christian Bergelt

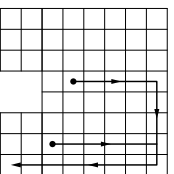
Genetische Algorithmen

170

Genetische Programmierung: Stimulus-Response-Agent

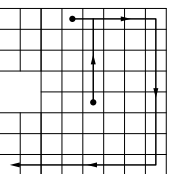
Bestes Individuum in Generation 2:

```
(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)))
```



Christian Bergelt

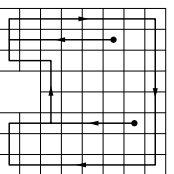
Genetische Algorithmen

171

Genetische Programmierung: Stimulus-Response-Agent

Bestes Individuum in Generation 10:

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



- Dieses Ergebnis ist erstaunlich einfach und daher nicht plausibel. Es steht zu vermuten, daß die Ergebnisse der genetischen Programmierung übertrieben wird, was sich z.B. durch Auswahl des besten von vielen Läufen erreichen läßt.

Christian Bergelt

Genetische Algorithmen

172

Mit dem Programm von der Vorlesungsseite gefundene Lösung:

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

Christian Bergelt

Genetische Algorithmen

175

Genetische Programmierung: Erweiterungen

• Umformung (Editing)

- Diart der Vereinfachung der Chromosomen.
- Ein Chromosom oder Chromosomenteil kann auf komplizierte Weise eine (sehr) einfache Funktion ausdrücken.
- Durch eine Umformung, die logische und/oder arithmetische Äquivalenzen ausnutzt, wird der Ausdruck vereinfacht.

• Kapselung (Encapsulation) / automatisch definierte Funktionen

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

Christian Bergelt

Genetische Algorithmen

175

Evolutionstrategien

Christian Bergelt

Genetische Algorithmen

175

Evolutionstrategien

- Bisher : Behandlung beliebiger (auch diskreter) Optimierungsprobleme
Jetzt : Beschränkung auf **numerische Optimierung**
- Gegeben: Funktion $f : \mathbb{R}^n \rightarrow \mathbb{R}$
Gesucht : Minimum oder Maximum von f
Folglich : Die Chromosomen sind **Vektoren reeller Zahlen**.
- **Mutation** besteht in der Addition eines normalverteilten **Zufallsvektors** \vec{r} . Jedes Element r_i des Zufallsvektors ist die Realisierung einer normalverteilten Zufallsvariable mit
 - Erwartungswert 0 (unabhängig vom Elementindex i) und
 - Varianz σ_i^2 bzw. Standardabweichung σ_i .Die Varianz σ_i^2 kann abhängig oder unabhängig vom Elementindex i und abhängig oder unabhängig von der Generationsnummer t sein.
- Auf **Crossover** (Rekombination zweier Vektoren) wird oft verzichtet.

Christian Bergelt

Genetische Algorithmen

176

Evolutionstrategien: Selektion

- **Strenges Eliteprinzip:**
Nur die besten Individuen kommen in die nächste Generation.
- Bezeichnungen: μ – Anzahl der Individuen in der Elterngeneration
 λ – Anzahl der (durch Mutation) erzeugten Nachkommen
- **Zwei prinzipielle Selektionsstrategien:**
 - **+-Strategie** (Plus-Strategie, $(\mu + \lambda)$ -Strategie)
Aus den $(\mu + \lambda)$ Individuen der Elterngeneration und der erzeugten Nachkommen werden die besten μ Chromosomen ausgewählt.
(Bei dieser Strategie gilt meist $\lambda < \mu$.)
 - **-Strategie** (Komma-Strategie, (μ, λ) -Strategie)
Es werden $\lambda > \mu$ Nachkommen erzeugt,
aus denen die besten μ Chromosomen ausgewählt werden.
(Die Chromosomen der Elterngeneration gehen auf jeden Fall verloren.)

Christian Borgelt

Genetische Algorithmen

177

Evolutionstrategien: Selektion

- **Beispiel:** Spezialfall der $(1+1)$ -Strategie
 - Anfangs, population“: \vec{x}_0 (zufällig erzeugter Vektor reeller Zahlen)
 - Erzeugen der nächsten Generation:
Erzeuge einen reellen Zufallsvektor \vec{r}_t und berechne $\vec{x}_t^* = \vec{x}_t + \vec{r}_t$.
Setze dann
$$\vec{x}_{t+1} = \begin{cases} \vec{x}_t^*, & \text{falls } f(\vec{x}_t^*) \geq f(\vec{x}_t) \\ \vec{x}_t, & \text{sonst.} \end{cases}$$
 - Erzeuge weitere Generationen bis ein Abbruchkriterium erfüllt ist.
- Dies entspricht offenbar dem am Anfang der Vorlesung besprochenen **Zufallsaufstieg**.
- Die allgemeine +-Strategie kann folglich als paralleler Zufallsaufstieg gesehen werden, der gleichzeitig an mehreren Orten des Suchraums durchgeführt wird, wobei stets die erfolgversprechendsten μ Wege verfolgt werden.

Christian Borgelt

Genetische Algorithmen

178

Evolutionstrategien

- Zur Erinnerung (aus der Liste der Prinzipien der organismischen Evolution):
Evolutionstrategische Prinzipien
Optimiert werden nicht nur die Organismen, sondern auch die Mechanismen der Evolution: Vermehrungs- und Sterberaten, Lebensdauern, Anfälligkeit gegenüber Mutationen, Mutationschrittweiten, Evolutionsgeschwindigkeit etc.
- Hier: Anpassung der Varianz des Zufallsvektors (Mutationschrittweite)
 - Geringe Varianz → kleine Änderungen an den Chromosomen
→ lokale Suche (Ausbeutung)
 - Hohe Varianz → große Änderungen an den Chromosomen
→ globale Suche (Durchforstung)
- Weitere Möglichkeiten, Parameter des Evolutionsprozesses anzupassen:
 - Wahl der Zahl der zu ändernden Gene (Vektorelemente).
 - Wahl der Zahl λ der zu erzeugenden Nachkommen.

Christian Borgelt

Genetische Algorithmen

179

Evolutionstrategien: Varianz Anpassung

- **Globale Varianz Anpassung** (chromosomenunabhängige Varianz)
 - Idee: Wähle σ^2 bzw. σ so, daß die mittlere Konvergenzrate möglichst hoch ist.
 - Ansatz von [Rechenberg 1973]: Bestimme optimale Varianz σ für
 - $f_1(x_1, \dots, x_n) = a + bx_1$ und
 - $f_2(x_1, \dots, x_n) = \sum_{i=1}^n a_i^2 x_i^2$,indem die Wahrscheinlichkeiten für eine erfolgreiche (d.h. verbesserte) Mutation bestimmt werden. Diese sind
 - für f_1 : $p_1 \approx 0.184$ und
 - für f_2 : $p_2 \approx 0.270$.
- Aus diesem Ergebnis wurde heuristisch die **$\frac{1}{\sqrt{2}}$ -Erfolgsregel** abgeleitet:
In der +-Strategie ist die Mutationschrittweite richtig, wenn etwa $\frac{1}{\sqrt{2}}$ der Nachkommen besser sind als die Eltern.

Christian Borgelt

Genetische Algorithmen

180

Evolutionstrategien: Varianz Anpassung

Globale Varianz Anpassung (chromosomenunabhängige Varianz)

- Anpassung der Varianz σ^2 auf der Grundlage der $\frac{1}{T}$ Erfolgsregel:
 - Sind mehr als $\frac{1}{2}$ der Nachkommen besser als die Eltern, vergrößere die Varianz/Standardabweichung

$$\sigma^{(\text{neu})} = \sigma^{(\text{alt})} \cdot c_{\text{succ}} \quad c_{\text{succ}} \approx 1.22$$

- Sind weniger als $\frac{1}{2}$ der Nachkommen besser als die Eltern, verkleinere die Varianz/Standardabweichung.

$$\sigma^{(\text{neu})} = \sigma^{(\text{alt})} \cdot c_{\text{dec}} \quad c_{\text{dec}} = \frac{1}{c_{\text{succ}}} \approx 0.83$$

- Bei größeren Populationen ist die $\frac{1}{T}$ Erfolgsregel z.T. zu optimistisch.
- Man kann analog zum **simulierten Ausgühten** auch eine Funktion definieren, mit der die Varianz im Laufe der Generationen verkleinert wird.

Christina Bergelt

Genetische Algorithmen

181

Evolutionstrategien: Varianz Anpassung

Lokale Varianz Anpassung (chromosomenspezifische Varianz)

- Die Varianz/Standardabweichung wird in die Chromosomen aufgenommen:
 - Eine Varianz für alle Vektorelemente oder
 - eine individuelle Varianz für jedes Vektorelement (doppelte Vektorlänge)

- **Beachte:** Die zusätzlichen Vektorelemente für die Varianz(en) haben *keinen direkten Einfluss* auf die Fitness eines Chromosoms.

- **Erwartung:** Chromosomen mit „schlechteren“ Varianzen, d.h.
 - zu klein: Chromosomen entwickeln sich nicht schnell genug weiter oder
 - zu groß: Chromosomen entfernen sich zu weit von ihren Eltern, erzeugen vergleichsweise mehr „schlechte“ Nachkommen.Dadurch sterben ihre Gene (und damit auch ihre Varianzen) leichter aus.

