Übung 9

Ruman Gerst, 136994

June 14, 2014

Problem

Es soll ein Programm gesucht werden, das durch inkrementelles Aufbauen eine RNA-Sekundärstruktur aus einer Sequenz herstellen soll. Dabei soll das Programm auf die bereits vorhandene Struktur lesend zugreifen können und auch die noch zu hinzufügenden Nukleotide benutzen können.

Umsetzung

Individuum

Ein Individuum $(S, Q, g_{main}, G_{adf}, F)$ besteht aus einer Struktur S, einem Queue Q, einer Main-Funktion g_{main}, einer Menge an ADF G_{adf} und der Fitness F.

Bei der Bewertung des Individuums wird zuerst die RNA-Sequenz in den Queue gelegt und für maximal 10 mal für jedes Nukleotid in der Sequenz g_{main} ausgeführt. Funktionen geben eine Zahl in $\mathbb{Z}$ (Integer) zurück, die in eine relative Richtung (Links, Rechts, Geradeaus) gewrappt wird.

Die Funktion hierfür ist $w(x) = \begin{cases} LEFT & (x+1) \, mod \, 3 = 0 \\ STRAIGHT & (x+1) \, mod \, 3 = 1 \\ RIGHT & (x+1) \, mod \, 3 = 2 \end{cases}$.

Entsprechend dieser Richtung wird das nächste Nukleotid aus Q genommen und an die bestehende Struktur angehängt (Mit Aktualisierung der Bindung, ...).

Für die Struktur S gibt es eine Energiefunktion $E(S) = \sum_n e(n)$, die die Summer über die Energien aller Nukleotide bildet. Diese wiederum betrachtet die

Bindung zu den anderen Nukleotiden $e(n) = \begin{cases} 2 & wenn\, keine\, Bindung \\ -6 & wenn\, G-C \\ -4 & wenn\, A-U \\ -2 & wenn\, G-U \\ 4 & wenn\, Konflikt \end{cases}$

Die Bindungsfunktion stellt bevorzugt Bindungen vom Typ "Konflikt" her, um falsch nebeneinanderliegende Nukleotide zu verhindern. Die Bewertung von +4 zur Energie ist durch Ausprobieren zustandegekommen und ist so niedrig gewählt, um die Wahrscheinlichkeit zu verringern, dass die Struktur in einem lokalen Minium endet.

Die Fitness eines Individuums ist dann $F(G) = \frac{(E(S)+|Q|^2)}{|S|}$. Diese Funktion bewertet die übriggelassene Sequenz in Q schlecht und teilt das Ergebnis durch die Länge der Struktur, um das Ergebnis besser mit anderen RNA vergleichen zu können.

Funktion

Eine Funktion (R, T_{dyn}, P_{dyn}) besteht aus einer Menge an Registern R und aus Mengen an dynamischen (also zu dieser Funktion gehörigen) Terminalfunktionen und Parameterfunktionen. Ein Register ist entweder eine Terminalfunktion oder eine Parameterfunktion, die wieder Terminalfunktionen als Parameter hat (aber keine Parameterfunktionen!).

Neben den dynamischen Parameterfunktionen und Terminalfunktionen gibt es je eine Menge an statischen Terminal- und Parameterfunktionen, die in jeder Funktion vorkommen können. Wenn eine Terminal- oder Parameterfunktion zufällig gewählt wird, ist jedes Element gleichwahrscheinlich.

Beispiele für dynamische Terminalfunktionen sind P1, P2, ... die Parameter, die einem ADF übergeben werden und ein Beispiel für eine dynamische Parameterfunktion ADF0 <P1> <P2> - ein ADF in der Main-Funktion.

Jede Funktion kann durch RETURN <P1> einen Wert zurückgeben oder es wird das letzte Register als Ausgaberegister benutzt.

