Auf Darwins Spuren

Mit Genetischen Algorithmen nach dem biologischen Vorbild auf evolutionäre Art komplexe Probleme lösen.

Klaus Meffert

Bestimmte Klassen von Problemen sind durch unflexible Heuristiken nur mit sehr hohem Aufwand lösbar. Das gilt besonders dann, wenn der potentielle Lösungsraum riesig ist oder eine suboptimale Lösung genügt, die vorgegebenen Kriterien entspricht. Genetische Algorithmen bieten einen ebenso einfachen wie effizienten Ansatz zur Lösung derartiger Probleme. Der Artikel führt in die Thematik der Genetischen Algorithmen ein und veranschaulicht die Implementierung mit Java und dem Framework JGAP anhand eines Beispiels.

Die von Charles Darwin begründete Evolutionstheorie beschäftigt sich mit der Entstehung neuer Arten durch natürliche Selektion. Durch genetische Rekombination bildet sich aus vorhandenem Erbgut neues aus. Welche Population letztendlich überlebt und sich weiter ausbilden kann, entscheidet die Praxistauglichkeit der Population. Dieses einfache Prinzip kann auf Computerprogramme übertragen und zur Lösung vieler Problemklassen verwendet werden. Während die Praxistauglichkeit in der Natur durch die Fähigkeit, sich an gegebene Umwelteinflüsse anzupassen, ermittelt wird, bedarf es bei den sogenannten Genetischen Algorithmen (kurz: GA) der Implementierung eines als Fitness- oder Bewertungsfunktion bezeichneten Bewertungsmaßstabes.

Die Mächtigkeit evolutionärer Algorithmen wurde nicht zuletzt durch die NASA unter Beweis gestellt. Für die 2001 gestartete Mission des Mars Odyssey Orbiter wurde eine extrem leistungsfähige Antenne benötigt. Herkömmliche Produkte genügten den Ansprüchen der NASA nicht. Durch ein an den Erkenntnissen von Charles Darwin angelehntes Verfahren gelang es, eine effiziente Lösung mit dem Computer zu entwickeln. Die dort angewandte Heuristik beruht auf den gleichen Grundansätzen, wie sie in diesem Artikel vorgestellt werden. Nach einer Einführung werden anhand eines Beispiels die Schritte zur Implementierung eines solchen evolutionären Algorithmus, der zur Lösung unterschiedlichster Probleme verwendet werden kann, verdeutlicht.

Das biologische Grundprinzip

Die Implementierung eines GA mit Java orientiert sich stark am biologischen Vorbild. Wie funktioniert Vererbung in der Natur? Diese Frage müssen wir erst beantworten, bevor wir darauf aufbauende technische Betrachtungen anstellen können.

Die Chromosomen sind Träger des Erbgutes, das durch die DNS (Desoxyribonukleinsäure) in Form einer Doppelhelix festgelegt ist. Das Erbgut eines Organismus ist in seinen Genen verankert. Ein Gen ist der Informationsträger für ein Merkmal. Mehrere Gene sind auf einem Chromosom untergebracht, der Mensch hat ein Set von zweimal 23 davon in jeder Zelle (mit Ausnahme der Keimzellen). Die spezifische Merkmalsausprägung eines Gens heißt Allel. Pro Gen kann es mehrere Merkmalsausprägungen, also Allele, geben. Ein Allel repräsentiert demnach eine alternative Zustandsform eines Gens. Einander entsprechende Gene heißen deshalb Allele.

Bevor neues Leben entsteht (meist durch Sexualität) muss das Erbgut der Eltern zusammenkommen und die Befruchtung der Eizelle initiieren. Die Entstehung von neuem Erbgut unterliegt im Wesentlichen den folgenden Grundprinzipien.

Von den zur Verfügung stehenden Chromosomen werden einige Allele unverändert übernommen. Das wird als Replikation oder Reproduktion bezeichnet. Andere unterliegen dem Vorgang der Mutation. Die Mutation sorgt für zufällige Veränderungen des Erbgutes. Auf Dauer

setzen sich nur gutartige Veränderungen durch. Kurzfristige Defekte können in diesem Prozess ebenso auftreten. Das Crossing Over sorgt für eine Vermischung von selektierten Allelen. Dies kann man sich einfach so vorstellen, dass mitten in einem Gen ein Kreuzungspunkt festgelegt wird. An diesem Punkt wird das eine Allel geteilt. Jeder der beiden Teile wird nun durch einen passenden Teil eines anderen Allels ergänzt, das zuvor ebenfalls an einem korrespondierenden Kreuzungspunkt getrennt wurde.

