Übung 10

Ruman Gerst, 136994

June 21, 2014

Vergleich zwischen Baum- und Linear-GP

Das RNA-Faltungsproblem, das bereits als GP mit linearer Repräsentation implementiert wurde, wird mit einem ähnlichen Funktionsset als Baum-GP implementiert. Neben dem Wegfall einiger Funktionen bleibt der Funktionsumfang gleich.

Änderungen des ursprünglichen Programms

Das Programm wurde gegenüber den letzen Stand weiter verändert. So wurde die Bindungs-Funktion aktualisiert und bereinigt, sowie eine Unterstützung für das Einbringen mehrerer RNA-Sequenzen als Eingabe eingefügt.

Die Fitness eines Individuums ist nun die Summe der einzelnen Fitness-Werte der RNA-Sequenzen $F(I) = \frac{1}{\#(RNA)} \sum_{rna} fitness(I, rna)$ mit $fitness(I, rna) = \frac{energy(I,rna)+|stack|\cdot 14}{|rna|}$.

$energy(I, rna)$ ist die Energie der RNA als Sekundärstruktur nach dem Bindungs-Modell. $|stack|$ ist die Anzahl an nicht eingebauten Nukleotiden, die negativ bewertet wird. Multipliziert wird der Wert mit der negativen Energie, die ein G-C-Paar in die Struktur einbringt plus 2 (also schlechter bewertet).

Implementierung als Baum-GP

Der Funktionssatz bleibt mit Außnahme der RETURN-Funktion und der SKIP-Funktion der selbe. Eine Funktion, die das Individuum benutzt ist erst einmal eine Top-Level-Funktion, die einen Parameter hat. Im Gegensatz zum Linearen GP dürfen im Baum-GP Parameter wieder Funktionen mit Parameter sein. Der Parameter der Top-Level-Funktion wird zufällig gewählt. Dabei wird ein Baum von maximal 7 Tiefe erzeugt, was auch für jeden anderen Fall der Erzeugung eines zufälligen (Teil)-Baums zutrifft.

Mutiert und rekombiniert wird nach Koza. Bei der Mutation wird mit einer bestimmten Mutationswahrscheinlichkeit erst mutiert.

Tests

Getestet wird mit einer Generationszahl von 4000 Generationen mit zwei RNA-Sequenzen als Eingabe. Es gibt 3 ADF mit je 3 Parametern als Eingabe. Die Mutationswahrscheinlichkeit

beträgt $\frac{1}{20}$, die Rekombinationswahrscheinlichkeit $0,2$. Das lineare GP benutzt in der Hauptfunktion und in den ADF 20 Register.

Es werden die Ergebnisse der jeweiligen Testläufe mit folgenden Sequenzen betrachtet:

>G–C–RNA
GGGGGGGGGGGGGGGCGCGCCCCCCCCCCCC

>Äquivalente RNA mit A–U
AAAAAAAAAAAAAAAUAUAUUUUUUUUUUU

Lineares GP

Ein relativ guter Lauf beim linaren GP ergibt, trotz Einbeziehung von einer anderen Sequenz ein leichtes Overfitting, falls ein gutes Ergebnis herauskommt. Bei diesen Eingabesequenzen ist Overfitting nur schwer zu dämpfen, da ein gutes Ergebnis der G-C-Sequenz ein schlechtes Ergebnis der A-U-Sequenz überlagern kann.