Christina Bergelt

Genetische Algorithmen

182

Evolutionstrategien: Varianz Anpassung

Lokale Varianz Anpassung (chromosomenspezifische Varianz)

- Die elementenspezifischen Mutationschrittweiten (Standardabweichungen) werden nach dem folgenden Schema mutiert:

$$\sigma_i^{(\text{neu})} = \sigma_i^{(\text{alt})} \cdot \exp(r_1 \cdot N_i(0,1) + r_2 \cdot N_i^2(0,1)).$$

- $N_i(0,1)$: einmal je Chromosom zu bestimmende normalverteilte Zufallszahl
- $N_i^2(0,1)$: für jedes Element/Gen zu bestimmende normalverteilte Zufallszahl
- Empfohlene Werte für die Parameter r_1 und r_2 sind [Bäck und Schwefel 1993]

$$r_1 = \frac{1}{\sqrt{2n}}, \quad r_2 = \frac{1}{\sqrt{2}\sqrt{n}},$$

wobei n die Anzahl der Vektorelemente ist, oder [Nissen 1997]

$$r_1 = 0.1, \quad r_2 = 0.2$$

- Oft wird eine untere Schranke für die Mutationschrittweiten festgelegt

Christina Bergelt

Genetische Algorithmen

183

Evolutionstrategien: Varianz Anpassung

Erweiterungen der lokalen Varianz Anpassung

- In der Standardform der lokalen Varianz Anpassung sind die Varianzen der verschiedenen Vektorelemente unabhängig voneinander.
(Format: Die Kovarianzmatrix ist eine Diagonalmatrix.)

- Sollen die Variationen eines Chromosoms bevorzugt in bestimmten Richtungen erzeugt werden, so kann dies mit Einzelvarianzen nur ausgedrückt werden, wenn diese Richtungen achsenparallel sind.

- **Beispiel:** Variationen von Chromosomen mit zwei Genen sollen bevorzugt in Richtung der Hauptdiagonale, d.h. in Richtung $(1,1)$, erzeugt werden. Dies kann mit Einzelvarianzen nicht beschrieben werden.

- **Lösung:** Benutze eine Kovarianzmatrix mit hoher Kovarianz, z.B.

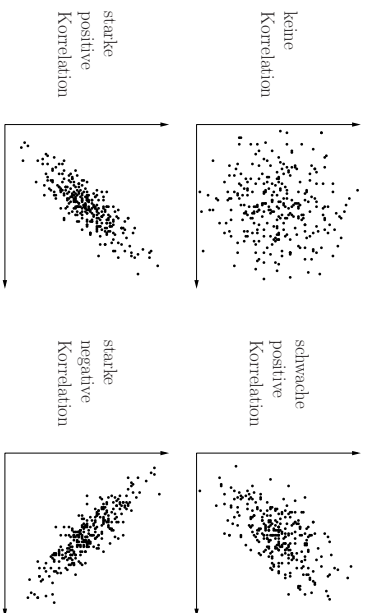
$$\Sigma = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1 \end{pmatrix}.$$

Christina Bergelt

Genetische Algorithmen

184

Kovarianz und Korrelation



Christian Bergth

Generelle Algorithmen

186

Cholesky-Zerlegung

- Anschaulich: **Berechne ein Analogon der Standardabweichung**
- Sei **S** eine symmetrische, positiv definite Matrix (d.h. eine Kovarianzmatrix). Die Cholesky-Zerlegung dient dazu, eine „Quadratwurzel“ von **S** zu berechnen.
 - symmetrisch: $\forall i, j \leq m : s_{ij} = s_{ji}$
 - positiv definit: für alle m -dimensionalen Vektoren $\vec{v} \neq \vec{0}$ gilt $\vec{v}^T \mathbf{S} \vec{v} > 0$
- Formal: Berechne eine linke/untere Dreiecksmatrix **L**, so daß $\mathbf{L}\mathbf{L}^T = \mathbf{S}$. (**L**^T ist die Transponierte der Matrix **L**.)

$$l_{ii} = \left(s_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)^{\frac{1}{2}}$$

$$l_{ji} = \frac{1}{l_{ii}} \left(s_{ij} - \sum_{k=1}^{i-1} l_{ik} l_{jk} \right), \quad j = i+1, i+2, \dots, m.$$

Christian Bergth

Generelle Algorithmen

186

Cholesky-Zerlegung

Spezialfall: Zwei Dimensionen

- Kovarianzmatrix
- Cholesky-Zerlegung

$$\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy} \\ \sigma_{xy} & \sigma_y^2 \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} \sigma_x & 0 \\ \frac{\sigma_{xy}}{\sigma_x} & \frac{1}{\sigma_x} \sqrt{\sigma_x^2 \sigma_y^2 - \sigma_{xy}^2} \end{pmatrix}$$

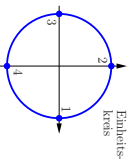
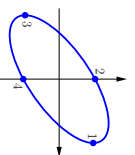


Abbildung mit **L**



Christian Bergth

Generelle Algorithmen

187

Eigenwertzerlegung

- Liefert auch ein **Analogon der Standardabweichung**.
- Rechenaufwendiger als die Cholesky-Zerlegung.
- Sei **S** eine symmetrische, positiv definite Matrix (d.h. eine Kovarianzmatrix).
 - **S** kann geschrieben werden als

$$\mathbf{S} = \mathbf{R} \operatorname{diag}(\lambda_1, \dots, \lambda_m) \mathbf{R}^{-1},$$

wobei die $\lambda_j, j = 1, \dots, m$, die Eigenwerte von **S** und die Spalten von **R** die (normierten) Eigenvektoren von **S** sind.

◦ Die Eigenwerte $\lambda_j, j = 1, \dots, m$, von **S** sind alle positiv und die Eigenvektoren von **S** sind orthonormal ($\rightarrow \mathbf{R}^{-1} = \mathbf{R}^T$).

- Folglich kann **S** geschrieben werden als $\mathbf{S} = \mathbf{T}\mathbf{T}^T$ mit
- $$\mathbf{T} = \mathbf{R} \operatorname{diag} \left(\sqrt{\lambda_1}, \dots, \sqrt{\lambda_m} \right)$$

Christian Bergth

Generelle Algorithmen

188

Evolutionstrategien: Plus- versus Komma-Strategien

- Offensichtlicher **Vorteil** der +-Strategie:
 - Es treten, wegen des strengen Elitprinzipis, nur Verbesserungen auf.
- **Nachteile:**
 - Gefahr des Hängenbleibens in lokalen Minima.
 - Für eine $(\mu + \lambda)$ -Evolutionstrategie mit $\frac{\lambda}{\mu} \geq$ „beste Wahrscheinlichkeit für eine erfolgreiche Mutation“ ($\approx \frac{1}{2}$) haben die Chromosomen einen Selektionsvorteil, die ihre Varianz σ^2 möglichst klein halten, da nicht genügend große Mutationen durchgeführt werden, um eine „echte“ Verbesserung zu erreichen („Beinahe-Stagnation“).
Übliche Wahl des Verhältnisses von μ und λ : etwa 1:7.
- Wenn über mehrere Generationen keine Verbesserungen aufgetreten sind, wird oft von der +-Strategie vorübergehend auf die -Strategie umgeschaltet, um die Diversität in der Population wieder zu erhöhen.

Christian Borgelt

Genetische Algorithmen

105

Mehrkriterienoptimierung

Mehrkriterienoptimierung

- In vielen Alltagsproblemen wird nicht eine einzelne Größe optimiert, sondern **verschiedene Ziele** sollen zu möglichst hohem Grad erreicht werden.
- **Beispiel:** Beim Autokauf wünschst man sich
 - einen niedrigen Preis,
 - einen geringen Kraftstoffverbrauch,
 - möglichst viel Komfort wie elektrische Fensterheber oder eine Klimaanlage.
- Die verschiedenen, zu erreichenden Ziele sind oft nicht unabhängig, sondern **gegenseitlich**: Sie können nicht alle gleichzeitig voll erreicht werden.
- **Beispiel:** Autokauf
 - Für viele Ausstattungsmerkmale ist ein Aufpreis zu zahlen.
 - Die Wahl z.B. einer Klimaanlage oder einfach ein geräumigeres Auto bedingen oft einen etwas leistungsfähigeren Motor und damit einen höheren Preis und Kraftstoffverbrauch.

Christian Borgelt

Genetische Algorithmen

109

Mehrkriterienoptimierung

- **Formale Beschreibung:** Es sind k Kriterien gegeben, denen jeweils eine zu optimierende Zielfunktion zugeordnet ist:
$$f_i : S \rightarrow \mathbb{R}, \quad i = 1, \dots, k$$
- **Einfachster Lösungsansatz:** Fasse die k Zielfunktionen zu einer Gesamtzielfunktion zusammen, z.B. durch Bildung einer gewichteten Summe

$$f(s) = \sum_{i=1}^k w_i f_i(s).$$

- **Wahl der Gewichte:**
 - **Vorzeichen:** Soll die Gesamtzielfunktion zu maximieren sein, so müssen die Vorzeichen der Gewichte w_i der Zielfunktionen f_i , die zu maximieren sind, positiv, die der Gewichte der übrigen Zielfunktionen negativ sein.
 - **Absolutwert:** Mit den Absolutwerten der Gewichte wird die relative Wichtigkeit der Kriterien ausgedrückt (Schwankungsbreite berücksichtigt).

