Heuristik: Genetischer Algorithmus am Beispiel des 0/1-Rucksackproblems

von Alfred Hermes

Einleitung

Was packe ich in meinen Rucksack möglichst effektiv, wenn die Gegenstände, die ich gerne mitnehmen möchte, nicht alle in den Rucksack passen? Welche Strecke ist die günstigste für meine Rundreise durch eine Anzahl von Städten? Wie finde ich einen optimalen Stundenplan für das kommende Halbjahr? Drei Probleme, die eines miteinander verbindet: Die Anzahl der Möglichkeiten steigt exponentiell mit der Anzahl der Gegenstände, Orte oder Klassen und überschreitet sehr schnell die Möglichkeiten auch noch so schneller Computer.
Betrachten wir bezüglich des Aufwandes beispielsweise das Rucksackproblem mit 100 in Betracht kommende Gegenständen unterschiedlicher Größen und Werte. Wir dürfen den Rucksack nicht überfüllen und erwarten eine optimalen Gesamtwert. Spielen wir das Ansatz durch, einen Computers alle Kombinationen durchzutesten und das Ergebnis mit dem höchsten persönlichen Wert ausgeben. Zu jedem Gegenstand untersuchen wir die Alternative, ihn einzupacken oder nicht mitzunehmen. Dieses Entscheidungsproblem führt zu 2^{100} grundsätzlichen Möglichkeiten. Sicherlich kann man mit intelligenten Vorüberlegungen viele Kombinationen von vornherein ausschließen, doch führt dies höchstens zu einer polynominalen Reduzierung, die Größenordnung bleibt bei n Gegenständen im Bereich von 2^n (n>0).

Probleme der beschriebenen Art heißen NP-vollständig (NP = nondeterministic polynominal).

Betrachten wir die Zahl 2^{100} etwas genauer: $2^{100} = 2^{10*10} > 1000^{10} = 10^{30}$. Nehmen wir an, unser Rechner könne eine Milliarde Alternativen pro Sekunde abfragen und verarbeiten, so benötige er eine Zeit von 10^{21} Sekunden, um alle Permutationen der 100stelligen Ziffernfolge aus 0 und 1 auszuprobieren. Die Zeit übertrifft das geschätzte Alter der Erde von 5 Milliarden Jahre.

Als Ausweg aus dem zeitlichen Dilemma verzichten wir auf den Anspruch, die absolut beste Lösung zu finden. Wir verwenden Heuristiken und begnügen uns mit akzeptablen Ergebnissen, die sich mehr oder weniger gut dem Optimum nähern. Zu Verfahren dieser Art gehören die Anwendung von Faustregeln, Nachbildungen menschlicher Problemlösungsstrategien oder die vereinfachte Übertragung von Optimierungsverfahren der Natur, die als genetischer Algorithmen bezeichnet werden.

Lösungsstrategie: Genetischen Algorithmus (GA) zur Optimierung des 0/1-Rucksackproblems

Problem

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

Lösung

Wir beschreiben Lösungen der Verpackungsaufgaben als 0/1-Folgen, interpretieren diese als genetische Codes und wenden einen genetischen Algorithmus an.

Nr.	0	1	2	3	4	5	6	7	8	9	
Aufgenommen:1 nicht enthalten:0	1	0	0	1	1	0	1	1	0	1	Genetischer Code (Individuum,Chromosom)
Masse	3	(7)	(4)	12	8	(10)	9	14	(10)	12	Gesamtmasse=58
Wert	3	(5)	(2)	11	4	(6)	2	15	(12)	9	Gesamtwert=44

Die Tabelle zeigt eine Permutation von Nullen und Einsen: Eingepackt werden die Gegenstände mit den Nummern 0,3,4,6,7 und 9.

Der genetische Code, das Chromosom, ist die Folge der Gene 1,0,0,1,1,0,1,1,0,1. Es handelt sich bei einem Gen um ein einziges Bit in der binären Repräsentation einer Lösung. Das entsprechende Individuum hat im Beispiel eine Gesamtmasse von 58 Einheiten und einen Wert von 44 Einheiten, der in der Sprache der GA auch mit ***Fitness*** bezeichnet wird.
Der Lösungsweg lautet in der Sprache der genetischen Algorithmen: Ausgehend von einer Anfangspopulation, also einer bestimmten Anzahl von Individuen, erzeugen wir durch Kreuzungen und eventuelle Mutationen immer neue Generationen gleich bleibender Größe mit dem Ziel, durch Selektion höherwertige Individuen und damit bessere Lösungen zu erhalten.