Register

Ein Register (L, P, v) besteht aus einem Label L (Name der Parameterfunktion oder eine Terminalfunktion), einer Menge an Parametern (Terminalfunktionen), sowie $v \in \mathbb{Z}$, dem aktuellen Wert des Registers. Vor Ausführung der Funktion wird $v = 0$ gesetzt.

Mutation und Rekombination

Es werden immer zwei Individuen mit 2-Punkt-Crossover rekombiniert. Wenn rekombieniert wird, werden die Register der entsprechenden Funktionen rekombiniert.

Bei der Mutation der Register wird entweder das komplette Register durch ein zufälliges ausgetauscht oder ein zufälliger Parameter. Dabei wird die komplette Mutation als "zusätzlicher" Parameter gezählt. Wenn dann dieser "Parameter" ausgewählt wird, dann wird das komplette register mutiert. Sonst wird nur ein Parameter durch eine zufällige Terminalfunktion ausgetauscht.

Ergebnisse

Es werden mit unterschiedlichen RNA-Sequenzen Tests durchgeführt. Die Generationszahl beträgt $M = 2000$, mit Turnierselektion (10 Turniere) werden aus der Population von 800 Individuen 200 mal 2 Eltern ausgewählt. Diese erzeugen dann mit Hilfe von Rekombination und Mutation zwei Kinder. Die Eltern werden nicht in die neue Population übernommen.

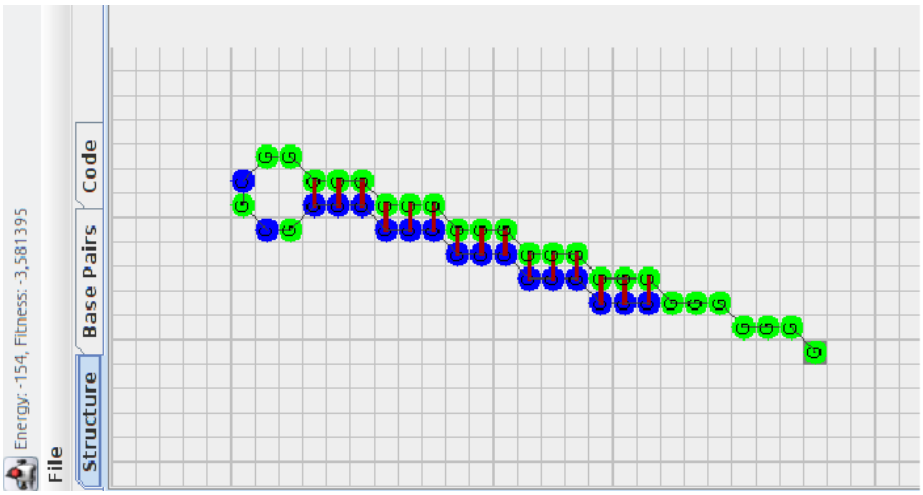
In den folgenden Test wurde mit der angegebenen Sequenz 10 mal ein Programm evolviert. Das beste und das schlechteste Ergebnis, sowie der Code werden mitangegeben.

