Computational Intelligence

M. Sc. Andreas Buschermöhle, M. Sc. Jan H. Schoenke

Übungsblatt 1 - Evolutionäre Algorithmen I: Genetische Algorithmen

Abgabe bis Mittwoch, 14.05.2014, 12:00 Uhr

In diesem Übungsblatt werden Algorithmen in Matlab geschrieben und zur Auswertung ein- und zweidimensionale Plots angefertigt. Erstellen Sie im Dateiordner Ihrer Gruppe in Stud.IP einen Ordner mit dem Namen `Blatt 1` und laden Sie zu zweit sowohl die m-Files, als auch die fertigen Matlab-Figures mit sinnvoller Achsenbeschriftung rechtzeitig in diesen Ordner hoch. Geben Sie in Vips je nach Aufgabenstellung die textuelle Interpretationen ihrer Ergebnisse sowie ggf. die Namen der Dateien an, die sich auf die jeweilige Aufgabe beziehen.

Aufgabe 1.1:Framework für Genetische Algorithmen (2 + 4 + 10 + 4 = 20 P)

In dieser Aufgabe soll ein Framework für die Untersuchung genetischer Algorithmen erstellt werden. Hierzu müssen im Wesentlichen vier Funktionalitäten bereitgestellt werden.

- Startpopulation erstellen
- Fitness evaluieren
- Crossover
- Mutation

Zunächst muss eine Anfangspopulation erstellt werden, mit der die Optimierung beginnt. Für eine solche Population muss dann die Fitness der einzelnen Individuen evaluiert werden. Basierend auf dieser Fitness können dann Individuen für das Crossover ausgewählt werden, durch das die Individuen der Folgegeneration entstehen. Diese werden abschließend durch einen Mutationsoperator variiert und die nächste Generation steht bereit. Im Folgenden werden die einzelnen Funktionen für Matlab umgesetzt.

a) Erstellen Sie zunächst eine Funktion zum Erzeugen einer Population mit folgendem Funktionskopf:

```
function population = generatePopulation(popSize, genSize)
```

Eine Population lässt sich einfach als zweidimensionale binäre Matrix darstellen, wobei ein Index das Gen und der andere das Individuum angibt. Der Parameter `popSize` gibt die Anzahl der Individuen in der Population an und der Parameter `genSize` die Größe des Genoms eines Individuums. Es sollen alle Gene der Population zufällig gleich verteilt initialisiert werden. Füllen Sie nun diese Funktion aus.

b) Die Evaluation der Fitness basiert hier auf einer Funktion $f(x)$, für die der Funktionswert maximiert werden soll. Das Genom ist dabei die Binärdarstellung des Arguments x. Der Funktionskopf für die Fitnessevaluation ist wie folgt gegeben:

```
function fitness = evalFitness(population, fhandle)
```

Die Population in `population` enthält die Binärrepräsentationen der Argumente und alle sollen in eine Dezimaldarstellung überführt werden. Diese kann dann mit der Funktion, die mit dem Function Handle (Function Handles wurden im Tutorial vorgestellt. Siehe auch die Matlab-Hilfe.) `fhandle` übergeben wurde, ausgewertet werden und das Ergebnis liefert direkt die Fitness, die in einem Rückgabevektor `fitness` als Resultat geliefert wird. Zum Umwandeln der Binärdarstellung lässt sich die Matlab-Funktion `bin2dec` verwenden. Diese erwartet als Eingabe einen String mit 0 und 1. Um einen Binärvektor `binary_vector` aus Matlab in einen solchen String zu überführen, kann der folgende Ausdruck verwendet werden: `char('0'+binary_vector)`.

c) Für die Rekombination (oder Crossover) der Gene zu einer Folgepopulation müssen zum einen je Individuum zwei Eltern zufällig proportional zu ihrer Fitness ausgewählt werden. Zum anderen wird für das 1-Punkt Crossover eine zufällige Position im Genom gewählt, an der die beiden Teilstücke der Eltern zusammengefügt werden. Für das vollständige Erzeugen einer neuen Generation durch Corssover soll eine Funktion mit folgendem Funktionskopf geschrieben werden:

```
function nextPopulation = doCrossover(population, fitness)
```