Die Selektion bestimmt, welche Allele, die durch die gerade beschriebenen Mechanismen erzeugt wurde, für die nächste Generation (also für das neu entstehende Lebewesen) relevant sind. Selektiert werden Allele in der Natur von Eltern, die bis dahin überlebt haben. Die besten Individuen haben dabei naturgemäß die größten Chancen sich fortzupflanzen. Das Verfahren ist auch deshalb flexibel, weil schwächere Individuen mit einer geringeren Wahrscheinlichkeit ebenfalls selektiert werden können.

Darwin auf dem Computer

Wie lässt sich der biologische Vorgang der Evolution auf den Computer übertragen? Indem wir eine geeignete Repräsentation der biologischen Elemente (Chromosom, Gen, Allel) und Vorgänge (Replikation, Crossing Over etc.) durch ein Programm abbilden.

Glücklicherweise existieren bereits mehrere frei verfügbare Frameworks, die bei der Implementierung von GA's die meiste Arbeit abnehmen, indem sie die biologischen Konzepte ausreichend abbilden. Die eigentliche Leistung besteht nun darin, zu einem gegebenen Problem eine sinnvolle und effiziente Repräsentation zu finden und eine Fitness-Funktion aufzustellen. Wir wollen anhand eines Beispiels das generelle Vorgehen zur Umsetzung eines GA's in ein Programm verdeutlichen. GA's eignen sich besonders gut für Probleme, für die ein großer potentieller Lösungsraum existiert, deren Lösung näherungsweise erfolgen kann und für die das Vorgehen zum Finden einer Lösung nicht von vornherein feststeht. Bei richtiger Aufstellung der Problemrepräsentation und der Fitness-Funktion ist das Finden einer sehr guten Annäherung nur eine Frage der Zeit. Genauso wie in der Biologie ist der Schwerpunkt also nicht, eine perfekte Lösung zu finden, sondern eine, die unseren Ansprüchen genügt.

Die Problemrepräsentation ist wichtiger Bestandteil eines GA. Sie setzt sich zusammen aus der verwendeten Datenstruktur und der Interpretation der aktuellen Ausprägung bzw. Werte der Daten. Die mit den jeweiligen Werten eines Datenelements assoziierten Interpretationen werden als ‚Funktionen' und ‚Terminale' bezeichnet. Terminale sind atomare Einheiten wie beispielsweise die willkürliche Zahl 2,583. Funktionen besitzen Parameter und sind problemspezifisch. In unserem folgenden Übungsprojekt handelt es sich tatsächlich um mathematische Funktionen.

Ein Beispiel: Formel Finder

Wir wollen versuchen, zu einer vorgegebenen Wertetabelle (siehe Tabelle 1), die aus Paaren von Eingabe- und Ausgabewerten besteht, eine dazu passende Formel zu finden. Dieses für einen Menschen anspruchsvolle Problem kann ein GA mit verhältnismäßig wenig Aufwand in sehr guter Näherung lösen. Den Pseudocode für unsere Problemlösungsstrategie pro Generation finden wir in Kasten 1.

Ziel ist es, die totale Abweichung über alle Wertepaare zu eliminieren, sie also gegen Null zu bringen. Die Güte einer ermittelten Funktion wird demnach durch die Differenz der aktuellen Ausgabe des GA zum erwarteten Ausgabewert gemessen (und zwar summiert über alle Paare von Ein- und Ausgabewerten). Jede Differenzbildung wird absolut vorgenommen, negative (Teil-)Differenzen werden also vor Aufsummieren auf die Gesamtdifferenz positiv gemacht. Eine Fitness-Funktion bestimmt also die Güte eines Individuums einer Generation. Die Fitness-Funktion simuliert also gewissermaßen die Umwelt, die Lebensbedingungen jedes Individuums.

JGAP – Ein GA Framework

JGAP ist ein frei verfügbares Framework zur Implementierung Genetischer Algorithmen. Es stellt alle notwendigen Strukturen und Logiken dafür zur Verfügung. Der Quelltext sowie umfangreiche Dokumentationen sind unter dem Link am Ende des Artikels verfügbar. Anhand unseres Beispiels gehen wir die einzelnen Schritte zur Lösung unserer Aufgabe mit JGAP durch. Ein alternatives Frameworks ist am Ende des Artikels zu finden, ebenso der Quelltext zum Formel Finder. Ein einfacheres Beispiel ist direkt in JGAP enthalten.