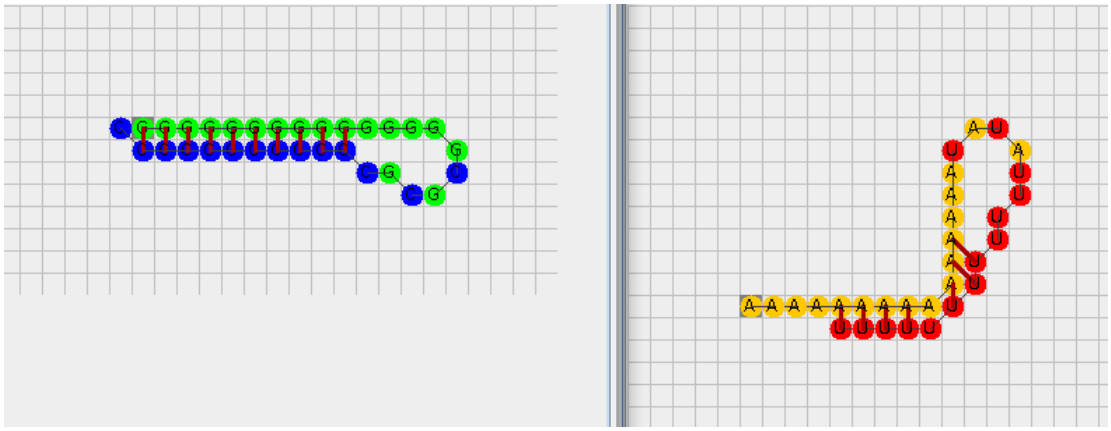

Figure 1: Beide Sequenzen bei einem linearen GP, Fitness -3.870967741935484

Der Code, der dabei entsteht lässt sich in eine Baum-Repräsentation bringen:

```
#ADF0 20 3
[...]

#ADF1 20 3
[...]

#ADF2 20 3
[...]

#MAIN 20 0
LOOK_FORWARD LEFT
R14
LOOK_FORWARD NORTH
R7
IF_EQUAL R17 NORTH R1 R19
```

```
AVG R9 RIGHT NORTH R5 R5
AVG EAST STACK SKIP1 SKIP5 R6
IF_GREATER ENERGY_OLD COLLIDE_LEFT R6 R5
IF_LESS LEFT SKIP4 SKIP1 R16
STRAIGHT
NUC RLEFT
SKIP3
SUBTRACT R4 R13
IF_EQUAL R10 RLEFT R18 R2
SOUTH_WEST
R16
IF_LESS LENGTH R6 SKIP9 R18
CALCULATE_ENERGY R19 R2
IF_GREATER RRIGHT RSTRAIGHT SKIP6 RRIGHT
AVG SOUTH PREV R5 LENGTH R17
```

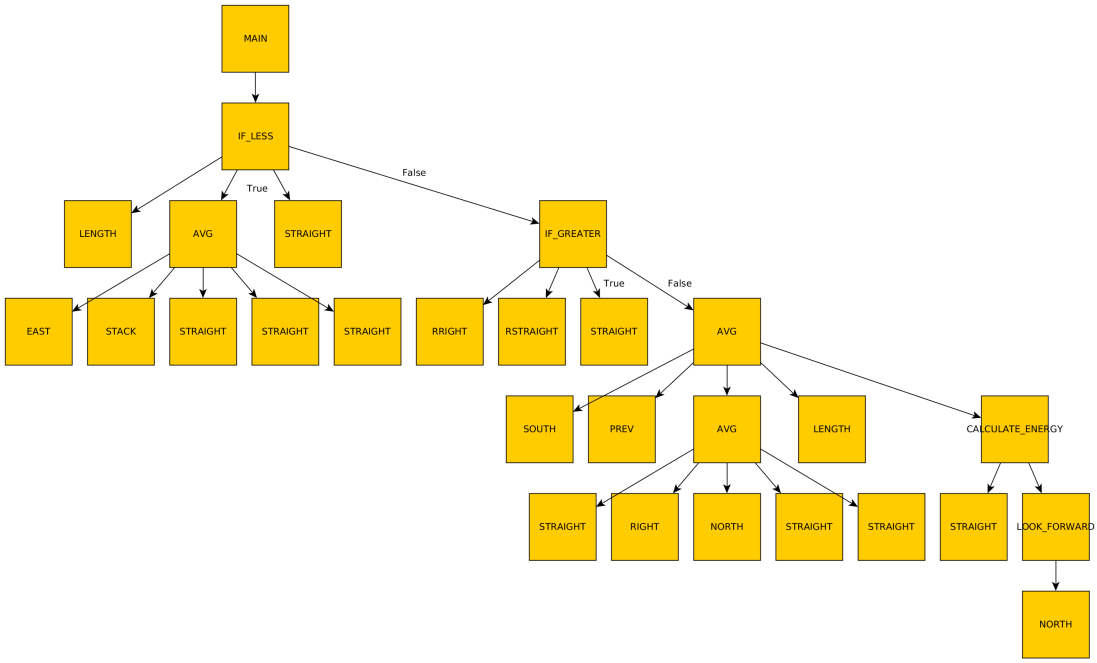

Figure 2: Aquivalente Baum-Repräsentation des Codes des Linear-GP. Vorwärts-Referenzen wurden durch STRAIGHT (=0) ausgetauscht.

Baum-GP

Wenn der Baum-GP das lokale Minimum von der Optimierung der GC-Sequenz überwindet, dann werden sowohl AU-Sequenz, als auch GC-Sequenz gleich gut optimiert.

