Schlangenwürfel

Programmierprojekt in Artificial Life

Maximilian Mühlfeld Malte Schmitz Eike von Tils

5. Februar 2013

Inhaltsverzeichnis

Einleitung **1**
initiale Population 2
Evaluation 2
Selektion 2
Reproduktion 2

Funktionsbeschreibung **3**
Klasse `Point` 3
Klasse `Snake` 3
Klasse `Evolution` 4
Skript Evolution 5
Skript Brute-Force 6

Durchführung **6**
Energiefunktion E_a 6
Energiefunktion E_b 7

Einleitung

Das Ziel des Versuches war es, anhand des Beispiels des Schlangenwürfels einen evolutionären (genetischen) Algorithmus zu implementieren. In unserem Fall handelt es sich bei den Individuen des Algorithmus um Verdrehungen eines Schlangenwürfels in der Ebene. Diese werden durch einen String repräsentiert, der für jedes Glied der Schlange die Richtung angibt, in der das Folgeglied zu finden ist. Da die Schlange sich im 2D aufhält, gibt es die Richtungen `I` (geradeaus), `R` (rechts) und `L` (links). Die Energie einer Konfiguration wird aus einer Energiefunktion bezogen auf ihre Form und Lage im Raum berechnet.
Hierzu wurde ein Algorithmus nach dem folgenden Schema entwickelt:

1. initiale Population erzeugen
2. Bis maximale Epoche oder minimale Energie erreicht wird, wiederhole:
   a) Evaluation
   b) Selektion
   c) Reproduktion

initiale Population erzeugen

Die Initiale Population besteht bei unserem Problem aus einer Menge von zufällig erstellten Schlangen. Eine Schlange wird durch einen Bitstring repräsentiert. Da die Position der `I`-Glieder der Schlange konstant bleibt, sind nur die Richtungen `R` und `L` von Bedeutung. Wir codieren diese Richtungen als Richtungsänderungen bezüglich der vorherigen Position. Eine 1 steht hierbei für Richtungsänderung, eine 0 für gleich bleibende Richtung. Beginn ist immer eine Rechtsdrehung `R`.

Evaluation

Zur Bewertung der Konfigurationen kommen zwei unterschiedliche Energiefunktionen E_a und E_b zum Einsatz.

- E_a berechnet hierbei den Durchmesser des kleinsten Kreises auf der Ebene, der die Schlange umschließt.
- E_b berechnet die gesamte Fläche aller Löcher, die die Schlange bildet.

Die genauere Erläuterung zur Implementierung folgt bei der Funktionsbeschreibung.

Selektion

Die Selektion erfolgt als zufällige Auswahl von Individuen der aktuellen Population. Die Wahrscheinlichkeit, das ein Individuum gewählt wird, ist hierbei von der Energiefunktion abhängig. Individuen mit niedrigerer Energie werden dabei bevorzugt.

Reproduktion

Reproduktion wird 1. durch Mutation und 2. durch Rekombination ausgeführt.

1. Mutation erfolgt durch das zufällige Kippen einer zufälligen Anzahl an Bits im Bitstring. Durch die gewählte Kodierung ist sichergestellt, das alle folgenden Gelenke mit gedreht werden, sodass die Mutation eines Bits genau dem verdrehen eines Gelenks in die andere Richtung entspricht.
2. Rekombination wird durch das zufälligen Wählen eines Einsprungpunktes im Bitstring initiiert. Die beiden, ebenfalls zufällig gewählten, Individuen werden an den entsprechenden Stellen gekappt. Ein neues Individuum wird dann geboren, bestehend aus einem Teil des ersten Individuums bis zum Einsprungpunkt und dem zweiten Teil des zweiten Individuums ab dem Einsprungpunkt.

Funktionsbeschreibung

Die Implementierung erfolgte in Ruby. Der Quelltext und dieses Dokument stehen unter der MIT-Lizenz[1] und wurden auf GitHub[2] veröffentlicht. Im folgenden wird auf die einzelnen Funktionen und Klassen eingegangen.

Klasse `Point`

Die Klasse `Point` ist eine Hilfsklasse, die zur Repräsentation eines Punktes im 2D-Raum mit den Koordinaten `x` und `y` dient. Sie verfügt im Wesentlichen über folgende Methoden:

- `+(other)`
  Die Methode `+` addiert zwei Instanzen von `Point`. Hierbei wird jeweils auf die Koordinaten `x` und `y` die der übergebenen Instanz addiert, und eine neue Instanz zurückgegeben.
- `rotate!(d)`
  Rotiert die Koordinaten nach links (`L`) oder rechts (`R`). Die Rotation erfolgt hierbei jeweils um 90°.
- `min(a,b)` bzw. `max(a,b)`
  Bestimmt die minimale bzw. maximale Lage eines Punktes im Raum bezüglich der Koordinatenwerte `x` und `y`.