Die `population` enthält die aktuelle Generation und der `fitness`-Vektor die Fitnesswerte der Individuen. Für die nächste Generation können dann in einer Schleife für jedes neue Individuum zwei Eltern zufällig gewählt werden, wobei sich die Wahrscheinlichkeit p_i für die Auswahl aus der relativen Fitness f_i zur Gesamtpopulation nach $p_i = \frac{f_i}{\sum_j f_j}$ ergibt. Überlegen Sie sich hierzu eine Strategie, entsprechend der Wahrscheinlichkeitsverteilung ein Elternteil zu wählen. Sind zwei Elterngenome gewählt, muss eine Crossover-Position zufällig gewählt werden und das Kind dann aus den beiden Teilstücken der Eltern zusammengesetzt werden. So lässt sich iterativ eine vollständig neue Generation zusammensetzen. Schreiben Sie diese Matlab Funktion.

d) Für die Mutation der neuen Generation wird jedes Gen mit einer einstellbaren (niedrigen) Wahrscheinlichkeit invertiert. Der Funktionskopf hierfür ist wie folgt gegeben:

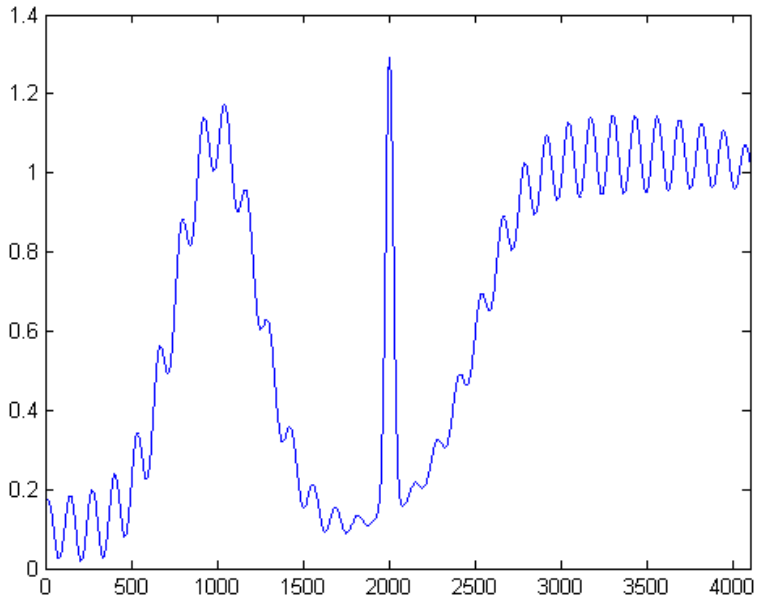
```
function population = mutatePopulation(population, flipProbability)
```

Dabei wird direkt die übergebene Population verändert, Ein- und Ausgabevariable sind also identisch. Die `flipProbability` $\in [0, 1]$ gibt die Wahrscheinlichkeit an, dass ein Gen invertiert wird. Füllen Sie nun diese Funktion aus, sodass jedes Gen durchlaufen und abhängig vom übergebenen Parameter zufällig invertiert wird.

Aufgabe 1.2: Maximumssuche (8 + 6 + 2 + 10 + 4 = 30 P)

Das in Aufgabe 1 entwickelte Framework für genetische Algorithmen soll nun eingesetzt werden, um das Maximum einer Funktion zu suchen. Die Funktion, die in dieser Aufgabe betrachtet werden soll liegt als m-File in StudIP (Funktion.m) vor und lautet

$$f(x) = e^{-(0{,}003x-3)^2} + 1{,}5e^{-\frac{1}{\left(1+e^{-\left(\frac{x-3500}{682}\right)^2}\right)^9}} + 1{,}15e^{-\frac{(x-2000)^2}{1000}} + \frac{\sin\left(\frac{x}{20}\right)+\cos\left(\frac{x}{21}\right)}{20} - 0{,}45\ .$$