Information zu genetischen Algorithmen

Genetische Algorithmen orientieren sich in Anpassungs- und Optimierungsvorgängen an der natürlichen Evolution. Je besser die Anpassung eine Individuums an die Umwelt, um so höher ist seine Überlebenschance und die seiner Gattung. Die Optimierung ist eine Prozess, der von Generation zu Generation fortschreitet.
Verantwortlich für das Aussehen und mögliche Fähigkeiten der Individuen sind die Zellen, die den vollständigen Bauplan enthalten. Ihre Modifikation führt in der Evolution zur Artenvielfalt und besseren Anpassung an die Lebensbedingungen.

Überlebensfähigere Individuen beeinflussen eine Gattung in der Regel deutlicher als schwächere und sorgen durch Überlegenheit in der Fortpflanzung für eine natürliche Auswahl beziehungsweise **Selektion**.

Nachkommen erben Teile der Baupläne ihrer Eltern als Kombinationen oder **Kreuzungen**.

Manchmal entstehen Übertragungsfehler, so genannte **Mutationen**. Sie werden bei Verschlechterungen zumeist durch Selektion ausgemerzt. Gelegentlich sorgt die Mutation für bessere oder alternative Eigenschaften und gewährleistet den Erhalt der Vielfalt einer Population.

Die Codierung der Erbinformation stellt man sich ein Makromolekül (DNA) vor, in der Hierarchie ***Gen*** und ***Chromosom*** (als Folge von Genen) und mit den wesentlichen Evolutionsprinzipien

Selektion
Kreuzung (Paarung)
Mutation

Genetische bzw. evolutionäre Algorithmen produzieren Mengen von möglichen Lösungen, die in Anlehnung an das biologische Modell ***Populationen*** genannt werden. Die einzelnen Lösungen heißen ***Individuen*** und werden bei genetischen Algorithmen häufig mit Chromosomen gleichgesetzt.

Genetische Algorithmen orientieren sich an den Evolutionsprinzipien, um in der Simulation von Generationswechseln in die Nähe einer optimalen Lösung vorzudringen, ohne den gesamten Lösungsraum zu durchsuchen. Voraussetzung ist eine realistische Abbildung eines Problems auf ein genetisches Modell.

Übertragung auf eine 0/1-Folge

Eine 0/1-Folge repräsentiert den genetischen Code bzw. das Chromosom eines Individuum. Sein Wert bzw. die persönliche Fitness lässt sich aus Anordnungen der Nullen und Einsen ablesen. Jede Folge codiert eine mögliche Lösung des Problems. In einer Population, d.h. einer Menge von Individuen, treten bei den ersten Generationen höchst wahrscheinlich Lösungen unterschiedlicher Qualität auf. Durch Anwendung der Evolutionsprinzipien entstehen neue Generationen gleicher Größe mit dem Ziel, bessere Lösungen zu produzieren. Das Individuum der letzten Generation mit dem höchsten Fitnesswert gilt als Repräsentanz einer möglichst optimierten Lösung.
Neue Generation lösen ihre Vorgänger durch einen vollständigen Austausch ab. Sie entstehen durch Anwendung der Evolutionsprinzipen Selektion, Kreuzung und Mutation nach folgendem Muster[1]:

Selektion

Aus zwei zufällig bestimmten Individuen einer Population wird das beste der beiden als erstes Mitglied eines Elternpaares ausgewählt. Ebenso erfolgt die Auswahl des zweiten Elternteils. Durch Kreuzung der beiden entstehen zwei neue Individuen, die mit einer geringen Wahrscheinlichkeit eine Mutation durchlaufen. Die Auswahl wiederholt sich, bis die neue Populationsgröße mit der alten übereinstimmt[2].

Kreuzung (1-Punkt-Überkreuzverfahren)

Bei der Kreuzung sind je zwei Lösungen zu kombinieren. Betrachten wir die folgenden Individuen:

[1] Die vorgestellten Muster sind nicht zwingend. Es gibt eine Vielzahl Ansätzen genetischer Algorithmen.

[2] Selektion: Man könnte auch zufällige Elternpaare bestimmen und die Population zeitweilig vergrößern und beim Wechsel durch Auswahl der möglichst besten die ursprüngliche Größe wieder herzustellen.

1	0	0	1	1	0	1	1	0	1	Chromosom des 1. Elternteils
1	1	1	0	0	1	1	0	1	1	Chromosom des 2. Elternteils

Beim ***Ein-Punkt-Überkreuzverfahren*** (engl. 1-point crossover): dient eine zufällige Stelle im genetischen Code als Trennmarkierung. Der Code der neuen Individuen setzt sich aus der Kombination beider Abschnitte zusammen.