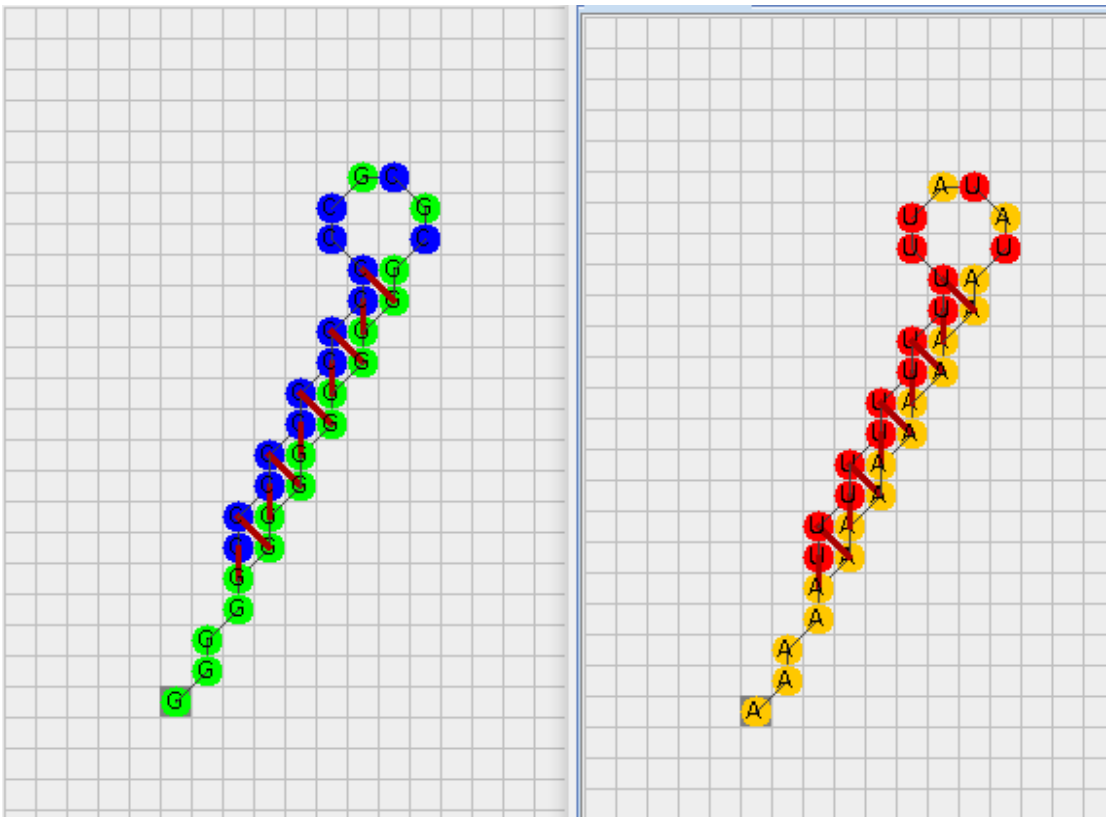

Figure 3: Beide Sequenzen bei Benutzung einer Baum-GP, Fitness -5.032258064516129

Der Baum, der ausgerechnet wurde, ist relativ komplex und groß; dafür schafft er es, beide Sequenzen gleich gut zu optimieren.

Erzeugter Baum

```
ADF0: (ADF0 (P1))
ADF1: (ADF1 (SOUTH_WEST))
ADF2: (ADF2 (EAST))
MAIN: (MAIN (PURINE (MAX (RSTRAIGHT) (SOUTH) (MIN (COLLIDE_LEFT) (
    SOUTH_EAST) (STACK) (G) (NORTH)) (SOUTH) (MIN (IF_LESS (
    COLLIDE_LEFT) (ADF0 (COLLIDE_LEFT) (C) (ADF1 (GETBOND (STRAIGHT)
    (WEST)) (NUC (SOUTH_EAST)) (EAST))) (RIGHT) (LOOK_FORWARD (ADF1 (
    LOOK_FORWARD (RSTRAIGHT)) (STACK) (STACK)))) (MAX (SOUTH) (
    STRAIGHT) (RSTRAIGHT) (LEFT) (C)) (COLLIDE_LEFT) (EAST) (RLEFT)))
    ))
```