Wird die Genomgröße zu 12 Genen gewählt, erlaubt dies eine Repräsentation von Zahlen aus dem Wertebereich $[0, 4095]$. Auf diesem Wertebereich verhält sich die Funktion wie in der obigen Abbildung zu sehen. Sie enthält einen Peak und viele lokale Maxima, die nah beieinander liegen und von denen sich die größeren in zwei Cluster aufteilen lassen. Somit stellt sie besondere Herausforderungen an den Optimierungsalgorithmus. Bei dieser einfachen Testfunktion ist das tatsächliche Maximum bekannt und kann somit genutzt werden, um das Ergebnis des genetischen Algorithmus zu bewerten. Der gesuchte Zielwert liegt bei einem Maximum von $1{,}2932 = f(2000)$.

a) Um das Framework einzusetzen, muss die zu maximierende Funktion in einem eigenen m-File abgelegt werden, dessen Name dann als Function-Handle an die Fitnessbewertung übergeben werden kann. Anschließend kann der genetische Algorithmus geschrieben werden, in dem anfangs eine Startpopulation gebildet und diese bewertet wird. Dann sollen in einer Schleife die Generationen iteriert werden. In jedem Schleifendurchlauf findet zunächst ein Crossover statt, um eine neue Generation zu erstellen. Dann wird diese neue Generation mutiert und anschließend ihre Fitness bewertet. Als Parameter sollten zu Beginn die Anzahl der Individuen in einer Generation, die flipProbability und die Anzahl der Generationen festgelegt werden.

   Merken Sie sich zudem in jedem Durchlauf das bislang Beste gefundene Individuum, sowie die beste, durchschnittliche und schlechteste Performance der Population. Plotten Sie das Resultat am Ende in zwei Subplots. Der erste Subplot soll die beste, durchschnittliche und schlechteste Performance im Lauf der Generationen zeigen und als Referenz auch das

tatsächliche Optimum angeben. Der zweite Subplot soll den Argumentwert des besten Individuums sowie das zugehörige Optimum zeigen. Verwenden Sie als Parameter zunächst eine Populationsgröße von 16 Individuen, eine flipProbability von 0.01 und 100 Generationen, die durchlaufen werden. Schreiben Sie dieses Programm und speichern Sie es in der Datei `ga_timeline.m`.

Da das Verhalten für jeden Durchlauf zufällig ist, testen Sie mehrere Durchläufe und wählen Sie zwei bis drei repräsentative Verläufe aus, die Sie als Figure abspeichern. Interpretieren Sie Ihre Ergebnisse.

b) Um die Abhängigkeit von den Parametern systematischer zu untersuchen, soll nun die Populationsgröße zwischen 2 und 16 Individuen und zudem die flipProbability zwischen 0.001 und 0.1 variiert werden. Für jede Parametereinstellung sollen dann wiederum 100 Generationen evolviert und das beste gefundene Individuum abgespeichert werden. Um die Varianz des Resultats durch den Zufall zu verringern, werden für jede Parameterkonstellation 50 Durchläufe gestartet. Von diesen 50 Durchläufen soll das schlechteste, beste und mittlere Ergebnis der Fitness gespeichert werden. Tragen Sie dann abschließend die drei Fitnessverläufe je in einem zweidimensionalen Plot über der Populationsgröße und der flipProbability auf, den Sie als Figure abspeichern. Schreiben Sie dieses Programm und speichern Sie es in der Datei `ga_paramTest.m`.

Interpretieren Sie ihre Ergebnisse.

Achtung: Diese Auswertung kann eine längere Rechenzeit in Anspruch nehmen. Beginnen Sie also frühzeitig mit der Bearbeitung dieser Untersuchung!

c) In diesem Aufgabenteil wird die zu maximierende Funktion f mit einem konstanten Offset von 100 versehen. Die neue zu maximierende Funktion f_2 lautet also $f_2(x) = f(x) + 100$. Wiederholen Sie die Untersuchungen aus Aufgabenteil a). Welche Unterschiede lassen sich erkennen? Interpretieren Sie Ihre Ergebnisse.

d) Um dem in Aufgabenteil c) beobachteten Effekt entgegenzuwirken werden im Folgenden zwei Varianten des Selektions- und Überlebenskriteriums untersucht: das windowing und die elitäre Selektion.