1. Abschnitt 2. Abschnitt

1	0	0	1	1	0	1	1	0	1	Chromosom des 1. Elternteils
1	1	1	0	0	1	1	0	1	1	Chromosom des 2. Elternteils

Neue Codes:

1. Abschnitt 2. Abschnitt

1	0	0	1	0	1	1	0	1	1	Chromosom des 1. Kindes
1	1	1	0	1	0	1	1	0	1	Chromosom des 2. Kindes

Das Verfahren lässt auch die exakte Kopie der genetischen Codes beider Eltern zu.

Mutation (bit flip)

Die Simulation von Mutationen beschränkt sich bei der Bitumwandlung auf eine eventuelle Veränderung des Wertes an einer zufälligen Stelle des Codes in sein Gegenteil: Eine Eins wird zur Null oder umgekehrt. Eine Mutation wird mit geringer Wahrscheinlichkeit ausgelöst und mit Hilfe des Zufallszahlengenerators entschieden. Um beispielsweise Mutationswahrscheinlichkeit von einem Prozent zu simulieren, erfolgt eine Mutation, wenn eine Zufallszahl des Wertebereichs [0,1[kleiner oder gleich 0,1 ist, andernfalls nicht."

Zufallszahl 5:

1	0	0	1	0	1	1	0	1	1	Individuum, Chromosom
1	0	0	1	0	0	1	0	1	1	nach der Mutation

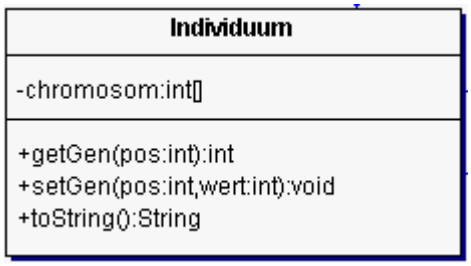
Begriffssammlung zu Genetischen Algorithmen (GA)

Individuum	eine Struktur, die eine Lösung repräsentiert, besteht aus Genen
Chromosom	wird hier mit Individuum gleichgesetzt
Gen	ein Bit der binären Repräsentation einer Lösung
Population	Menge von Individuen, auf die der genetischen Algorithmus angewendet wird
Eltern	zwei ausgesuchte Individuen einer Population
Kreuzung	Kombination aus den Genen zweier Individuen/Chromosome
Mutation	leichte Modifikation eines Individuums/Chromosoms
Fitness	Qualität einer Lösung
Generation	Eine Iteration im Laufe der Optimierung
Genotyp	kodierte Lösung eines des Problems
Phaenotyp	entschlüsselte Lösung des Problems

Lösungsdiskussion

Die Hauptdarsteller

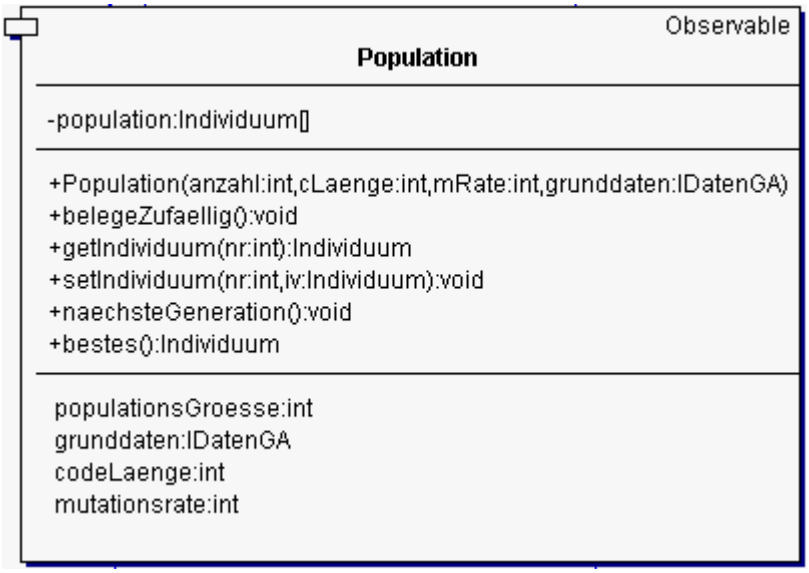
Gene: Ein Gen wird hier als Bit der binären Repräsentation einer Lösung codiert.



Individuum: Ein Individuum (hier gleichbedeutend mit Chromosom) repräsentiert eine Lösung. Es besteht aus Genen, die als natürliche Zahlen codiert werden. Beim 0/1-Rucksackproblem kommen ausschließlich die Zahlen 0 und 1 in Frage. Mit den Methoden *getGen()* und *setGen()* lässt sich das Chromosom initialisieren und auslesen. Die Methode *toString()* liefert eine Darstellung des Chromosoms als Zeichenkette. Falls keine explizite Initialisierung erfolgt, haben alle Gene eine Objektes der Klasse *Individuum* den Wert 0.