Klasse `Snake`

Die Klasse `Snake` dient zur Repräsentation einer Schlange auf einem 2D-Gitter. Zusätzlich beinhaltet diese Klasse die implementierten Energiefunktionen.

- `to_board`
  Hier wird versucht die Schlange auf ein 2D-Gitter zu legen. Kommt es zu Überschneidungen, so wird `nil` zurückgegeben. Ansonsten das Board mit Schlange. Das zurückgegebene Board ist zudem minimal groß.
- `to_string`)
  Ersetzt die kodierte Darstellung der Schlange (1 für Richtungsänderung, 0 für gleiche Richtung) durch die Richtungsangaben `I`, `R` und `L` anhand eines Templates in `snake_static`, indem `R` und `L` durch `X` ersetzt sind.

[1] `http://www.opensource.org/licenses/MIT`
[2] `https://github.com/malteschmitz/snakecube`

- `energy_a`
  Berechnet die Energiefunktion anhand der Größe des Boards auf dem die Schlange liegt. Der Durchmesser des kleinsten Kreises ist identisch mit der Diagonalen des Boards. Diese wird mithilfe des Satzes von Pythagoras berechnet und zurückgegeben.

- `energy_b`
  Berechnet die Energiefunktion anhand der Anzahl der Löcher. Hierzu wird das Board um die Schlange herum mit einem Flood-Filling-Algorithmus gefüllt. Alle Stellen auf dem Board, die dann noch leer sind, sind Löcher die von der Schlange umschlossen sind. Die Anzahl dieser Stellen wird zurückgegeben.

- `energy_c`
  Invertiert die Energiefunktion `energy_b`, damit eine Minimierung der Energie zu einer maximal großen Lochfläche führt.

Klasse `Evolution`

Die Klasse Evolution implementiert die Funktionalität eines evolutionären Algorithmus.

- `start`
  Die Methode erzeugt eine zufällige Population von Individuen als Binärzahl fester Länge mit vorgegebener Anzahl an Individuen.

- `crossover`
  Erzeugt neue Individuen durch Crossover. Hierzu wird ein zufälliger Einsprungpunkt gewählt an dem der Crossover stattfindet (s. o.).

- `mutation`
  Erzeugt neue Individuen mit zufälligen Mutationen. Zuerst wird hierzu eine Bitmaske erstellt, die pro Bitstelle mit einer vorgegebenen Wahrscheinlichkeit eine 1 enthält. Diese wird dann mit der Kodierten Schlange per `xor` verrechnet, so dass ein Bit an den Stellen gekippt wird, wo die Maske 1 ist.

- `step`
  Führt einen kompletten Iterationsschritt aus. Durch Aufruf von `crossover` und `mutation` werden neue Individuen erzeugt.

  Anschließend werden diese nach ihrer Energie, gewichtet mit einer Zufallszahl, sortiert. Aus dieser sortierten Folge werden die ersten Individuen bis zur vorgegebenen Populationsgröße gewählt. Alle anderen Individuen sterben aus.

- `iterate`
  Diese Methode startet die komplette Berechnung des Algorithmus. Sie ruft die Methode `start` auf und dann in jeder Iteration einmal `step`. Sie iteriert solange,

bis entweder die maximale Anzahl erreicht wurde, oder eine Energiegrenze unterschritten wurde.

- `bear`
  Erzeugt ein neues Individuum bestehend aus dem kodierten Bitstring und dem Wert der Energiefunktion.

Skript `snake_evolution.rb`

Dieses Skript erzeugt eine neue Schlange und setzt die Parameter des Algorithmus. Es startet anschließend den Algorithmus und gibt die Ergebnisse aus. Die Parameter sind dabei im einzelnen:

- `length`
  Anzahl der Gelenke der Schlange.
- `energy`
  Zu benutzende Energiefunktion.
- `size`
  Größe einer Population.
- `crossover`
  Anzahl an durch Crossover zu erzeugenden Individuen pro Iteration.
- `mutation`
  Anzahl an durch Mutationen zu erzeugenden Individuen pro Iteration.
- `flip`
  Wahrscheinlichkeit, dass ein Bit bei der Mutation gekippt wird.
- `selection`
  Anteil der Zufallskomponente bei der Selektion nach Energie der Individuen (Faktor aus dem Bereich von 0 bis 1).
- `n`
  Maximale Anzahl an Iterationen.
- `energy`
  Minimale Energie, bei deren Unterschreitung die Iteration früher beendet werden kann.
- `logging`
  Flag, das angibt, ob ein Log aller Energielevel erzeugt wird.