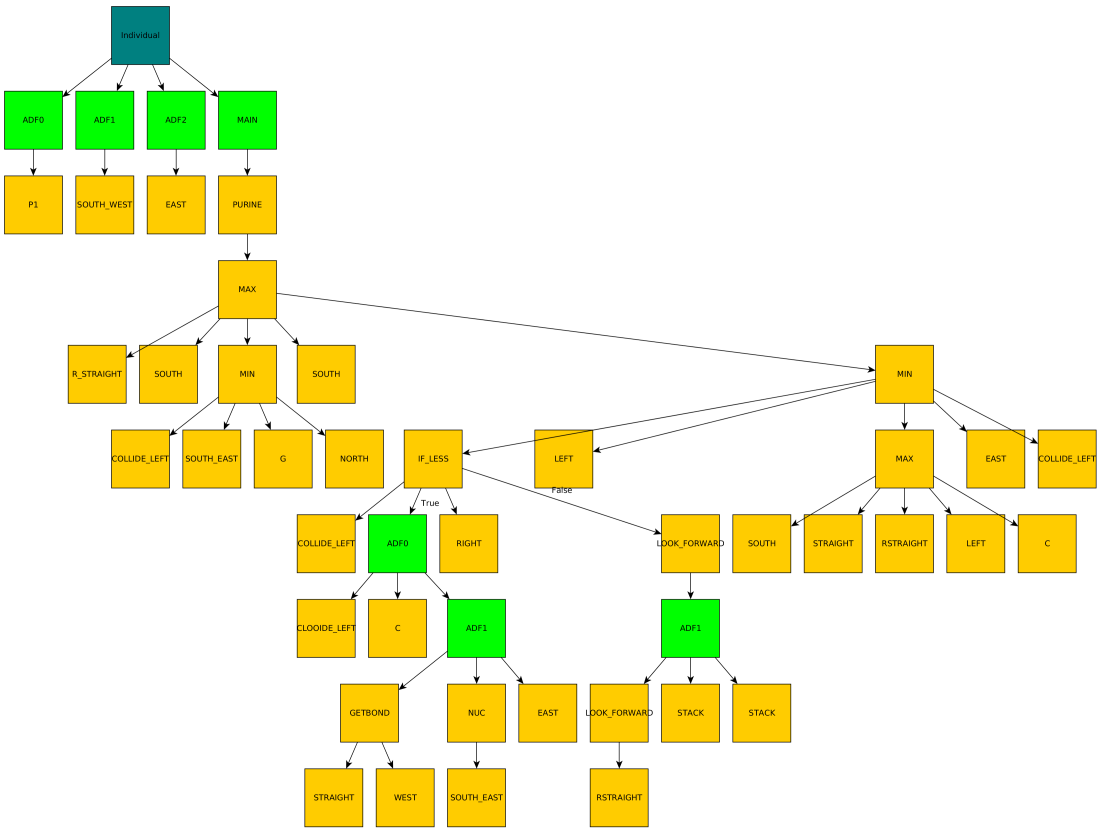

Figure 4: Graphische Repräsentation des Baums, der vom Baum-GP erzeugt wurde.

Diese Baum-Struktur lässt sich in ein äquivalentes lineares Programm umschreiben:

#ADF0:
R0: P1

#ADF1:
R0: SOUTH_WEST

#ADF2:
R0: EAST

#MAIN:
R0: MAX SOUTH STRAIGHT RSTRAIGHT LEFT C
R1: LOOK_FORWARD RSTRAIGHT
R2: ADF1 R1 STACK STACK
R3: LOOK_FORWARD R2
R4: NUC SOUTH_EAST
R5: GETBOND STRAIGHT WEST
R6: ADF1 R5 R4 EAST
R7: ADF0 COLLIDE_LEFT C R6
R8: IF_LESS COLLIDE_LEFT R7 RIGHT R3
R9: MIN R8 R0 COLLIDE_LEFT EAST RLEFT
R10: MIN COLLIDE_LEFT SOUTH_EAST STACK G NORTH

R11 : MAX RSTRAIGHT SOUTH R10 SOUTH R9
R12 : PURINE R11

Vergleich der Ergebnisse

Beim Vergleich zwischen dem linearen und dem Baum-GP wird deutlich, dass das lineare GP viele redundate Register hat. So hat das Baum-GP, das ein viel besseres Ergebnis hat, einen großen Baum, benutzt aber effektiv nur 15 Register, wobei 3 von diesen sogar weggelassen werden können (Die ADF haben nur eine Funktion). Wenn man nun ein lineares Programm in einen Baum umwandelt, so ist dieser Baum im Verhältis zur Anzahl der Register sehr klein.

Bäume haben also weniger Redundanzen als lineare Programme und können zusätzlich theoretisch beliebig groß werden, wobei bei der Implementation als Baum-GP ähnliche Verhaltensweisen der Optimierung auftreten. So benötigt auch beim Baum-GP der Algorithmus eine 'Anlaufzeit', um die erste Optimierung auszuwürfeln. Auch ein Baum-GP kann in einem lokalen Minimum gefangen sein, wobei dies bei den Tests etwas weniger auftrat als beim linearen GP.

Es ist für dieses Problem also eher sinnvoll ein Baum-GP zu verwenden, besonders, da durch die theoretisch nicht vorhandene Größenbeschränkung der Baum sehr groß werden kann. Gleichzeitig kann dieser aber auch bei kleinen Problemen sehr klein bleiben.