Christian Borgelt

Genetische Algorithmen

109

Christian Borgelt

Genetische Algorithmen

106

Mehrkriterienoptimierung

- **Probleme** des Ansatzes mit einer gewichteten Summe der Zielfunktionen:
 - Man muß bereits vor Beginn der Suche festlegen, welche relative Wichtigkeit die verschiedenen Kriterien haben.
 - Es ist nicht immer einfach, die Gewichte so zu wählen, daß die Präferenzen zwischen den Kriterien angemessen wiedergegeben werden.
- Die Probleme, die mit einer Linearkombination der Zielfunktionen auftreten, sind jedoch noch viel fundamentaler:
 - Allgemein stellt sich das Problem der **Aggregation von Präferenzordnungen**.
 - Dieses Problem tritt auch bei Personenauswahl auf.
(Die Kandidatpräferenzen der Wähler müssen zusammengefaßt werden.)
 - **Arrow'sches Paradoxon** [Arrow 1951]:
Es gibt keine Wahlfunktion, die alle wünschenswerten Eigenschaften hat.

Christiane Bergelt

Genetische Algorithmen

197

Mehrkriterienoptimierung

- Die Arrow'schen Ummöglichkeitssätze [Arrow 1951] lassen sich im Prinzip durch Verwendung **skalierter Präferenzordnungen** umgehen.
- **Aber:** Die Skalierung der Präferenzordnung ist ein weiterer Freiheitsgrad. Es ist n. U. noch schwieriger, eine passende Skalierung zu finden, als die Gewichte einer Linearkombination angemessen zu bestimmen.
- **Alternativer Ansatz:**
Versuche, alle bzw. möglichst viele **Pareto-optimale** Lösungen zu finden.
- Ein Element s des Suchraums S heißt **Pareto-optimal** bezüglich der Zielfunktionen $f_i, i = 1, \dots, k$, wenn es *kein* Element $s' \in S$ gibt, für das gilt
$$\forall i, 1 \leq i \leq k : f_i(s') \geq f_i(s) \quad \text{und}$$
$$\exists j, 1 \leq j \leq k : f_j(s') > f_j(s).$$
- **Anschaulich:** Der Wert keiner Zielfunktion kann verbessert werden, ohne den Wert einer anderen zu verschlechtern.

Christiane Bergelt

Genetische Algorithmen

198

Mehrkriterienoptimierung

- Ausführlichere Definition des Begriffs „Pareto-optimal“:
 - Ein Element $s_1 \in S$ **dominiert** ein Element $s_2 \in S$, wenn gilt
$$\forall i, 1 \leq i \leq k : f_i(s_1) \geq f_i(s_2),$$
 - Ein Element $s_1 \in S$ **dominiert** ein Element $s_2 \in S$ **echt**, wenn s_1 s_2 dominiert und außerdem gilt
$$\exists k, 1 \leq k \leq k : f_k(s_1) > f_k(s_2).$$
 - Ein Element $s_1 \in S$ heißt **Pareto-optimal**, wenn es von keinem Element $s_2 \in S$ echt dominiert wird.
- **Vorteile** der Suche nach Pareto-optimalen Lösungen:
 - Die Zielfunktionen müssen nicht zusammengefaßt werden, die Bestimmung von Gewichten entfällt.
 - Die Suche muß auch für verschiedene Präferenzen nur einmal durchgeführt werden, da erst anschließend aus den gefundenen Lösungen gewählt wird.

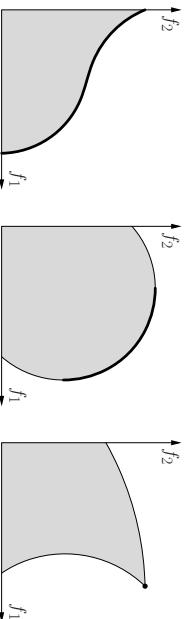
Christiane Bergelt

Genetische Algorithmen

199

Mehrkriterienoptimierung

Veranschaulichung Pareto-optimaler Lösungen



- Alle Punkte des Suchraums liegen im grau gezeichneten Bereich.
- Pareto-optimale Lösungen liegen auf dem fett gezeichneten Teil des Randes.
- Man beachte, daß je nach Lage der Lösungsranddaten die Pareto-optimale Lösung auch eindeutig bestimmt sein kann (siehe rechtes Diagramm).

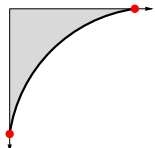
Christiane Bergelt

Genetische Algorithmen

200

Mehrkriterienoptimierung mit genetischen Algorithmen

- **Einfachster Ansatz:** Verwende eine gewichtete Summe der einzelnen Zielfunktionen als Fitneßfunktion. (Dies hat die erwarteten Nachteile.)
 - **Bemerkung:** Dieser Ansatz führt zu einer Pareto-optimalen Lösung, aber eben durch die Gewichtungen ausgerechnet.
- **Naheliegende Alternative:** Gegeben seien k Kriterien, denen die Zielfunktionen $f_1, 1, \dots, k$ zugeordnet sind.
 - $\forall i, 1, \dots, k$: Wähle $\frac{\text{popsize}}{k}$ Individuen auf Grundlage der Fitneßfunktion f_i .
 - **Vorteil:** einfach, geringer Rechenaufwand
 - **Nachteil:** Lösungen, die alle Kriterien recht gut, aber keines maximal erfüllen, haben einen deutlich Selektionsnachteil.
 - **Folge:** Suche konzentriert sich auf Randlösungen.



Christian Borgelt

Genetische Algorithmen

201

Mehrkriterienoptimierung mit genetischen Algorithmen

- **Besserer Ansatz:** Nutze Dominanzbegriff zur Selektion.
- Aufbau einer **Rangskala** der Individuen einer Population:
 - Finde alle nicht dominierten Lösungskandidaten der Population.
 - Ordne diesen Lösungskandidaten den höchsten Rang zu und entferne sie aus der Population.
 - Wiederhole das Bestimmen und Entfernen der nicht dominierten Lösungskandidaten für die weiteren Ränge, bis die Population leer ist.
- Führe mit Hilfe der Rangskala eine **Rangauswahl** durch.
- Meist wird der Ansatz mit **Nischentechniken** kombiniert (um zwischen Individuen mit gleichem Rang zu unterscheiden).
- **Alternative: Turnierauswahl**, wobei der Turniersieger über den Dominanzbegriff und ggf. Nischentechniken bestimmt wird.

Christian Borgelt

Genetische Algorithmen

202

Parallelisierung genetischer Algorithmen

Christian Borgelt

Genetische Algorithmen

203

Parallelisierung genetischer Algorithmen

- Genetische Algorithmen sind recht **teure Optimierungsverfahren**, da oft
 - mit einer großen Population (einige tausend bis einige zehntausend Individuen)
 - mit einer großen Zahl an Generationen (einige hundert) gearbeitet werden muß, um eine hinreichende Lösungsgröße zu erreichen.
- Dieser Nachteil wird zwar durch eine oft etwas höhere Lösungsgröße im Vergleich zu anderen Verfahren wettgemacht, trotzdem kann die Laufzeit eines genetischen Algorithmus unangenehm lang sein.
- Möglicher Lösungsansatz: **Parallelisierung**, d.h. die Verteilung der notwendigen Operationen auf mehrere Prozessoren.
 - **Frage:** ◦ **Welche Schritte kann man parallelisieren?**
 - **Welche vorteilhaften Techniken kann man bei der Parallelisierung zusätzlich anwenden?**

Christian Borgelt

Genetische Algorithmen

204

Parallelisierung: Was kann man parallelisieren?

- **Erzeugen der Anfangspopulation**
Meist problemlos parallelisierbar, da i.a. die Chromosomen der Anfangspopulation zufällig und unabhängig voneinander erzeugt werden.
Der Versuch, Duplikate zu vermeiden, kann die Parallelisierung behindern.
Die Parallelisierung dieses Schrittes hat eher geringe Bedeutung, da eine Anfangspopulation nur einmal erzeugt wird.
- **Bewertung der Chromosomen**
Problemlos parallelisierbar, da die Chromosomen i.a. unabhängig voneinander bewertet werden (Fitneß hängt nur vom Chromosom selbst ab).
Auch beim Gefangenendilemma können Paarungen parallel bearbeitet werden.
- **Berechnung der (relativen) Fitneßwerte**
Zur Berechnung der relativen Fitneß oder einer Rangordnung der Chromosomen müssen die Bewertungen zusammengeführt werden.

Christina Bergelt

Genetische Algorithmen

205

Parallelisierung: Was kann man parallelisieren?

- **Selektion**
Ob sich der Auswahlsschritt parallelisieren läßt, hängt sehr stark vom verwendeten Selektionsverfahren ab:
 - **Erwartungswertmodell und Elitismus**
Erfordern beide eine globale Betrachtung der Population und sind daher nur schwer parallelisierbar.
 - **Glücksrad- und Rangauswahl**
Nachdem die relativen Fitneßwerte bzw. die Ränge bestimmt sind (was schwer parallelisierbar ist), ist die Auswahl selbst leicht parallelisierbar.
 - **Turnierauswahl**
Ideal für die Parallelisierung geeignet, besonders bei kleinen Turniergrößen, da keine globale Information bestimmt werden muß, sondern sich der Vergleich der Fitneßwerte auf die Individuen des Turniers beschränkt.

Christina Bergelt

Genetische Algorithmen

206

Parallelisierung: Was kann man parallelisieren?

- **Anwendung genetischer Operatoren**
Leicht zu parallelisieren, da jeweils nur ein (Mutation) oder zwei Chromosomen (Crossover) betroffen sind. Zusammen mit einer Turnierauswahl kann ein Steady-State genetischer Algorithmus daher sehr gut parallelisiert werden.
- **Abbruchbedingung**
Der einfache Test, ob eine bestimmte Generationenzahl erreicht ist, bereitet bei einer Parallelisierung keine Probleme.
Abbruchkriterien wie
 - das beste Individuum der Population hat eine bestimmte Mindestgröße oder
 - über eine bestimmte Anzahl von Generationen hat sich das beste Individuum nicht/kaum verbessertsind dagegen für eine Parallelisierung weniger geeignet, da sie eine globale Betrachtung der Population erfordern.