Einfache GC-RNA

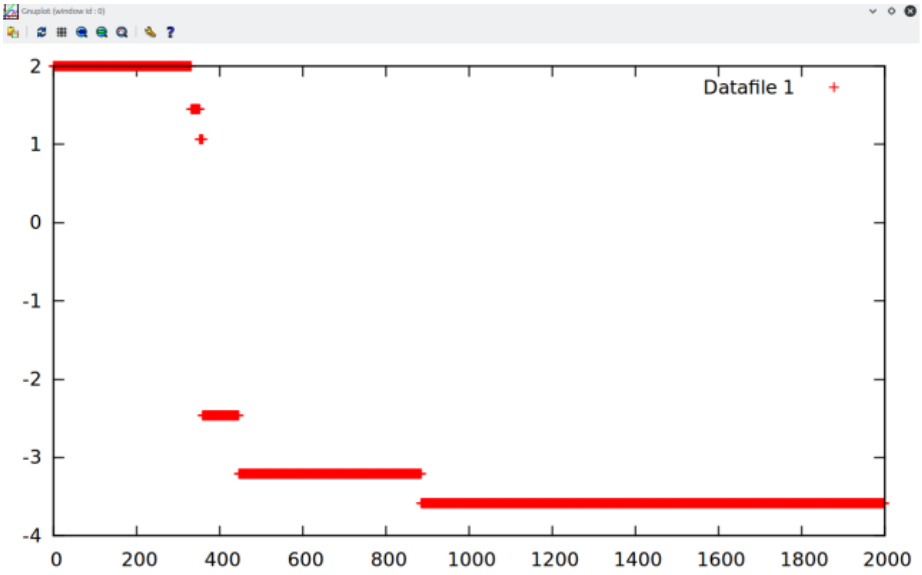
Folgende RNA-Sequenz wurde getestet:

GGGGGGGGGGGGGGGGGGGGGGGCGCGCCCCCCCCCCCCCCC

Das Programm ist dazu in der Lage, eine optimale Sekundärstruktur mit -154 Energie zu finden.



(a) Struktur



(b) Entwickung der Fitness des besten Individuums über die Generation

Figure 1: Bestes Ergebnis der GC-RNA

```
>Individual with Fitness -3.5813953488372094
GGGGGGGGGGGGGGGGGGGGGGGGGCGCGCCCCCCCCCCCCCCCC
#ADF0 20 5
IF_GREATER STACK STRAIGHT R7 R2
R7
P0
```

```
R11
R2
STACK
NUC SKIP3
PURINE P0
STRAIGHT
R13
COLLIDE_RIGHT
R19
R6
ADD SKIP5 SKIP2
ENERGY
DIVIDE SOUTH P1
RSTRAIGHT
P2
CALCULATE_ENERGY R15 C
PYRIMIDINE C
#MAIN 20 0
CALCULATE_ENERGY RRIGHT STRAIGHT
IF_GREATER R11 R8 SKIP6 LEFT
NORTH_WEST
LENGTH
IF_LESS COLLIDE_STRAIGHT SKIP3 SKIP5 SKIP2
R11
ADD STACK SKIP1
IF_LESS R8 R15 RIGHT R3
ADF0 RSTRAIGHT PREV COLLIDE_STRAIGHT R15 A
R8
RETURN RIGHT
R15
MULTIPLY SOUTH_WEST SKIP1
PURINE NEXT
ADF0 A R6 COLLIDE_RIGHT SKIP1 U
MULTIPLY RIGHT R17
RSTRAIGHT
R11
IF_LESS R8 R19 R7 G
IF_GREATER SOUTH SKIP8 R13 RLEFT
```

Das schlechteste Ergebnis kommt daher zustande, da fehlende Nukleotide sehr hart bestraft werden und es somit günstiger ist, einfach eine gerade Struktur zu erzeugen, als eine, die durch eine falsche Richtung schon sehr früh abbricht.