Zuerst initialisieren wir die GA-Konfiguration. Hierzu verwenden wir die Klasse *org.jgap.impl.DefaultConfiguration*. Sie stellt eine Standard-Konfiguration für JGAP bereit, die für viele Problemstellungen geeignet ist. Hierin werden die zu verwendenden genetischen Operationen (Mutation etc.) festgelegt. Außerdem legt sie den Mechanismus zur Selektion von Nachfahren fest. Das geschieht im Normalfall mit dem *org.jgap.impl.WeightedRouletteSelector*. Dieser Selektor funktioniert ähnlich einem Roulette-Rad. Er weist jedem Chromosom eine Anzahl von Schlitzen (Slots) zu, die abhängig von deren Fitness ist. Je mehr Slots ein Chromosom einnimmt - also je fitter es ist -, desto wahrscheinlicher ist dessen Selektion für die nächste Generation. Die Selektion findet durch Drehen des Rades statt, genau wie beim Roulette. Weiterer Bestandteil der *DefaultConfiguration* ist ein Zufallsgenerator, der das Interface *org.jgap.RandomGenerator* implementiert. Die Klasse *org.jgap.impl.StockRandomGenerator* tut genau dies, indem sie einfach alle Methoden von *java.util.Random* erbt.

Weiterer Bestandteil der Konfiguration ist ein exemplarisches Chromosom, das eine charakteristische Belegung an Genen besitzt. Es wird der Konfiguration anfangs einmalig zugewiesen und ermöglicht es JGAP, allgemeine Daten (wie die Anzahl der Gene im Chromosom) zu ermitteln und Plausibilitätsprüfungen durchzuführen.

Als nächstes stellen wir eine Fitness-Funktion auf. Wie sie genau aussieht, kann dem Quelltext entnommen werden. Wir weisen sie dem Konfigurationsobjekt zu und informieren so JGAP über deren Existenz. Die Fitness-Funktion muss, wie bereits angedeutet, so konzipiert sein, dass sie auch wirklich nur den besten Allelen die höchsten Noten gibt. Außerdem sollte sie prinzipiell einer besseren Lösung einen höheren Fitnesswert bescheinigen als einer schlechteren Lösung. Die Fitness-Funktion sollte also monoton sein. JGAP setzt voraus, dass die Klasse unserer Fitness-Funktion die Basisklasse *org.jgap.FitnessFunction* erweitert, also von ihr erbt.

Zur Umsetzung der Fitness-Funktion gehört auch die Interpretation der aktuellen Ausprägung eines Chromosoms. Diese Interpretation ist willkürlich. Sie sollte natürlich so gewählt sein, dass sie Sinn macht und für den Menschen leicht nachvollziehbar ist. In unserem Beispiel haben wir einfach ein zusammengesetztes Gen, ein *org.jgap.impl.CompositeGene*, verwendet. Es ist vergleichbar mit einer Liste, die mehrere atomare Gene oder weitere zusammengesetzte Gene enthalten kann. Für den Formel Finder haben wir zwei Gene in der Liste aufgenommen. Bei beiden handelt es sich um ein *org.jgap.impl.IntegerGene*. Dieses Gen enthält einfach nur einen Integerwert. Das erste Gen dieser Art repräsentiert entweder eine Funktion oder eine Konstante (also eine Zahl). Das zweite Gen in der Liste repräsentiert einen Operator (Plus, Minus usw.). Dass eine Funktion immer einen Parameter benötigt, wird bei der Erstellung einer zufälligen Formel berücksichtigt. Formeln, die ungültig sind, etwa bei Ziehen einer Wurzel aus einer negativen Zahl, werden einfach ignoriert. Das können wir uns leisten, weil hinreichend viele Formeln im zufälligen Erzeugungsprozess gültig sind.