Christina Bergelt

Genetische Algorithmen

207

Parallelisierung: Inselmodell und Migration

- Auch wenn z.B. ein schwer zu parallelisierendes Selektionsverfahren verwendet wird, kann man eine Parallelisierung erreichen, indem mehrere unabhängige Populationen parallel berechnet werden.
Jede Population wird als eine Insel bewohnend angesehen, daher **Inselmodell**.
- Das reine Inselmodell ist äquivalent zu einer mehrfachen seriellen Ausführung des gleichen genetischen Algorithmus. Es liefert meist etwas schlechtere Ergebnisse als ein einzelner Lauf mit entsprechend größerer Population.
- Zwischen den Inselpopulationen können zu festgelegten Zeitpunkten (*nicht* in jeder Generation) Individuen ausgetauscht werden.
Man spricht in diesem Fall von **Migration** (Wanderung).
- Es findet normalerweise keine direkte Rekombination von Chromosomen statt, die von verschiedenen Inseln stammen. Erst nach einer Migration wird genetische Information von einer Insel mit der einer anderen Insel kombiniert.

Christina Bergelt

Genetische Algorithmen

208

Parallelisierung: Inselmodell und Migration

- **Steuerung der Migration zwischen Inseln**
 - **Zufallsmodell**
Die beiden Inseln, zwischen denen Individuen ausgetauscht werden sollen, werden zufällig bestimmt. Beliebige Inseln können Individuen austauschen.
 - **Netzwerkmodell**
Die Inseln werden in einem Graphen angeordnet. Individuen können zwischen den Inseln nur entlang der Kanten des Graphen wandern. Die Kanten, über die Individuen ausgetauscht werden sollen, werden zufällig bestimmt.
- **Wettbewerb zwischen den Inseln**
 - Die genetischen Algorithmen, die auf den Inseln angewandt werden, unterscheiden sich (in Verfahren und/oder Parametern).
 - Die Populationsgröße einer Insel wird entsprechend ihrer durchschnittlichen Fitness der Individuen erhöht oder erniedrigt. Es gibt jedoch eine Mindestpopulationsgröße, die nicht unterschritten werden darf.

Christian Borgelt

Genetische Algorithmen

209

Parallelisierung: Zellulare genetische Algorithmen

(auch: “isolation by distance”)

- Die Prozessoren werden in einem (rechthwinkligen) Gitter angeordnet. Das Gitter bedeckt gewöhnlich die Oberfläche eines Torns.
- Selektion und Crossover werden auf im Gitter benachbarte (durch Kanten verbundene), Mutation auf einzelne Prozessoren beschränkt.
- Beispiel: Jeder Prozessor verwaltet ein Chromosom.
 - **Selektion:** Ein Prozessor wählt das beste Chromosom seiner (vier) Nachbarprozessoren oder eines dieser Chromosomen zufällig nach ihrer Fitness.
 - **Crossover:** Der Prozessor führt Crossover mit dem gewählten und dem eigenen Chromosom durch oder mutiert sein Chromosom. Er behält den besseren der beiden Nachkommen bzw. von Elter und Kind.
- Es bilden sich Gruppen benachbarter Prozessoren, die ähnliche Chromosomen verwalten. Dies mildert die oft zerstörende Wirkung des Crossover.

Christian Borgelt

Genetische Algorithmen

210

Parallelisierung: Mühlenbeins Ansatz

- **Kombination von genetischen Algorithmen mit Zufallsaufstieg:**
Jedes Individuum führt lokalen Zufallsaufstieg durch, d.h.
 - bei einer vorteilhafteren Mutation wird der Elter ersetzt,
 - bei einer nachteiligen Mutation bleibt der Elter erhalten.Dieser Zufallsaufstieg kann leicht parallelisiert werden.
- Individuen suchen sich einen Crossover-Partner in ihrer Nachbarschaft. (Dazu wird eine Definition des Abstandes zweier Individuen benötigt — vergleiche die Nischentechniken, z.B. *power law starting*)
- Die Nachkommen (Crossover-Produkte) führen lokalen Zufallsaufstieg durch.
- Die Individuen der nächsten Generation werden nach dem „**lokalen**“ **Elite-prinzip** ausgewählt, d.h., die beiden besten Individuen unter den Eltern und den optimierten Nachkommen werden übernommen.

Christian Borgelt

Genetische Algorithmen

211

Anwendungsbeispiele

1. Flugroutenplanung: ROGENA

Christian Borgelt

Genetische Algorithmen

212

Problemstellung: Flugrouteplanung

- Flugzeuge bewegen sich im Luftraum normalerweise
 - auf Standardrouten zwischen den Flughäfen,
 - mit einem vorgeschriebenen Mindestabstand auf der gleichen Route,
 - abhängig von ihrer Flugrichtung auf unterschiedlichen Höhen.
 - **Vorteile** dieser Lösung:
 - einfache Regeln/Vorschriften für den Flugverkehr,
 - leichte Kontrolle der Flugrouten durch die Fluglotsen,
 - große Sicherheit im Flugverkehr (i.w. Kollisionsvermeidung).
 - **Nachteile** dieser Lösung:
 - nur ein relativ kleiner Teil des Luftraums wird genutzt,
 - die Flugzeugdichte in der Nähe von Flughäfen ist relativ eng begrenzt.
- Diese Lösung ist dem steigenden Flugverkehr nicht mehr angemessen.

Christian Borgelt

Genetische Algorithmen

213

Luftraum in der Umgebung von Frankfurt

picture not available in online version

Christian Borgelt

Genetische Algorithmen

214

Lösungsidee: Fluglotsenunterstützung

- Der Luftraum zwischen den Standardrouten muß genutzt werden, um für zukünftige Luftverkehrsstörungen gewappnet zu sein.
- **Problem** dieses Ansatzes:
 - Sich freier bewegende Flugzeuge sind schwerer zu koordinieren.
 - Dadurch steigt die Arbeitsbelastung für die Fluglotsen.
- **Aufgabenstellung:** Entwicklung eines Unterstützungswerkzeugs, das die Konstruktion sicherer und effizienter Routen zwischen Eintritts- und Austrittspunkt in dem vom Lotsen kontrollierten Teil des Luftraumes übernimmt.
- **Vorgegebene Rahmenbedingungen:**
 - Eintrittszeitpunkt, Ein- und Austrittsort im kontrollierten Bereich,
 - Flugeschadfen der kontrollierten Flugzeuge,
 - Bewegung weiterer Flugzeuge (Kollisionsvermeidung),
 - Sperrgebiete (Behaunung, militärische Nutzung, schlechtes Wetter).

Christian Borgelt

Genetische Algorithmen

215

Lösungsansatz mit genetischen Algorithmen

- Der Suchraum für mögliche Flugrouten ist sehr groß; eine analytische Konstruktionsmethode ist schwer zu finden.
→ Suche mit genetischen Algorithmen
- **ROGENA** (free Routing with GENetic Algorithms) [Gardes 1994, 1995]
- Die notwendigen Daten und Berechnungsformeln für die Beschreibung der Flugeigenschaften und des Flugverhaltens von Flugzeugen wurden der BADA-Datenbank der EUROCONTROL entnommen [Byrne 1995].
- Es wurde ein Ausschnitt von 200×200 nautischen Meilen (NM) mit einer Höhe von 0 bis 10000 Fuß (ft) aus dem Luftraum betrachtet.
- Mindestabstände zwischen Flugzeugen: standardmäßig 5 NM, beim Erdaufgang zwischen 2.5 und 6 NM in Abhängigkeit vom Gewicht der beiden Flugzeuge.
- Der genetische Algorithmus beginnt erst bei Flugroutenkonflikten (Unterschreitung der Sicherheitsabstände) zu arbeiten.

Christian Borgelt

Genetische Algorithmen

216

ROGENA: Kodierung der Lösungskandidaten

- Eine Flugroute wird als Sequenz von Linienstücken dargestellt (vgl. Übungsaufgabe zum Finden eines Weges durch ein Gebiet).
- Die Chromosomen haben variable Länge (variable Anzahl von Genen),
 - jedes Gen stellt einen zu überfliegenden Punkt dar; zusätzlich wird die Überfluggeschwindigkeit angegeben:

	x	y	z	v
Gen 1	86.2	15.8	1.65	258
Gen 2	88.9	24.7	1.31	252
⋮	⋮	⋮	⋮	⋮
Gen k	105.0	98.0	0.00	120

- Es müssen Nebenbedingungen eingehalten werden, z.B.
 - monoton fallende Flugroute (da Landeanflüge modelliert werden),
 - maximale Beschleunigung/Verzögerung zwischen Punkten,
 - minimaler Winkel zwischen Linienstücken (Kurvearradius) etc.

Christina Bergelt

Genetische Algorithmen

217

ROGENA: Fineffunktion

- Die Fineffunktion berücksichtigt folgende Eigenschaften einer Flugroute:
 - einzahlhaltender Sicherheitsabstand zu anderen Flugzeugen,
 - kein Durchfliegen von gesperrten Luftraum,
 - Länge der Flugroute bis zur Landebahn,
 - Pünktlichkeit des Fluges (planmäßige Ankunft),
 - möglichst geringe Abweichungen von der optimalen Sinkrate (berechnet nach BADA-Datenbank),
 - keine zu spitzen Winkel zwischen den Linienstücken, um Abweichungen von der tatsächlichen Flugbahn klein zu halten.
- Diese Eigenschaften gehen mit Gewichtungsfaktoren versehen in die Fineffunktion ein. Die Fineffunktion ist zu minimieren.
- Ein Benutzer kann die Gewichtungsfaktoren für die einzelnen Eigenschaften über Schieberegler beeinflussen.