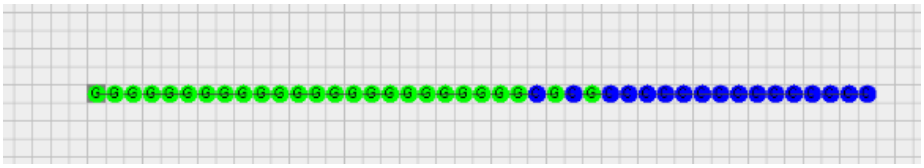


Figure 2: Schlechtestes Ergebnis der GC-RNA

```
>Individual with Fitness 2.0
GGGGGGGGGGGGGGGGGGGGGGGGGCGCGCCCCCCCCCCCCCCCC
#ADF0 20 5
COLLIDE_RIGHT
RRIGHT
R11
G
IF_LESS R8 STRAIGHT R0 SKIP1
R7
NUC SOUTH
DIVIDE U SKIP8
GETBOND PREV RRIGHT
DIVIDE SKIP9 R6
WEST
ENERGY
NUC R10
LEFT
SKIP7
DIVIDE EAST R12
R0
EAST
LOOK_FORWARD R5
ADD R7 NORTH
#MAIN 20 0
COLLIDE_LEFT
COLLIDE_RIGHT
ADF0 R2 EAST R11 R6 RSTRAIGHT
CALCULATE_ENERGY U STRAIGHT
IF_GREATER SOUTH_EAST U R13 SOUTH_WEST
ADF0 R4 RRIGHT PREV R8 R6
IF_EQUAL R1 SKIP8 SOUTH_EAST SKIP6
IF_LESS SOUTH_WEST NORTH R5 SKIP8
NORTH_EAST
RSTRAIGHT
ADF0 SOUTH_EAST NORTH_EAST SKIP2 ENERGY_OLD RRIGHT
IF_LESS R9 COLLIDE_STRAIGHT LENGTH R13
IF_LESS R3 LENGTH NORTH R6
```

ADF0 R11 G R11 R0 R3
GETBOND NEXT R18
A
SKIP7
SUBTRACT R5 RLEFT
G
SKIP5

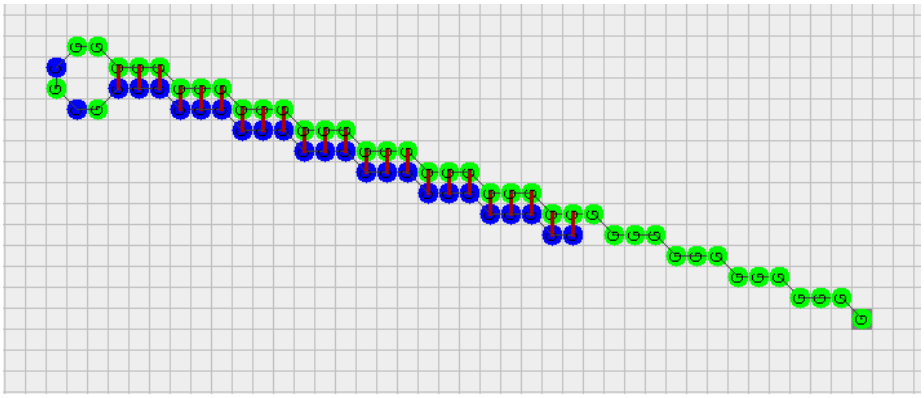
Folgende Fitness- und Energiewerte wurden während des GC-RNA-Tests erreicht:

Test	1	2	3	4	5	6	7	8	9	10
Fitness	-2,47	-3,58	-1,63	-1,72	-2,84	-3,58	-2,09	2,0	-0,26	-2,46
Energie	-106	-154	-70	-74	-122	-154	-90	86	-12	-106