Die Auswertung einer aufgestellten Formel ist nicht ganz trivial. Java erlaubt zwar die Auswertung regulärer Ausdrücke zur Laufzeit, was aber ist mit mathematischen Formeln? Hierfür kann ein beliebiger Interpreter verwendet werden, der unseren Ansprüchen genügt. Unsere Wahl fiel auf den JeksParser, der kostenlos erhältlich ist (siehe Links). Er unterstützt sogar direkt Konstanten, Operatoren und Funktionen aus der Klasse *java.lang.Math*. Um eine Formel auszuwerten, übergeben wir JeksParser eine Zeichenkette der Formel zusammen mit einem Eingabewert. Das Ergebnis ist der Ausgabewert, der durch Einsetzen der Eingabe in die Formel berechnet werden kann. Weitere Details können dem beiliegenden, kommentierten Quelltext entnommen werden.

Nun können wir den GA seiner Bestimmung übergeben und simulieren das Voranschreiten der Zeit und somit das Hervorbringen neuer Populationen. Wir lassen die Anfangspopulation so oft weiterentwickeln (also Kinder haben), wie wir das für nötig halten. Das bewerkstelligt JGAP mit dem Befehl *evolve()* aus Klasse *org.jgap.Genotype*. Zwei Abbruchkriterien sind denkbar: Entweder der GA findet eine für uns ausreichend gute Lösung. Oder eine obere Anzahl von Durchläufen wird überschritten. Im letzten Fall wird das bis dahin als bestes ermittelte Ergebnis ausgegeben.

Darwins Erbe

Wir haben anhand eines nicht trivialen Beispiels gezeigt, welche erstaunlichen Resultate ein Genetischer Algorithmus bei der Lösung komplexer Probleme hervorbringen kann. Das trifft insbesondere auf Probleme zu, deren potentielle Lösungsmenge sehr groß ist. Jedoch ist ein GA nicht zur Lösung aller Probleme geeignet. Für eine Vielzahl von Kategorien gibt es eine leistungsfähigere Variante eines evolutionären Verfahrens. Diese Variante, die ein Superset des GA ist, wird Genetische Programmierung (kurz: GP) genannt. Im Rahmen eines GP erzeugt der Algorithmus durch eine evolutionäre Herangehensweise Programme. Diese Programme sind natürlich wesentlich leistungsfähiger als die bloße statische Interpretation von Allelen, wie dies bei GA's geschieht. Das Konzept der GP sowie ein Beispiel dazu stellen wir in einem Folgeartikel vor.

Links & Literatur

- JGAP: http://jgap.sourceforge.net/
- JeksParser: http://www.eteks.com/jeks/en/
- GAJIT: http://www.angelfire.com/ca/Amnesiac/gajit.html

Tabelle 1: Testfälle für den Formel Finder (Ausschnitt)

Eingabe: x	Erwartete Ausgabe: y
1	6.28318
3	18.8495
15.7	98.6460
-100	-628.3185
5000	31415.9265
-1000	-6283.1853

Gesucht: Funktion f(x), so dass für alle x das Ergebnis das korrespondierende y ist.

Kasten 1: Pseudocode Genetischer Algorithmus

```
Bestimme Zufallspopulation nach biologischem Vorbild.
Interpretiere aktuelle Population und erstelle eine Formel daraus.
Setze die totale Abweichung auf 0.
Für jeden Eingabewert der vorgegebenen Wertetabelle:
    • Berechne den Funktionswert (die Ausgabe des GA also).
    • Ermittle die absolute Differenz zwischen berechnetem und erwartetem
      Funktionswert.
    • Addiere diese Differenz zur totalen Abweichung.
Wenn die totale Abweichung tolerabel ist, gib die gefundene Formel aus. Fertig.
Ansonsten beginne von vorne.
```

Kasten 2: Konfiguration eines GA

Die richtige Wahl der Konfigurationsparameter ist entscheidend für den Erfolg. Falsch gewählte Werte führen zu Problemen bei der Lösungssuche. Hier einige Tipps:

- Wenn der GA für ein Problem über längere Zeit eine nicht besonders gute Lösung konstant beibehält, kann die Erhöhung der Populationszahl helfen.
- Die Fitness-Funktion ist essentiell. Sie ist das einzige Kriterium, das dem GA hilft, bessere Lösungen erkennen zu können.

- Eine sehr langsame Lösungssuche kann durch zu niedrige Wahrscheinlichkeiten für Mutation und Crossing Over begründet sein.
- Je mehr Terminale für den GA zur Verfügung stehen, desto wahrscheinlicher ist eine längere Laufzeit bis zum Finden einer hinreichend guten Lösung.