Christina Bergelt

Genetische Algorithmen

218

ROGENA: Ablauf des genetischen Algorithmus

- **Ziel:** Erzeugen einer sicheren und effizienten Flugroute für ein neu in den kontrollierten Luftraum eintretendes Flugzeug.
- **Basis:** modifizierte Form eines genetischen Algorithmus (ein Chromosom wird entweder Crossover oder Mutation unterworfen)
- **Populationsgröße:** 60 Individuen/Chromosomen
- **Initialisierung der Anfangspopulation:** Durch wiederholte zufällige Veränderung der Standardflugroute.
- **Selektionsverfahren:** Im wesentlichen Glücksradauswahl, wobei allerdings Chromosomen, deren Fitness über einen Schwellenwert liegt, nicht in die Berechnung der Selektionswahrscheinlichkeit eingehen (entspricht der Forderung einer Mindestgröße). Der Schwellenwert wird im Laufe der Generationen gesenkt, und zwar in Abhängigkeit vom Fitnessdurchschnitt der Population. (Kombination aus Glücksradauswahl und SimulatedAnnealing)

Christina Bergelt

Genetische Algorithmen

219

ROGENA: Anwendung genetischer Operatoren

- 20 Chromosomen werden unverändert übernommen, darunter die 5 besten Chromosomen (Elitismus).
- 20 Chromosomen werden einem **2-Punkt-Crossover** unterworfen.
- 20 Chromosomen werden einem speziellen **Mutationsoperator** unterworfen:
 - entweder völlig zufällige Koordinatenänderung (globale Suche)
 - oder mittlere Veränderung in einem Punkt „in der Nähe“
 - oder kleine Veränderung in einem Punkt „in der Nähe“ (Zufallsaufstieg).
- Zusätzlich Veränderung der Genanzahl (mit geringer Wahrscheinlichkeit). Der Lösch- oder Einfügeort im Chromosom wird zufällig bestimmt, der neue Punkt zufällig in der Nähe seiner Nachbarn initialisiert.
- **Reparaturmechanismen:** Eine Abfolge von Sink- und Steigvorgängen ist unökonomisch, ebenso eine Abfolge von Beschleunigungs- und Bremsvorgängen.

Christina Bergelt

Genetische Algorithmen

220

ROGENA: 2-Punkt-Crossover

picture not available in online version

Diese Form des Crossover wurde gewählt, um Teilstücke von Routen auszutauschen und auf diese Weise nützliche Teile verschiedener Routen zu kombinieren.

Christian Bergelt

Genetische Algorithmen

221

ROGENA: Benutzeroberfläche

picture not available in online version

Christian Bergelt

Genetische Algorithmen

222

ROGENA: Testläufe und Ergebnisse

- **Benutzeroberfläche:**
 - Originalroute in rot (direkte Verbindung von Eintrittsort und Zielpunkt)
 - beste aktuelle Route in grün
 - alternative Routen in grau
 - Höhen diagramm der Route in getrennten Fenster
 - Links: Zeitleiter mit Landzeiten aller Flugzeuge
- Es wurden mehrere **Simulationsläufe** mit realen Daten durchgeführt (beschreiben Eintrittsort/-zeitpunkt und tatsächliche Flugbahn).
- Die von ROGENA erzeugten Flugrouten sind deutlich kürzer als die tatsächlich- den, ohne daß dadurch Konflikte mit anderen Flugzeugen erzeugt wurden.
- Wirkungsvolle Unterstützung eines Fluglotsen bei der Koordinierung, die Verkürzung der Routen ermöglicht einen höheren Flugzeugdurchsatz.

Christian Bergelt

Genetische Algorithmen

223

Anwendungsbeispiele

2. Erlernen von Fuzzy-Reglern

Christian Bergelt

Genetische Algorithmen

224

Kurzeinführung Fuzzy-Theorie

- **Klassische Logik:** nur Wahrheitswerte *wahr* und *falsch*.
 - **Klassische Mengenlehre:** entweder *ist Element* oder *ist nicht Element*.
 - Die Zwewertigkeit der klassischen Theorien ist oft nicht angemessen.
 - Beispiel zur Illustration: **Sorites-Paradoxon** (griech. *sortes*: Haufen)
 - Eine Milliarde Sandkörner sind ein Sandhaufen. (*wahr*)
 - Wenn man von einem Sandhaufen ein Sandkorn entfernt, bleibt ein Sandhaufen übrig. (*wahr*)
- Es folgt daher:
- 999 999 999 Sandkörner sind ein Sandhaufen. (*wahr*)
 - 999 999 999 Sandkörner sind ein Sandhaufen. (*falsch!*)
- Mehrfache Wiederholung des gleichen Schlusses liefert schließlich
- 1 Sandkorn ist ein Sandhaufen. (*falsch!*)
- Bei welcher Anzahl Sandkörner ist der Schluss nicht wahrheitsbewahrend?

Christian Bergelt

Grundlegende Algorithmen

225

Kurzeinführung Fuzzy-Theorie

- Offenbar: Es gibt keine genau bestimmte Anzahl Sandkörner; bei der der Schluss auf die nächstkleinere Anzahl falsch ist.
- Problem: Begriffe der natürlichen Sprache (z.B. „Sandhaufen“, „kalkköpfig“, „warm“, „schnell“, „hoher Druck“, „leicht“ etc.) sind **vage**.
- Beachte: Vage Begriffe sind zwar *unexakt*, aber trotzdem nicht *unbrauchbar*.
 - Auch für vage Begriffe gibt es Situationen/Objekte, auf die sie *sicher nicht anwendbar* sind und solche auf die sie *sicher nicht anwendbar* sind.
 - Dazwischen liegt eine **Pennombra** (lat. für *Halbschatten*) von Situationen, in denen es unklar ist, ob die Begriffe anwendbar sind, oder in denen sie nur mit Einschränkungen anwendbar sind („kleiner Sandhaufen“).
 - Die Fuzzy-Theorie versucht, diese Pennombra mathematisch zu modellieren („weicher Übergang“ zwischen *anwendbar* und *nicht anwendbar*).

Christian Bergelt

Grundlegende Algorithmen

226

Fuzzy-Logik

- Die **Fuzzy-Logik** ist eine Erweiterung der klassischen Logik um Zwischenwerte zwischen *wahr* und *falsch*.
- Als Wahrheitwert kann jeder Wert aus dem reellen Intervall $[0, 1]$ auftreten, wobei $0 \cong$ *falsch* und $1 \cong$ *wahr*.
- Folglich notwendig: **Erweiterung der logischen Operatoren**

◦ Negation	klassisch: $\neg a$	fuzzy: $\sim a$	Fuzzy-Negation
◦ Konjunktion	klassisch: $a \wedge b$	fuzzy: $T(a, b)$	t -Norm
◦ Disjunktion	klassisch: $a \vee b$	fuzzy: $\perp(a, b)$	t -Konnorm
- **Grundprinzipien** der Erweiterung:
 - Für die Extremwerte 0 und 1 sollen sich die Operationen genauso verhalten wie ihre klassischen Vorbilder (Rand-/Eckbedingungen).
 - Für die Zwischenwerte soll das Verhalten monoton sein.
 - Soweit möglich, sollen die Gesetze der klassischen Logik erhalten werden.

Christian Bergelt

Grundlegende Algorithmen

227

Fuzzy-Negationen

Eine **Fuzzy-Negation** ist eine Funktion $\sim: [0, 1] \rightarrow [0, 1]$, die die folgenden Bedingungen erfüllt:

- $\sim 0 = 1$ und $\sim 1 = 0$ (Randbedingungen)
- $\forall a, b \in [0, 1] : a \leq b \Rightarrow \sim a \geq \sim b$ (Monotonie)

Gelten in der zweiten Bedingung statt \leq und \geq sogar die Beziehungen $<$ und $>$, so spricht man von einer *streikten* Negation.

Weitere Bedingungen, die manchmal gestellt werden, sind:

- \sim ist eine stetige Funktion.
- \sim ist *involutiv*, d.h. $\forall a \in [0, 1] : \sim \sim a = a$.

Involutivität entspricht dem klassischen *Gesetz der Identität* $\neg \neg a = a$.

Die obigen Bedingungen legen die Fuzzy-Negation nicht eindeutig fest.