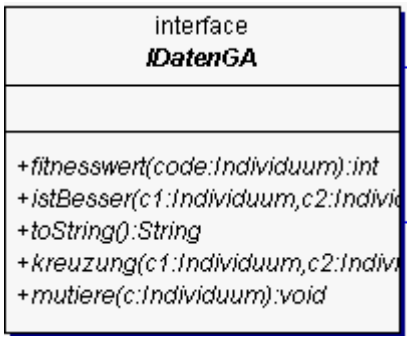
Paar: Kreuzungen führen zu neuen Individuen, die den genetischen Code ihrer Eltern erben. Das Ein-Punkt-Überkreuzverfahren liefert ein Geschwisterpaar. Der Entwurf der Klasse *Paar* beruht auf Programm technischen Überlegungen, um Objekte dieser Art als Rückgabewerte von Funktionen zu ermöglichen. Mit *erstes* und *zweites* lassen sich die Individuen auslesen.

Population: Eine Menge von Individuen, auf die der genetische Algorithmus angewendet wird.
Als Attribute eignen sich die Anzahl der beteiligten Individuen, deren Codelänge zur Repräsentation von Lösungen, die Mutationsrate in Prozent und gewisse Grunddaten und -funktionen, die zur Gütebestimmung und zur Interpretation von codierten Lösungen notwendig sind. Eine Schnittstelle (*interface IDaten*) normiert die benötigten Methoden durch Festlegung ihrer Signaturen. Wie die Methoden implementiert werden, hängt vom Problem ab. Die Klasse Population implementiert die Schnittstelle *Observable* um als Modell im Reigen von Modell, Sicht und Steuerung zu operieren. Es benachrichtigt nach jedem Generationswechsel mögliche angemeldete Sichten, um diese auf dem Laufenden zu halten[3].



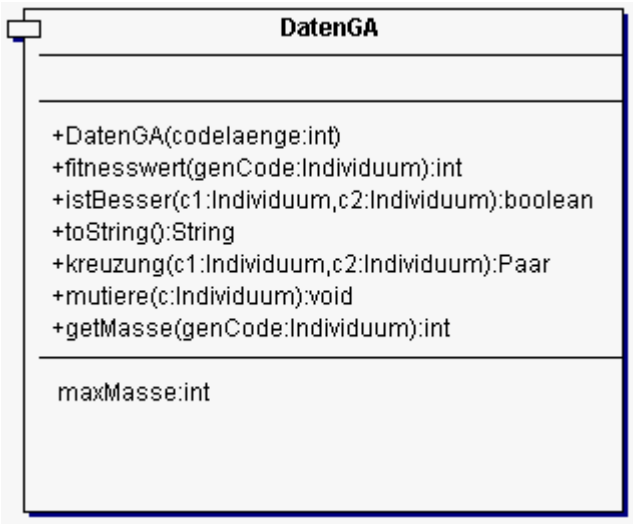
[3] MVC unter der Lupe, Alfred Hermes, in diesem Heft

IDatenGA: Das Interface enthält die Signaturen von Grundfunktionen genetischer Algorithmen. Sie sind dem Problem entsprechend zu implementieren.

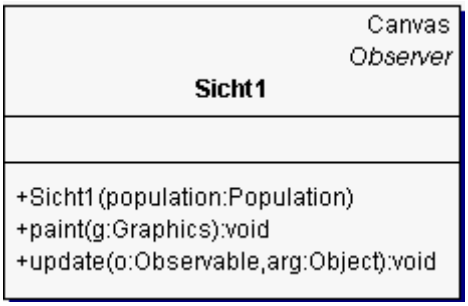


fitnesswert() liefert die Güte bzw. den Wert eines Individuums
istBesser() teilt mit, ob der Wert eines Individuums höher ist, als der eines zweiten.
toString() veröffentlicht die wesentlichen Grunddaten als Zeichenkette.
kreuzung() liefert auf der Basis zweier Chromosome ein Paar von Individuen, die aus dem Kreuzungsvorgang hervorgehen.
mutiere() führt eine Mutation an einer zufälligen Stelle durch.
Ein Interface bzw. Schnittstelle ähnelt einer abstrakten Klasse. In Java kann eine Klasse mehrere Schnittstellen implementieren aber nur eine abstrakte Klasse.