windowing: Beim windowing wird nur die Fitness-proporionale Selektion verändert. Die Fitness aller Individuen, nach welcher die Elternteile der kommenden Generation ausgewählt werden, wird um die Fitness des schlechtesten Individuums reduziert. Anschließend wird mit der so verschobenen Fitness eine Fitness-proportionale Selektion durchgeführt. Das schlechteste Individuum einer jeden Generation trägt also nichts zur jeweils folgenden Generation bei.

elitäre Selektion: Bei der elitären Selektion verbleiben die k besten Individuen einer jeden Generation unverändert erhalten, d.h. auf diesen Individuen wird auch kein Mutationsoperator ausgeführt. Die übrigen Nachkommen der folgenden Generation werden durch Fitness-proportionale Selektion und Rekombination aus allen Individuen der aktuellen Generation gebildet und anschließend durch Mutation verändert.

Vergleichen Sie das Verhalten dieser beiden Strategien mit dem Standardverfahren des Genetischen Algorithmus bei der Maximierung von f und f_2. Verwenden Sie dazu eine Populationsgröße von 40 mit einer Mutationswahrscheinlichkeit von 0.1 über 100 Generationen und behalten Sie bei der elitären Selektion jeweils die vier besten Individuen. Speichern Sie für eine systematische Untersuchung die Ergebnisse der Fitness von 200 verschiedenen Durchläufen der drei Varianten führt die Maximierung von f sowie f_2 und stellen Sie diese als Histogramm dar. Markieren sie im Histogramm jeweils die Fitness des linken und rechten Clusters sowie des Peaks auf der x-Achse. Interpretieren Sie Ihre Ergebnisse.

e) Ändern Sie die Interpretation des Genoms als Argumentenwert von Binär zu Gray-Code und führen Sie die Untersuchungen aus Aufgabenteil a),b) und d) erneut durch. Ändert sich das Verhalten signifikant?

Für die Umwandlung von Gray-Code in die binäre Darstellung können Sie folgende Funktion verwenden, welche die Bitrepräsentation ebenfalls wie die Funktion `bin2dec` als String erwartet:

```
function b = gray2bin(g)
  b(1) = g(1);
  for i = 2 : length(g);
    x = xor(str2num(b(i-1)), str2num(g(i)));
    b(i) = num2str(x);
  end
```

Aufgabe 1.2: **Tic Tac Toe** **(20+25+5 = 50 P)**

In dieser Aufgabe soll mit Hilfe eines Genetischen Algorithmus ein (möglichst) perfekter Spieler für das einfache Spiel Tic Tac Toe entwickelt werden.

Beim Spiel Tic Tac Toe setzen zwei Spieler abwechselnd ihre jeweiligen Symbole (Kreise und Kreuze) auf ein freies Feld in einem 3×3 Felder großen Spielfeld. Der Spieler, welcher als erster drei Symbole in einer Zeile, Spalte oder Diagonale platziert, gewinnt das Spiel. Wenn es keine freien Felder mehr gibt und kein Spieler die Gewinnbedingung erfüllt, endet das Spiel unentschieden.

Ein perfekter Tic Tac Toe Spieler verliert nie. Spielen zwei perfekte Tic Tac Toe Spieler gegeneinander, dann endet das Spiel stets unentschieden.

a) Erstellen Sie ein Konzept für die Darstellung einer Spielstrategie und einen Genetischen Algorithmus, der durch Turniere eine gute Spielstrategie entwickelt. Dabei empfiehlt sich folgendes Vorgehen:

   - Wählen Sie eine Kodierung für das Spielfeld.
     - Anmerkung: Viele Situationen auf dem Spielfeld lassen sich durch Rotation und Spiegelung ineinander überführen und bilden somit eine Äquivalenzklasse.
     - Wie können Sie eine beliebige Situation auf dem Spielfeld eindeutig auf einen Repräsentanten der zugehörigen Äquivalenzklasse abbilden?
     - Wie viele gültige Situationen des Spielfeldes gibt es?
     - In welchen dieser Situationen gibt es eine Auswahl über die möglichen Züge?
   - Entwickeln Sie eine Darstellung für einen Tic Tac Toe Spieler.
     - Wie viele Gene hat das von Ihnen gewählte Genom?
     - Was repräsentiert ein einzelnes Gen?
     - Wie können sie die Anzahl der Gene verringern, um den Suchraum zu verkleinern?
   - Geben Sie einen Mutations- und Rekombinationsoperator sowie ein Überlebenskriterium und eine Selektionsstrategie an.
     - Wie können Sie sicherstellen, dass der Mutationsoperator nur Spieler erzeugt, die immer gültige Züge machen?
   - Geben Sie eine Fitnessfunktion an, die den Ausgang eines einzelnen Spiels für beide beteiligten Spieler bewertet. Erweitern Sie anschließend ihre Fitnessfunktion, sodass die Ergebnisse mehrerer Spiele berücksichtigt werden.
     - Ist Ihre Fitnessfunktion für die fitnessproportionale Selektion geeignet?
     - Welche Modifikationen sind ggf. nötig?

b) Setzten Sie Ihr Konzept in Matlab um. Dabei empfiehlt sich folgendes Vorgehen:

   - Schreiben Sie eine Funktion, welche eine beliebige Situation auf dem Spielfeld in der von Ihnen gewählten Kodierung auf den jeweils eindeutigen Repräsentanten der zugehörigen Äquivalenzklasse abbildet. Als weiteren Rückgabewert sollten Sie die zugehörige Rücktransformation angeben.
   - Schreiben Sie eine Funktion, welche eine initiale Population von Individuen erzeugt.
   - Schreiben Sie eine Funktion, welche den von Ihnen gewählten Mutationsoperator umsetzt.
   - Prüfen Sie ob Sie den Rekombinationsoperator auf Aufgabe 1 bzw. 2 wiederverwenden können. Schreiben Sie eine Funktion, welche den von Ihnen gewählten Rekombinationsoperator umsetzt.

- Schreiben Sie einen Tic Tac Toe Spieler, der immer zufällig zieht. Wird diesem eine beliebige Situation auf dem Spielfeld präsentiert, wählt er zufällig eines der freien Felder aus und gibt dieses Feld in der von Ihnen gewählten Kodierung als Position für seinen Zug zurück.
- Schreiben Sie eine Funktion, die zwei beliebige Spieler gegeneinander antreten lässt und den Ausgang des Spiels eindeutig kodiert zurückgibt. Die Funktion sollte die beiden Spieler als Function-handle erwarten, wobei das erste Übergabeargument den Spieler repräsentiert, der als erster setzen darf. Die hinter den Function-handles hinterlegten Funktionen sollten für jede beliebige Situation auf dem Spielfeld die Position der Feldes zurückgeben, auf welches der zugehörige Spieler sein Symbol setzten will.
- Schreiben Sie eine Funktion, welche die von Ihnen gewählte Fitness-Funktion umsetzt, indem sie jeden Spieler der Population mehrfach gegen den zufällig spielenden Tic Tac Toe Spieler mit Hin- und Rückrunde antreten lässt. In der Hinrunde hat der Spieler aus der Population den ersten Zug, in der Rückrunde der zufällig ziehende Spieler.
- Schreiben Sie eine Variante der von Ihnen gewählten Fitness-Funktion, bei der die Spieler der Population jeweils mit Hin- und Rückrunde gegeneinander antreten.
- Prüfen Sie welche Teile des Frameworks aus Aufgabe 1 Sie wiederverwenden können und schreiben Sie ein Framework mit dem Sie für die Fitness-Evaluation sowohl Spiele gegen den zufällig ziehenden Spieler, als auch Spiele gegeneinander und eine Kombination von beidem verwenden können.

c) Evolvieren Sie mit Hilfe Ihres Genetischen Algorithmus in Matlab einen Tic Tac Toc Spieler. Wählen Sie abschließend das beste von Ihnen gefundene Individuum aus und schreiben Sie eine eigenständige Funktion in Matlab, die das von Ihnen gewählte Individuum verwendet, um für eine beliebige (aber gültige) Situation des Spielfeldes den nächsten Zug zu bestimmen. Wählen Sie für Ihre Funktion folgenden Funktionskopf, wobei Sie `XX` durch die Nummer Ihrer Gruppe ersetzen:

```
function position = bestPlayerGroupXX(field)
```