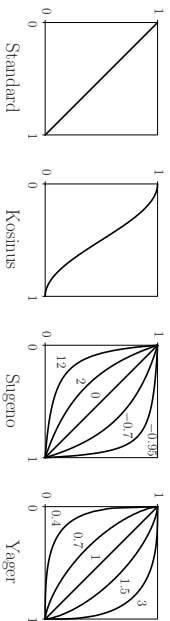
Christian Bergelt

Grundlegende Algorithmen

228

Fuzzy-Negationen

- Standardnegation: $\sim a = 1 - a$
- Schwellenwertnegation: $\sim(a; \theta) = \begin{cases} 1, & \text{falls } x \leq \theta, \\ 0, & \text{sonst.} \end{cases}$
- Kosinusnegation: $\sim a = \frac{1}{2}(1 + \cos \pi a)$
- Sigano-Negation: $\sim(a; \lambda) = \frac{1-a}{1 + \lambda a}$
- Yäger-Negation: $\sim(a; \lambda) = (1 - a^\lambda)^{\frac{1}{\lambda}}$



Christian Bergelt

Generelle Algorithmen

229

t-Normen / Fuzzy-Konjunktionen

Eine **t-Norm** oder **Fuzzy-Konjunktion** ist eine Funktion $T : [0, 1]^2 \rightarrow [0, 1]$, die die folgenden Bedingungen erfüllt:

- $\forall a \in [0, 1] : T(a, 1) = a$ (Randbedingung)
- $\forall a, b, c \in [0, 1] : b \leq c \Rightarrow T(a, b) \leq T(a, c)$ (Monotonie)
- $\forall a, b \in [0, 1] : T(a, b) = T(b, a)$ (Kommutativität)
- $\forall a, b, c \in [0, 1] : T(a, T(b, c)) = T(T(a, b), c)$ (Assoziativität)

Weitere Bedingungen, die manchmal gestellt werden, sind:

- T ist eine stetige Funktion (Stetigkeit)
- $\forall a \in [0, 1] : T(a, a) < a$ (Subidempotenz)
- $\forall a, b, c, d \in [0, 1] : a < b \wedge c < d \Rightarrow T(a, b) < T(c, d)$ (strikte Monotonie)

Die ersten beiden dieser Bedingungen (zusätzlich zu den ersten vier) definieren die Teilklasse der sogenannten *Archimedischen t-Normen*.

Christian Bergelt

Generelle Algorithmen

230

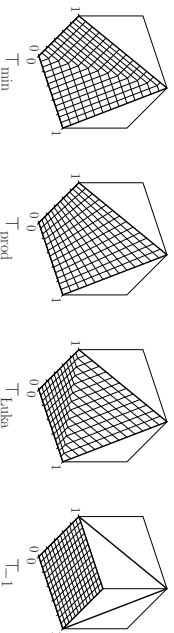
t-Normen / Fuzzy-Konjunktionen

Standardkonjunktion: $T_{\min}(a, b) = \min\{a, b\}$

Algebraisches Product: $T_{\text{prod}}(a, b) = a \cdot b$

Lukasiewicz: $T_{\text{Lukas}}(a, b) = \max\{0, a + b - 1\}$

Drahtisches Product: $T_{-1}(a, b) = \begin{cases} a, & \text{if } b = 1; \\ b, & \text{if } a = 1; \\ 0, & \text{otherwise.} \end{cases}$



Christian Bergelt

Generelle Algorithmen

231

t-Konormen / Fuzzy-Disjunktionen

Eine **t-Konorm** oder **Fuzzy-Disjunktion** ist eine Funktion $\perp : [0, 1]^2 \rightarrow [0, 1]$, die die folgenden Bedingungen erfüllt:

- $\forall a \in [0, 1] : \perp(a, 0) = a$ (Randbedingung)
- $\forall a, b, c \in [0, 1] : b \leq c \Rightarrow \perp(a, b) \leq \perp(a, c)$ (Monotonie)
- $\forall a, b \in [0, 1] : \perp(a, b) = \perp(b, a)$ (Kommutativität)
- $\forall a, b, c \in [0, 1] : \perp(a, \perp(b, c)) = \perp(\perp(a, b), c)$ (Assoziativität)

Weitere Bedingungen, die manchmal gestellt werden, sind:

- \perp ist eine stetige Funktion (Stetigkeit)
- $\forall a \in [0, 1] : \perp(a, a) > a$ (Superidempotenz)
- $\forall a, b, c, d \in [0, 1] : a < b \wedge c < d \Rightarrow \perp(a, b) < \perp(c, d)$ (strikte Monotonie)

Die ersten beiden dieser Bedingungen (zusätzlich zu den ersten vier) definieren die Teilklasse der sogenannten *Archimedischen t-Konormen*.

Christian Bergelt

Generelle Algorithmen

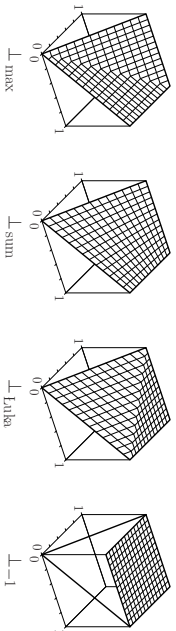
232

Standarddisjunktion: $\perp_{\max}(a, b) = \max\{a, b\}$

Algebraische Summe: $\perp_{\text{sum}}(a, b) = a + b - a \cdot b$

Lukasiewicz: $\perp_{\text{Luka}}(a, b) = \min\{1, a + b\}$

Drahtische Summe: $\perp_{-1}(a, b) = \begin{cases} a, & \text{if } b = 0, \\ b, & \text{if } a = 0, \\ 1, & \text{otherwise.} \end{cases}$



Zusammenspiel der Fuzzy-Operatoren

- Es gilt $\forall a, b \in [0, 1] : \perp_{-1}(a, b) \leq \perp_{\text{Luka}}(a, b) \leq \perp_{\text{prod}}(a, b) \leq \top_{\min}(a, b)$.
Auch alle anderen denkbaren t -Normen liegen zwischen \top_{-1} und \top_{\min} .
- Es gilt $\forall a, b \in [0, 1] : \perp_{\max}(a, b) \leq \perp_{\text{sum}}(a, b) \leq \perp_{\text{Luka}}(a, b) \leq \perp_{-1}(a, b)$.
Auch alle anderen denkbaren t -Konormen liegen zwischen \perp_{\max} und \perp_{-1} .
- Beachte: Es gilt i.a. *weder* $\top(a, \sim a) = 0$ *noch* $\perp(a, \sim a) = 1$.
- Ein Operatorpaar (\sim, \perp) bestehend aus einer Fuzzy-Negation \sim , einer t -Norm \top und einer t -Konorm \perp heißt **duales Tripel**, wenn mit diesen Operatoren die Verallgemeinerungen der DeMorganschen Gesetze gelten, d.h.
 $\forall a, b \in [0, 1] : \sim \top(a, b) = \perp(\sim a, \sim b)$
 $\forall a, b \in [0, 1] : \sim \perp(a, b) = \top(\sim a, \sim b)$
- Der am häufigsten benutzte Operatorpaar ist das duale Tripel $(\sim, \top_{\min}, \perp_{\max})$ mit der Standardnegation $\sim a \equiv 1 - a$.

Fuzzy-Mengenlehre

- Die klassische Mengenlehre basiert auf dem Begriff „*ist Element von*“ (\in).
Alternativ kann man die Zugehörigkeit zu einer Menge mit einer **Indikatorfunktion** beschreiben:
Sei X eine Menge. Dann heißt

$$I_M : X \rightarrow \{0, 1\}, \quad I_M(x) = \begin{cases} 1, & \text{falls } x \in X, \\ 0, & \text{sonst.} \end{cases}$$

Indikatorfunktion der Menge M bzgl. der Grundmenge X .

- In der Fuzzy-Mengenlehre wird die Indikatorfunktion durch eine **Zugehörigkeitsfunktion** ersetzt:
Sei X eine (klassische/scharfe) Menge. Dann heißt

$$\mu_M : X \rightarrow [0, 1], \quad \mu_M(x) \equiv \text{Zugehörigkeitsgrad von } x \text{ zu } M,$$

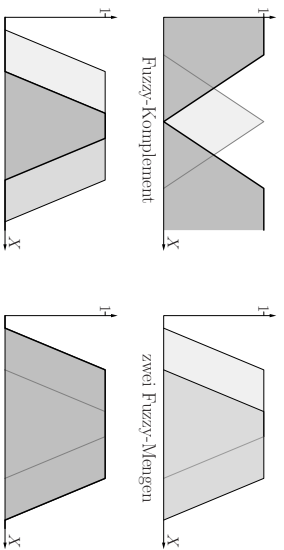
Zugehörigkeitsfunktion (membership function) der **Fuzzy-Menge** M bzgl. der **Grundmenge** X .

Meist wird die Fuzzy-Menge mit ihrer Zugehörigkeitsfunktion identifiziert.

Fuzzy-Mengenlehre: Operationen

- Wie beim Übergang von der klassischen Logik zur Fuzzy-Logik ist beim Übergang von der klassischen Mengenlehre zur Fuzzy-Mengenlehre eine Erweiterung der Operationen nötig.
 - **Grundprinzip dieser Erweiterung:**
Greife auf die logische Definition der Operationen zurück.
 \Rightarrow elementweise Anwendung der logischen Operatoren
 - Seien A und B (Fuzzy-)Mengen über der Grundmenge X .
- Komplement**
- | | |
|-----------|--|
| klassisch | $\bar{A} = \{x \in X \mid x \notin A\}$ |
| fuzzy | $\forall x \in X : \mu_{\bar{A}}(x) = \sim \mu_A(x)$ |
- Schnitt**
- | | |
|-----------|--|
| klassisch | $A \cap B = \{x \in X \mid x \in A \wedge x \in B\}$ |
| fuzzy | $\forall x \in X : \mu_{A \cap B}(x) = \top(\mu_A(x), \mu_B(x))$ |
- Vereinigung**
- | | |
|-----------|---|
| klassisch | $A \cup B = \{x \in X \mid x \in A \vee x \in B\}$ |
| fuzzy | $\forall x \in X : \mu_{A \cup B}(x) = \perp(\mu_A(x), \mu_B(x))$ |

Fuzzy-Mengenoperationen: Beispiele



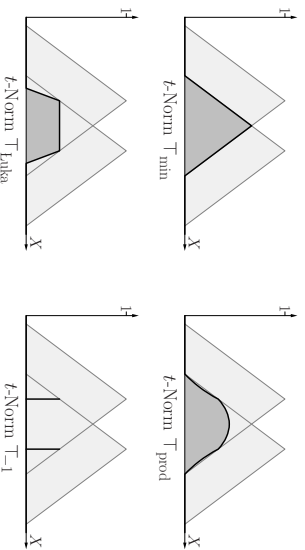
- Der links gezeigte Fuzzy-Schnitt und die rechts gezeigte Fuzzy-Vereinigung sind unabhängig von der gewählten t -Norm bzw. t -Konorm.