DatenGA: implementiert *IDatenGA*.
Die Berechnung der Fittnesswerte von Individuen basieren beim 0/1-Rucksackproblem auf den Eigenschaften Wert und Masse jedes Gens. Falls die Gesamtmasse zu hoch ist, hat das Individuum den Wert 0. Für die Berechnungen neuer Generationen genügen die im Interface festgelegten Operationen. Zur Visualisierung können Kenntnisse über Nebenbedingungen, wie die maximale Masse eines Individuums, nützlich sein. Daher bietet die Klasse diesbezüglich zusätzliche Methoden an.



Sicht: visualisiert den Optimierungsprozess und die Lösungen.



Zur Darstellung von Populationen und Generationswechseln, eventuell animiert, eigenen sich Malflächen (*Canvas, bei Swingklassen: Panel*) . Eine Aktualisierung ist jeweils angebracht, wenn eine neue Generation die alte ablöst. Die Informationen kommen vom Objekt der Klasse *Population*, wenn die Sicht sich angemeldet hat. Damit der Datenaustausch zwischen dem Modell *Population* und der Sicht funktioniert, implementiert *Sicht* die Schnittstelle *Observer* und die Methode *update()*, die vom Modell gegebenenfalls aufgerufen wird. Über das Modell kann die Sicht auch auf die Daten des Objekts der Klasse *DatenGA* zugreifen und sie zur Visualisierung gebrauchen.

Mögliche graphische Darstellungen
Die folgenden Schnappschüsse zeigen Momentaufnahmen von Sichten beim Start und nach 10 Generationswechseln. Die Anzahl der Gene beträgt 10, die der Individuen einer Population 30 und Mutationen treten mit einer Wahrscheinlichkeit von 1% auf.

Nr.	0	1	2	3	4	5	6	7	8	9
Wert	1	4	2	1	8	6	9	2	4	7
Masse	4	10	7	4	1	8	5	4	3	1

Die beste Lösung lässt sich bei der kleinen Chromosomgröße exakt bestimmen:

Beste Lösung: 0 1 1 0 1 1 1 0 1 1, Wert = 40, Masse = 35

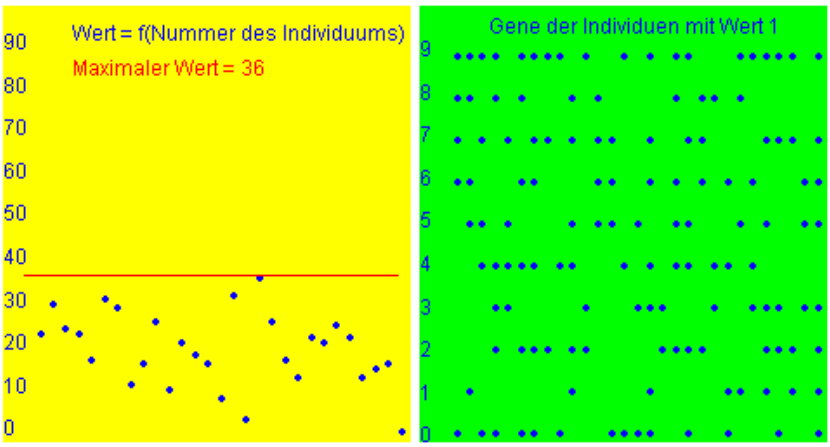


Abbildung 1: Sichten der ersten Generation

Die erste Generation zeigt eine Vielfalt von Individuen, deren Nummern auf der Horizontalen und deren Werte im linken Bild vertikal abgetragen sind. Das rechte Bild präsentiert die Chromosome. Gene (vertikal) mit dem Wert 1 sind als Punkte dargestellt.

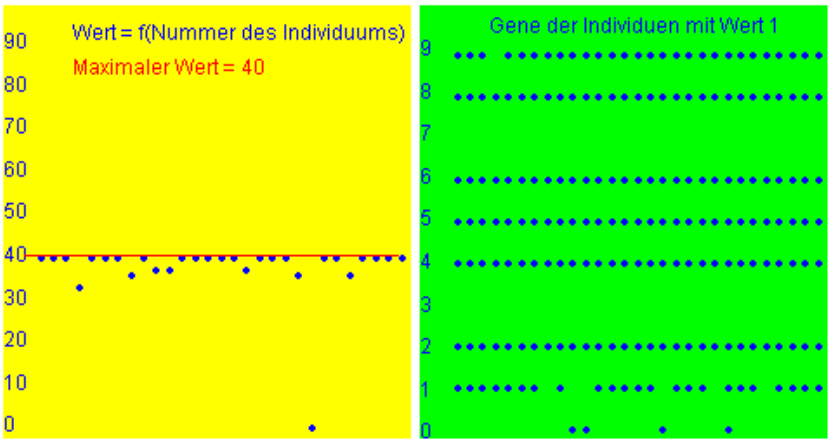


Abbildung 2: Sichten der zehnten Generation

Nach dem 10. Generationswechsel stabiliesiert sich die Darstellung. Die meisten der 30 Individuen repräsentieren die tatsächlich beste Lösung. Selbst bei einer Chromosomgröße von 10 und bei jeweils 50 Individuen liefert der genetische Algorithmus nach kürzester Zeit stabile Strukturen, die einer optimalen Lösung offensichtlich sehr nahe kommen.