Skript `snake_bruteforce.rb`

Dieses Skript erzeugt die gleiche Schlange, verwendet aber zum Vergleich einen Brute-Force-Ansatz, bei dem über alle möglichen Konfigurationen iteriert wird. Auf dieses Weise wird das absolute Energie-Minimum sicher gefunden, sodass die Ergebnisse des evolutionären Algorithmus besser beurteilt werden können.

Durchführung

Das Programm wird auf der Kommandozeile durch den Befehl

```
ruby snake_evolution.rb
```

gestartet.

Die Ausgabe besteht aus der Anzahl an Iterationen, der endgültigen Population sowie einer Liste aller Energielevel, die erreicht wurden und jeweils des ersten Individuums mit dieser Energie. Am Ende wird die Schlange auf ihrem 2D Gatter gezeichnet.

Als Schlange wird `IIXXXIXXIXXXIXIXXXXIXIXIXII` verwendet.

Energiefunktion E_a

Es werden die folgenden Parameter verwendet:

- `length = 27`
- `energy = energy_a`
- `size = 15`
- `crossover = 5`
- `mutation = 15`
- `flip = 0.3`
- `selection = 0.5`
- `n = 10000`
- `energy = 9`
- `logging = true`

Wir erhalten folgendes Ergebnis:

```
number of iterations:
52

final population:
8.602325        1110110110010010        IILRLILRILLRILILLRRIRILILII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
```

```
9.899495        1111100101011010        IILRLIRLILLRIRILLRLILIRIRII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
9.899495        0011100101011010        IIRRLIRLILLRIRILLRLILIRIRII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
9.899495        1111100101011010        IILRLIRLILLRIRILLRLILIRIRII
9.899495        1111100101011010        IILRLIRLILLRIRILLRLILIRIRII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
10.000000       1011100111011011        IILLRILRIRRLIRILLRLILIRILII
14.212670       1111010101111010        IILRLIRRILLRIRILRLRIRILILII
14.866069       1011101111110101        IILLRILRIRLRILIRLRRILILIRII

all energies and first found individual with that energy:
8.602325        1110110110010010        IILRLILRILLRILILLRRIRILILII
9.899495        0111100101011010        IIRLRILRIRRLILIRRLRIRILILII
10.000000       0111101101101010        IIRLRILRIRLRIRILRRLILIRIRII
...
17.804494       0110101111101011        IIRLRIRLILRLIRILRRLILIRILII
Inf     1011000010010101        IILLRILLILLLIRIRRLLIRIRILII

fittest individual in final population:
8.602325        1110110110010010        IILRLILRILLRILILLRRIRILILII

  LL
LIRI
IRRRL
LLI I
LIRRL
IIIL
LIIX
```

Man erkennt deutlich, dass hier bereits nach wenigen Iterationen eine sehr kompakte Konfiguration der Schlange erreicht wurde. Der Vergleich mit den Ergebnissen des Brute-Force-Skriptes zeigt, dass mit einer Energie von etwa 8.602 ein globales Optimum gefunden wird.

Energiefunktion E_b

Es werden die folgenden Parameter verwendet:

- `length = 27`
- `energy = energy_c`
- `size = 15`
- `crossover = 5`
- `mutation = 15`
- `flip = 0.3`

– selection = 0.5
– n = 10000
– energy = -15
– logging = true

Wir erhalten folgendes Ergebnis:

```
number of iterations:
71

final population:
-15.000000      0010111101111011        IIRRLILRILRLILIRLRLILIRILII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII
-10.000000      0110111101010010        IIRLRIRLIRLRIRILLRRIRILILII

all energies and first found individual with that energy:
-15.000000      0010111101111011        IIRRLILRILRLILIRLRLILIRILII
-13.000000      0110111101111011        IIRLRIRLIRLRIRILRLRIRILIRII
...
-1.000000       1101110100111110        IILRRILRILLRIRIRLRLIRILILII
0.000000        1011101011010011        IILLRILRIRLLIRILLRRIRILIRII

fittest individual in final population:
-15.000000      0010111101111011        IIRRLILRILRLILIRLRLILIRILII

  XIIL
IIR  I
 LR  RIL
 I     I
 LR   RL
  I  RL
  LR I
   LIL
```

An der gezeichneten Schlange erkennt man deutlich, dass eine Schleife mit großem Loch gefunden wurde. Bei dieser Energiefunktion ist das Ergebnis aufgrund der zerklüfteten

Energielandschaft (gut erkennbar an den Sprüngen in den Energien der Konfigurationen in der finalen Population) allerdings stärker von den zufälligen Startwerten abhängig. Entsprechend kann es sehr lange dauern – in ganz schlechten Läufen auch unendlich lange – bis das globale Optimum von -15 erreicht wird.