Christian Bergelt

Grundlegende Algorithmen

237

Fuzzy-Schnitt: Beispiele



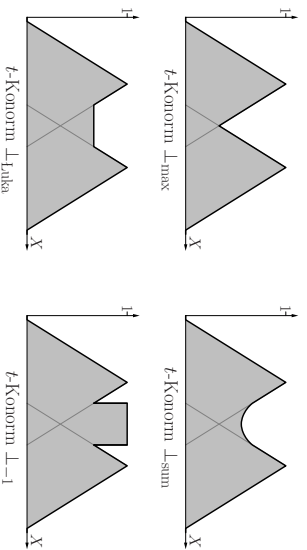
- Man beachte, daß alle Fuzzy-Schnitte zwischen dem oben links und dem unten rechts gezeigten liegen.

Christian Bergelt

Grundlegende Algorithmen

238

Fuzzy-Vereinigung: Beispiele



- Man beachte, daß alle Fuzzy-Vereinigungen zwischen der oben links und der unten rechts gezeigten liegen.

Christian Bergelt

Grundlegende Algorithmen

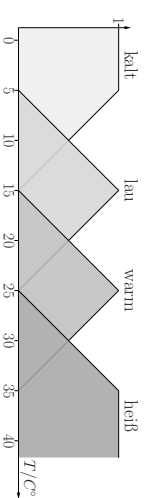
239

Fuzzy-Partitionen und Linguistische Variablen

- Um einen Wertebereich durch sprachliche (linguistische) Ausdrücke beschreiben zu können, wird er mit Hilfe von Fuzzy-Mengen fuzzy-partitioniert. Jeder Fuzzy-Menge der Partitionierung ist ein linguistischer Term zugeordnet.
- Übliche Bedingung: An jedem Punkt müssen sich die Zugehörigkeitsgrade aller Fuzzy-Mengen zu 1 addieren (*partition of unity*).

Beispiel: Fuzzy-Partitionierung für Temperaturen

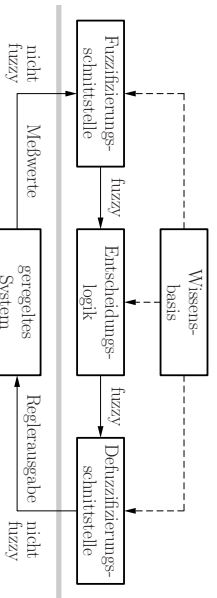
Wir dehnen eine linguistische Variable mit den Werten *kalt*, *lau*, *warm* und *heiß*.



Christian Bergelt

Grundlegende Algorithmen

240



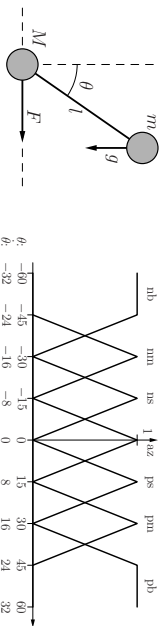
- Die Wissensbasis enthält die Fuzzy-Regeln für die Steuerung und die Fuzzy-Partitionen der Wertebereiche der Variablen.
- Eine Fuzzy-Regel lautet: **if** X_1 **is** $A_1^{(1)}$ **and** ... **and** X_n **is** $A_n^{(n)}$ **then** Y **is** B .
 X_1, \dots, X_n sind die Meßgrößen und Y ist die Stellgröße.
 $A_k^{(k)}$ und B sind linguistische Terme, deren Fuzzy-Mengen zugeordnet sind.

Christian Bergelt

Genetische Algorithmen

211

Beispiel-Fuzzy-Regler: Stabbalance



Abkürzungen

- pb – positive big
- pm – positive medium
- ps – positive small
- az – approximately zero
- ns – negative small
- nm – negative medium
- nb – negative big

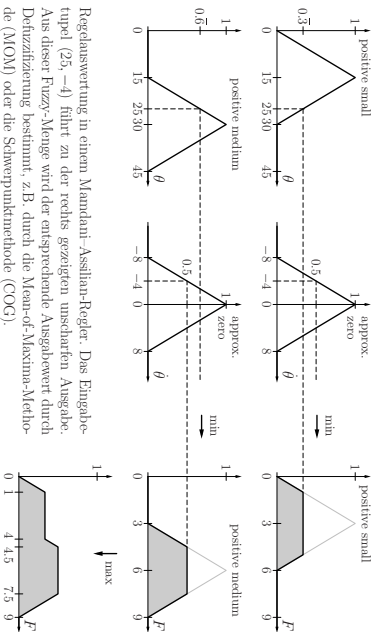
θ/θ	nb	nm	ns	az	ps	pm	pb
pb					ps	pb	
pm					pm		
ps			ps		az	ps	
az		nb	nm	ns	az	ps	pm
ns			ns				pm
nm						nm	
nb							nb

Christian Bergelt

Genetische Algorithmen

212

Fuzzy-Regelung nach Mamdani-Assilian



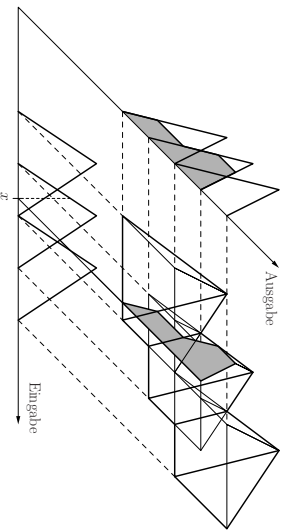
Regelbewertung in einem Mamdani-Assilian-Regler. Das Eingabe-tupel $(25, -4)$ führt zu der rechts gezeigten unscharfen Ausgabe. Aus dieser Fuzzy-Menge wird der entsprechende Ausgangswert durch Defuzzifizierung bestimmt, z.B. durch die Mean-of-Maxima-Methode (MOM) oder die Schwerpunktmethode (COG).

Christian Bergelt

Genetische Algorithmen

213

Fuzzy-Regelung nach Mamdani-Assilian



Ein Fuzzy-Regelsystem einer Meß- und einer Stellgröße und drei Fuzzy-Regeln. Jede Pyramide wird durch eine Fuzzy-Regel spezifiziert.

Der Eingabewert x führt zu der grau gezeichneten unscharfen Ausgabe.

Christian Bergelt

Genetische Algorithmen

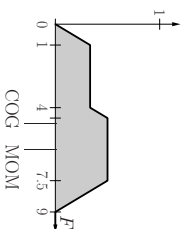
214

Defuzzifizierung

Die Auswertung der Fuzzy-Regeln liefert eine **Ausgabe-Fuzzy-Menge**.

Die Ausgabe-Fuzzy-Menge muß in einen **scharfen Stellwert** umgewandelt werden.

Dieser Vorgang heißt **Defuzzifizierung**.



Die wichtigsten Defuzzifizierungsmethoden sind:

- **Schwerpunktmethode** (Center of Gravity; COG)
Der Schwerpunkt der Fläche unter der Ausgabe-Fuzzy-Menge.

- **Flächenmittelpunktmethode** (Center of Area; COA)
Der Punkt, der die Fläche unter der Ausgabe-Fuzzy-Menge in gleich große Teile teilt.

- **Maxima-Mittelwert-Methode** (Mean of Maxima; MOM)
Das arithmetische Mittel der Stellen mit maximalen Zugehörigkeitsgrad.

Christian Bergelt

Genetische Algorithmen

215

Erzeugen/Optimieren von Fuzzy-Reglern mit GA

- Bei einem Mamdani-Assilian-Regler kann optimiert werden:
 - Die **Regelbasis** (welche Regeln, welche Ausgaben).
 - Die **Fuzzy-Mengen/Fuzzy-Partitionen** (Form, Lage, Ausdehnung, ggf. Anzahl der Fuzzy-Mengen).
 - Die **t-Norm** bzw. **t-Konorm** für die Regelauswertung (selten).
 - Parameter der **Defuzzifizierungsmethode** (falls vorhanden; selten).
 - Welche **Eingangsrößen** in den Regeln verwendet werden.
- Wir betrachten nur die Optimierung der Regelbasis und der Fuzzy-Mengen bei fester Wahl der Eingabe-/Meßgrößen.
- Die Regelauswertung geschieht (wie gezeigt) über Minimum und Maximum, die Defuzzifizierung über die Schwerpunktmethode.

Christian Bergelt

Genetische Algorithmen

216

Erzeugen/Optimieren von Fuzzy-Reglern mit GA

Mögliche Vorgehensweisen:

- **Die Regelbasis und die Fuzzy-Partitionen werden gleichzeitig optimiert.**
Nachteil: Sehr große Zahl von Parametern muß gleichzeitig optimiert werden.
- **Erst werden die Fuzzy-Partitionen bei vorgegebener Regelbasis optimiert, dann wird die Regelbasis mit den besten Fuzzy-Partitionen optimiert.**
Nachteil: Zieml. Aufstellen der Regelbasis muß Expertenwissen verfügbar sein. (Wahl einer zufälligen Regelbasis ist wenig erfolgversprechend.)
- **Erst wird die Regelbasis für vorgegebene Fuzzy-Mengen optimiert, dann werden die Fuzzy-Partitionen mit der besten Regelbasis optimiert.**
Die Fuzzy-Mengen können z.B. äquidistant (gleichmäßig) verteilt werden. In diesem Fall muß ein Anwender lediglich die Zahl der Fuzzy-Mengen je Meßgröße und für die Stellgröße vorgeben.
→ Hier betrachten wir nur diese Möglichkeit.