Codierungshinweise (Java)[4]

Individuum
Die Gene sind als Folge von Zahlen implementiert. Auf Grund der Codelänge lassen sich die Reihungen begrenzen und initialisieren. *toString()* liefert Grunddaten des Individuums als Zeichenkette: Die Gene werden in ihrer Reihenfolge aufgezählt und mittels Leerzeichen getrennt.
Beispiel: “1 0 1 1 0“ (bei einer Codelänge von 5).

```
public class Individuum{

public Individuum(int codelänge){
  this.codelänge = codelänge;
  chromosom = new int[codelänge];
}

  public int getGen(int pos){
    return chromosom[pos];
  }

  public void setGen(int pos, int wert){
    chromosom[pos]= wert;
  }

  public int getCodelänge(){
    return codelänge;
  }

  public String toString(){
    String s="";
    for (int i =0; i < codelänge; i++){
      s = s+chromosom[i];
      if (i<codelänge-1) s = s+" ";
    }
    return s;
  }
  private int[] chromosom;
  private int codelänge;
}
```

Quelltext 1: class Individuum

[4]Vollständige Quellcodes: www.zitadelle.juelich.de/informatik/login/ga/RSProg.zip

Population

```
import java.util.*;
public class Population extends Observable{
    public Population(int anzahl, int cLaenge, int mRate,
                      IDatenGA grunddaten) {
        populationsGroesse = anzahl;
        population = new Individuum[anzahl];
        mutationsrate= mRate;
        codeLaenge= cLaenge;
        this.grunddaten = grunddaten;
        belegeZufaellig();
    }
.....

public void naechsteGeneration() {
      Individuum[] naechsteGen = new Individuum[populationsGroesse];
      for (int k=0; k < populationsGroesse; k=k+2){
         Individuum i1 = liesZufällig(); //zufällige Auswahl
         Individuum i2 = liesZufällig(); //zufällige Auswahl
         Paar p = grunddaten.kreuzung(i1,i2);
         naechsteGen[k] = p.getErstes();         // 1. Kind
         int zufall = (int) (Math.random()*100);
         if (zufall <= mutationsrate)
           grunddaten.mutiere(naechsteGen[k]);
         if (k+1 < populationsGroesse){
          naechsteGen[k+1] = p.getZweites();      // 2. Kind
          zufall = (int) (Math.random()*100);
          if (zufall <= mutationsrate)
           grunddaten.mutiere(naechsteGen[k+1]);
           }
      }
      population = naechsteGen;
      setChanged(); notifyObservers(); //Benachrichtigung der Sichten
    }
....
```

Quelltext 2: Ausschnitte aus Population

PDatenGA
Basierend auf den Attributen

```
    private int[] massen,
    private int[] werte,
    private int maxMasse,
    private int codelaenge,
```

die vom Konstruktor zu initialisieren sind, handeln die Methoden zur Bestimmung der Fitnesswerte, der Kreuzung (engl. one point crossover) und Mutation (engl. bit flip) nach den oben beschriebenen Richtlinien. Die Fitness ist gleich null, wenn die maximal mögliche Masse überschritten wird.

```
     public int fitnesswert(Individuum genCode){
        int wert = 0;
        int masse=0;
        for (int k=0; k<codelaenge; k++){
            if(genCode.getGen(k)==1){
            wert = wert + werte[k];
            masse = masse + massen[k];
          }
        }//for
        if (masse > maxMasse) {
          wert = 0;
        }
        return wert;
    }
```

Quelltext 3: Fitness

„Besser“ kann je nach Aufgabestellung ein höherer Wert, wie beim 0/1-Rucksackproblem, aber auch ein niedriger Wert , wie bei der Planung einer Kosten günstigen Rundreise, bedeuten.

```
public boolean istBesser(Individuum c1, Individuum c2){
      return fitnesswert(c1) >= fitnesswert(c2);
    }
```