Christian Bergelt

Genetische Algorithmen

217

Erzeugen/Optimieren von Fuzzy-Reglern: Fitnessfunktion

- Ein guter Regler sollte verschiedene Kriterien erfüllen:
 - Aus jeder möglichen Situation sollte der Sollzustand erreicht werden.
 - Der Sollzustand sollte möglichst schnell erreicht werden.
 - Der Sollzustand sollte mit geringem (Energie-)Aufwand erreicht werden.
- Der Regler wird mehrfach testweise auf das zu regelnde System angewandt. (hier: Simulation des Stabbalance-Problems in einem Rechner; mehrere, zufällig gewählte Anfangssituationen)
- Je nach Reglerfolg/Regelgüte erhält der Regler Punkte (Anzahl Situationen, Dauer erfolgreicher Regelung, Energieaufwand).
- **Beachte:**
In dieser Anwendung ist die Bewertung der Individuen die mit Abstand aufwendigste Operation. Jedes Individuum muß über eine gewisse Mindestzahl von Zyklen zur Regelung eingesetzt werden.

Christian Bergelt

Genetische Algorithmen

218

Bewertung des Regel Erfolgs

- Weicht der Istwert zu stark vom Sollwert ab, wird abgebrochen (Fehlschlag). (z.B. Stabbalance-Problem: der Istwert muß im Intervall $[-90^\circ, 90^\circ]$ bleiben.)
- Nach einer bestimmten Dauer sollte der Istwert in der Nähe des Sollwertes liegen und dort verbleiben (Toleranzbereich). Ist dies nicht der Fall, wird ebenfalls abgebrochen (Fehlschlag).
- Der Toleranzbereich wird im Laufe der Generationen verringert (langsameres Hinfinden auf das gewünschte Ziel).
 - In den ersten Generationen ist es hinreichend, wenn der zu balancierende Stab nicht umfällt.
 - Später muß der Stab in einem immer engeren Winkelbereich um die senkrechte Lage gehalten werden.
- Die Beiträge der Stellwerte werden aufsummiert und als Strafterm verwendet. (In der Girdgewichtslage hat ein schnelles Umschalten zwischen großen Kräften gleiche Wirkung wie eine Regelung mit geringen Kräften. Hohe Kräfte sollen vermieden werden.)

Christina Bergelt

Genetische Algorithmen

219

Erzeugen/Optimieren der Regelbasis: Kodierung

- Es werden nur vollständige Regelbasen betrachtet (zu jeder Kombination von Eingabe-Fuzzy-Mengen gibt es eine Regel).
- Für jede Kombination von Eingabe-Fuzzy-Mengen muß lediglich der linguistische Term der Stellgröße festgelegt werden (i.w. Ausfüllen einer Tabelle).

Beispiel: Regelbasis für Stabbalance-Regler:

θ/ϑ	nb	nm	ns	az	ps	pm	pb
pb	az	ps	pb	pb	pb		
pm	ns	az	ps	pm	pm	pb	
ps	nm	ns	az	ps	pm	pm	pb
az	nb	nm	ns	az	ps	pm	pb
ns	nb	nm	nm	ns	az	ps	pm
nm	nb	nb	nm	nm	ns	az	ps
nb	nb	nb	nb	ns	ns	az	az

schematisch

Christina Bergelt

Genetische Algorithmen

220

Kodierung der Regelbasis

Darstellung der Regelbasis als Chromosom

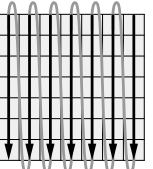
- **Linearisierung** (Umwandlung in Vektor)

Die Tabelle wird in einer willkürlichen, aber festen Reihenfolge durchlaufen und die Tabelleneinträge werden in einen Vektor aufgelistet.

Beispiel: zeilenweise Auflistung

Problem: Nachbarschaftsbeziehungen zwischen den Zeilen gehen verloren

(benachbarte Einträge sollten ähnliche linguistische Terme enthalten; dies sollte z.B. beim Crossover berücksichtigt werden)



- **Tabelle** (direkte Verwendung des Schemas)

Es werden zwei- oder mehrdimensionale Chromosomen erzeugt.

(In diesem Fall werden spezielle genetische Operatoren benötigt.)

Christina Bergelt

Genetische Algorithmen

221

Genetische Operatoren für die Regelbasis

- **Mutation:** (analog zu Standardmutation)
 - Eine Regel (ein Tabelleneintrag) wird zufällig gewählt.
 - Der linguistische Term der Ausgabe wird zufällig verändert.
 - Es können ggf. mehrere Regeln/Tabellenfelder gleichzeitig geändert werden.
 - Es ist ggf. günstig, die Mutation einer Regelbasis so einzuschränken, daß ein Tabelleneintrag nur auf linguistische Terme geändert werden kann, die dem vorhandenen Eintrag benachbart oder hinreichend nahe sind.
Beispiele: „positive small“ kann nur auf „approximately zero“ oder „positive medium“ geändert werden.
„negative big“ kann nur auf „negative medium“ oder ggf. „negative small“ geändert werden.
- (Dies verhindert ein zu schnelles „Zerfallen“ gesammelter Information; die Regelbasen werden nur „vorsichtig“ verändert.)

Christina Bergelt

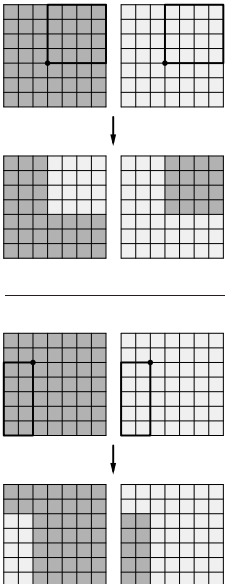
Genetische Algorithmen

222

Genetische Operatoren für die Regelbasis

- **Crossover:** (Ein-Punkt-Crossover)

Wähle zufällig einen inneren Gitterpunkt der Tabelle und eine Ecke. Tausche die so definierte Teiltafelchen zwischen zwei Eltern aus.



- **Beachte:** Um bevorzugt beobachtbare Regeln zusammen zu vererben, *solle* das Crossover ortsabhängige Verzerrung zeigen!

Christian Bergelt

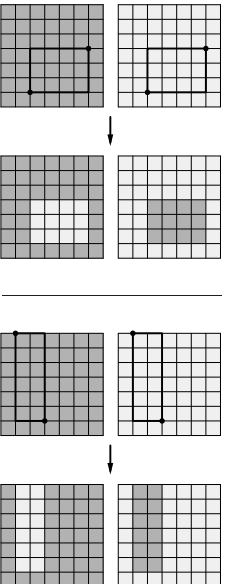
Genetische Algorithmen

253

Genetische Operatoren für die Regelbasis

- **Crossover:** (Zwei-Punkt-Crossover)

Wähle zufällig zwei Gitterpunkte der Tabelle (auch Randpunkte), Tausche die so definierte Teiltafelchen zwischen zwei Eltern aus.



- Zwei-Punkt-Crossover ist besser geeignet als Ein-Punkt-Crossover, da Teillösungen flexibler ausgetauscht werden können.

Christian Bergelt

Genetische Algorithmen

254

Optimierung der Fuzzy-Mengen

- **Gegeben:** optimierte Regelbasis mit fest gewählten und nicht veränderten äquidistanten (gleichmäßig verteilten) Fuzzy-Mengen.
- **Gesucht:** weitere Verbesserung des Reglerverhaltens durch Anpassen der Fuzzy-Mengen bei fester Regelbasis („Fine-Tuning“)
- **Kodierung der Fuzzy-Mengen:** (erste Möglichkeit)
 - Wähle die Form der Fuzzy-Mengen. (z. B. Dreieck, Trapez, Gaußlocke, Parabel, Spline etc.)
 - Liste die definierenden Parameter der Fuzzy-Mengen auf. (z. B. Dreieck: linker Rand, Mitte, rechter Rand)

Beispiel Stabbalance-Regler mit dreiecksförmigen Fuzzy-Mengen (Ausschnitt):

...	mm	ms	az	ps	...							
...	-45	-30	-15	-30	0	-15	0	15	0	15	30	...

Christian Bergelt

Genetische Algorithmen

255

Optimierung der Fuzzy-Mengen

Nachteile dieser Kodierung:

- Kodierung ist sehr „starr“ bzgl. der Form der Fuzzy-Mengen: Es wird z. B. vorher festgelegt, ob Dreiecke oder Trapeze verwendet werden.
- Genetische Operatoren können die Ordnung der Parameter zerstören (es muß bei Dreiecken z. B. gelten: linker Rand \leq Mitte \leq rechter Rand).
- Mögliche „Überholvorgänge“ zwischen Fuzzy-Mengen: Durch Mutation und Crossover kann die sinnvolle Reihenfolge der Fuzzy-Mengen zerstört werden (es sollte z. B. gelten: ms liegt rechts von ps).
- Die Bedingung „Summe der Zugehörigkeitsgrade = 1“ wird ggf. verletzt (kann durch einmalige Darstellung identischer Parameter behandelt werden).

...	-45	-15	15	-15	15	30	15	30	45	30	45	60	...
...	-45	-30	-20	-30	-20	-10	-20	-10	0	-10	10	30	...

Christian Bergelt

Genetische Algorithmen

256