Quelltext 4: Vergleich von Fitnesswerten

Unter vielen möglichen Kreuzungsstrategien zeigt der folgende Quelltext das „Ein Punkt Überkreuzverfahren“. Eine zufällig ausgewählte Stelle im Chromosom dient als Teilungsmarke. Bis zur Marke bleibt der Ursprungscode erhalten, danach gilt der Code des Partners.

```
public Paar kreuzung( Individuum c1, Individuum c2){
  int codeLaenge = c1.getCodelänge();
  int stelle = (int) (Math.random()*codeLaenge)+1;
  Individuum i1 = new Individuum(codeLaenge);
  Individuum i2 = new Individuum(codeLaenge);
  for(int k = 0; k < stelle; k++) {
    i1.setGen(k, c1.getGen(k));
    i2.setGen(k, c2.getGen(k));
  }
  for(int k = stelle; k < codeLaenge; k++) {
    i1.setGen(k, c2.getGen(k));
    i2.setGen(k, c1.getGen(k));
  }
  return new Paar(i1,i2);
}
```

Quelltext 5: Kreuzungsstrategie

Die Mutation betrifft höchstens ein auszuwürfelndes Bit. Sie alterniert den jeweiligen Wert.

```
    public void mutiere( Individuum c){
     int pos =  (int)( Math.random()*c.getCodelänge());
      if (c.getGen(pos) == 1) c.setGen(pos,0);
      else c.setGen(pos,1);
    }
```

Quelltext 6: Mutation an einer zufälligen Stelle

Nachtrag: Exakte Lösung bei kleiner Anzahl

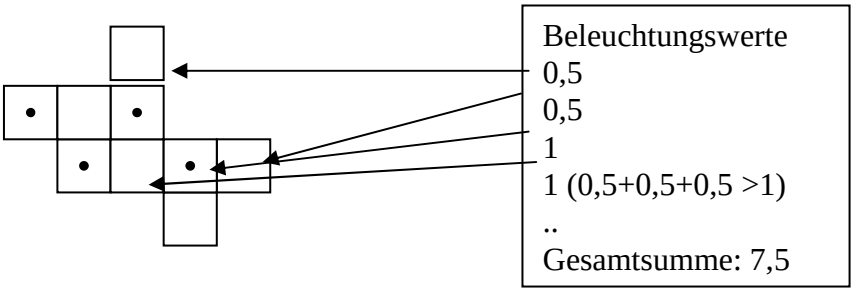
Die folgende Lösung in Java verwendet Rekursion und durchsucht durch Rückverfolgung den gesamten Lösungsraum.

```
public class ExakteLoesung {
  public ExakteLoesung (DatenGA d, int cLaenge){
    codeLaenge = cLaenge;
    grunddaten=d;
   }

  /** Falls der Fitnesswert des Individuums besser ist als der des
      aktuell besten, werden der maximale Wert neu festgelegt und
      und das aktuell beste durch Kopie aller Genwerte restauriert.
      Ein Gleichsetzen würde dazu führen, dass bei jeder Änderung
      des aktuellen Individuums sich auch das beste ändert und am Schluss
      nur mit Nullen belegt wäre. */
  private void testeOptimum (){
      int wertNeu = grunddaten.fitnesswert(individuum);
      if (grunddaten.istBesser(individuum, bestIndividuum)){
       maxWert = wertNeu;
       for(int k=0; k<codeLaenge; k++){ //kopiere den Code
         bestIndividuum.setGen(k, individuum.getGen(k));
       }
      }
  }

  /**Durchsuchung des gesamten Lösungsraumes */
  private void sucheBestes(int position){
   individuum.setGen(position,1);  // setze eine 1
   if (position + 1 < codeLaenge){
         sucheBestes(position+1);
   }
    else testeOptimum();
   individuum.setGen(position,0);  //versuche mit 0 statt mit 1
   if (position + 1 < codeLaenge)
     sucheBestes(position+1);
   else
     testeOptimum();
  }

  public Individuum bestes(){
```

```
    maxWert=0;
    bestIndividuum=new Individuum(codeLaenge);  //0,0,0,0,...
    individuum = new Individuum(codeLaenge);
    sucheBestes(0);
    return bestIndividuum;
  }

  private DatenGA grunddaten;
  private int codeLaenge;
  private Individuum bestIndividuum;
  private int maxWert;
  private Individuum individuum;
}
```

Aufgabe: Beleuchtung

(Die Idee entstammt einer früheren Aufgabe des Bundeswettbewerbs Informatik)



Ein verwinkelter Raum, der in Quadrate eingeteilt werden kann, ist optimal zu beleuchten. Die Lampen werden, wenn überhaupt, über den Quadratmitten angebracht (Punkte in der Skizze). Jede Lampe erzeugt in dem zugehörigen Quadrat eine ausreichende Helligkeit (100%). Die vier Nachbarquadrate profitieren von dem Leuchtkörper durch eine Aufhellung von 50%. Weiter entfernte Quadrate oder diagonale Nachbarn erfahren der Einfachheit halber - entgegen der Wirklichkeit – keine Aufhellung. In dem stark vereinfachten Modell werden Helligkeitswerte, die von mehreren Lampen erzeugt werden addiert, übersteigen allerdings nicht den Wert von 100%.

Mit Hilfe eines genetschen Algorithmus ist eine gute Lösung zur optimalen Beleuchtung des Raumes bei einer festgelegten Anzahl von Lampen (siehe Beispielskizze) zu finden. Jedes Quadrat soll mindestens zu 50% beleuchtet sein.

Nummeriert man im Beispiel die Planquadrate in Schriftrichtung durch, findet man Lampen in den Quadraten 1,3, 4 und 6 und erhält eine codierte Lösung (0,1,0,1,1,0,1,0,0). Attribute der Quadrate sind ihre Nachbarn und ihre Ausleuchtung.

Erweiterung: Die Aufgabe ist unter Einbeziehung physikalischer Gesetze realistischer zu behandeln.

Projektvorschlag: Das Problem des Handlungsreisenden

Ein Handlungsreisender plant eine kostengünstige Rundreise durch eine größere Anzahl von Orten, die er jeweils nur einmal besuchen möchte. Die Kosten leiten sich aus den Entfernungen ab.
Das Problem entspricht im Lösungsumfang dem des 0/1-Rucksackproblems (NP-vollständig). Es existieren viele Ansätze zur Abbildung auf ein genetisches Modell. Die Definition von Individuen, Kreuzungen und Mutationen müssen überdacht werden.

Eine Diskussion des Problems und eine mögliche Lösung stellt LogIn im Heft Nr. 125 vor.

Zusammenfassung

Genetische Algorithmen (GA) sind in den USA von John Holland und Ann Arbor in Michigan entwickelt worden. Es handelt sich um heuristische Verfahrung zur annähernden Lösung von Optimierungsproblemen. Die speziellen Lösungsmethoden orientieren sich an der Evolution der Natur und reduzieren sich auf Selektion, Kreuzung und Mutation.
Ihr Einsatz drängt sich bei Problemen mit riesigen Lösungsräumen auf, die aus Zeitgründen die Suche einer nachweisbaren optimalen Lösung nicht zulassen. Ein klassisches Einsatzgebiet sind NP-vollständige Probleme (NP = nondeterministic polynominal), für die keine optimalen und in ihrer Qualität abschätzbareren Lösungen bekannt sind. Falls exakte Lösungen möglich sind, sollten heuristische Verfahren ausgeschlossen werden. Es wäre beispielsweise nicht angebracht, genetische Verfahren zum Sortieren von Zahlenfolgen zu verwenden (Fitnesswert = Qualität der Sortierung), da eine exakte Lösung innerhalb eines Zeitlimits im Gegensatz zu bekannten Verfahren nicht garantiert werden kann.

Heuristische Verfahren eröffnen den Schülerinnen und Schülern eine nicht nur für die Informatik spezifische Sichtweise
von NP-vollständigen Problemen,
von Schwierigkeiten, selbst mit schnellsten Computern riesige Lösungsräume gewisser Optimierungsprobleme zu durchforsten und
von Möglichkeiten, mit heuristische Verfahren recht gute Lösungen zu finden, wenn auf der Anspruch der Suche nach dem Optimum fallen gelassen wird

Die Behandlung genetischer Algorithmen als evolutionäre Verfahren bietet sich im Zusammenhang mit NP-Vollständigkeit im vierten Halbjahr des Informatikunterrichtes der Sekundarstufe II an. Während die algorithmischen Anforderung und die verwendeten Datenstrukturen recht elementar sind, steigen die Ansprüche bei der Planung und Strukturierung der Unterrichtseinheit, um einen Prototyp zur Lösung ähnlicher Probleme zu erstellen.

Literatur und Internetquellen

Die behandelten Programme in Aktion und weitere Anwendungen findet man im Internet unter der folgenden Adresse: www.zitadelle.juelich.de/informatik/login/ga. Die Quellcodes in Java lassen sich in der selben Adresse unter dem Namen RSProg.zip abrufen.

Darwin, Ian F.: Java cookbook, O'Reilly & Associates, Sebastopol, 2001.

Gunther Dueck, Tobias Scheuer, Hans-Martin Wallmeier: Toleranzschwelle und Sintflut: neue Wege zur Optimierung, Spektrum der Wissenschaft: Wissenschaftliches Rechnen, 1999

Eckart Zitzler, Applet und Lernaufgabe zu "Genetische Algorithmen"
http://www.educeth.ch/informatik/interaktiv/genalg/