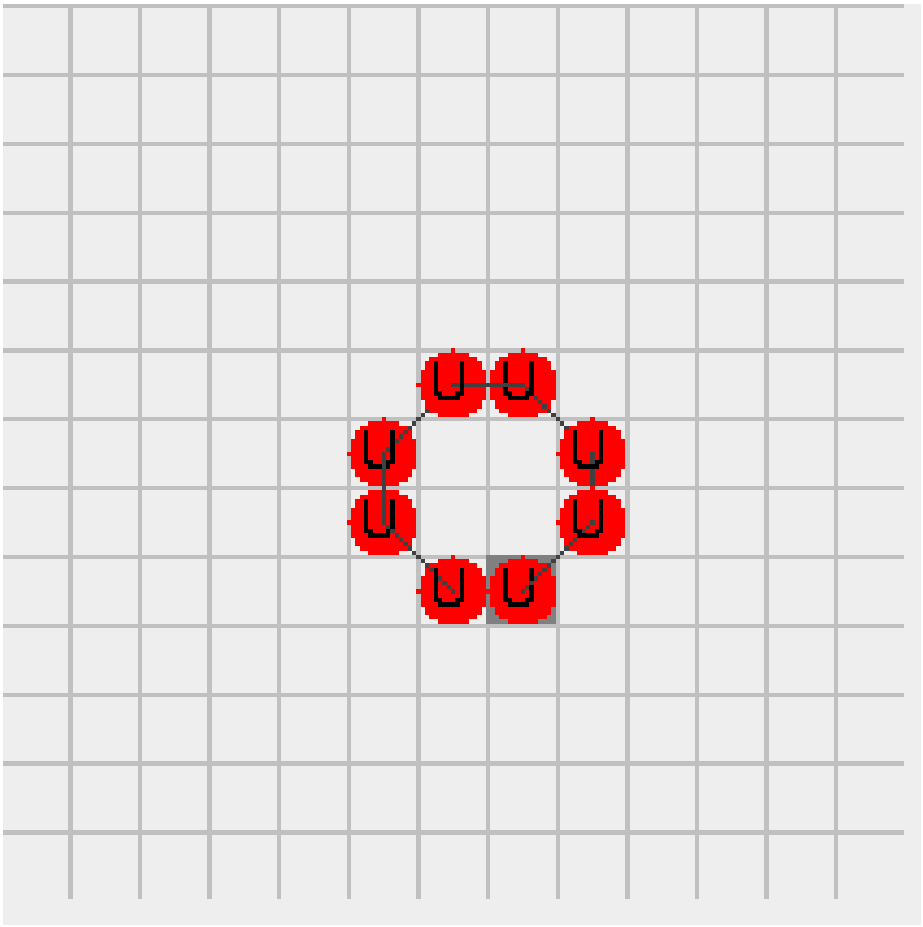
Table 1: Fitness- und Energiewerte des GC-RNA-Tests, $\emptyset_{fitness} = 1,86$

Lernt das Programm?

Um zu testen, ob das Programm auch mit anderen Strukturen umgehen kann, wird die RNA-Sequenz in den *.RNASLV-Dateien umgeändert und das Programm ausgeführt:



(a) Es wurden einige G's und C's zur Sequnz hinzugefügt.



(b) G-C wurde durch A-U ausgetauscht

Figure 3: Verhalten des evolvierten Programms bei veränderter Sequenz

Es wird sehr deutlich, das das Programm nur mit Änderungen in dem vom Programm gelernten "Schema" umgehen kann. Schon eine kleine Änderung, die aus dem Schema ausbricht, führt dazu, das das Programm versagt.

Einfacher Hairpin

Folgende Sequenz wurde getestet:

ACUCGGUUACGAG

Es liegt eine originale Struktur vor, die zeigt, wie das Ergebnis aussehen soll:

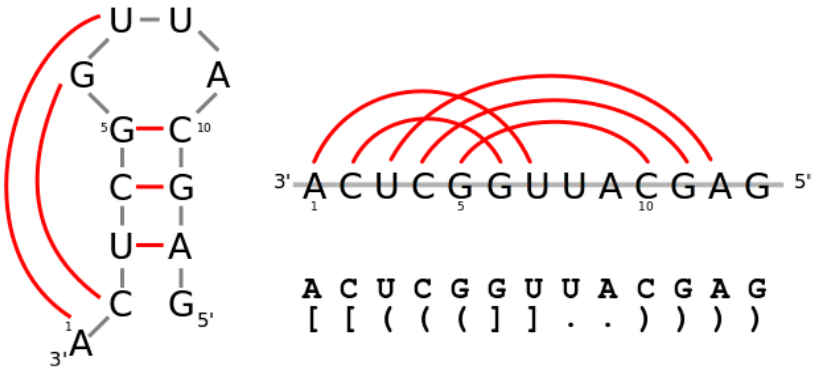
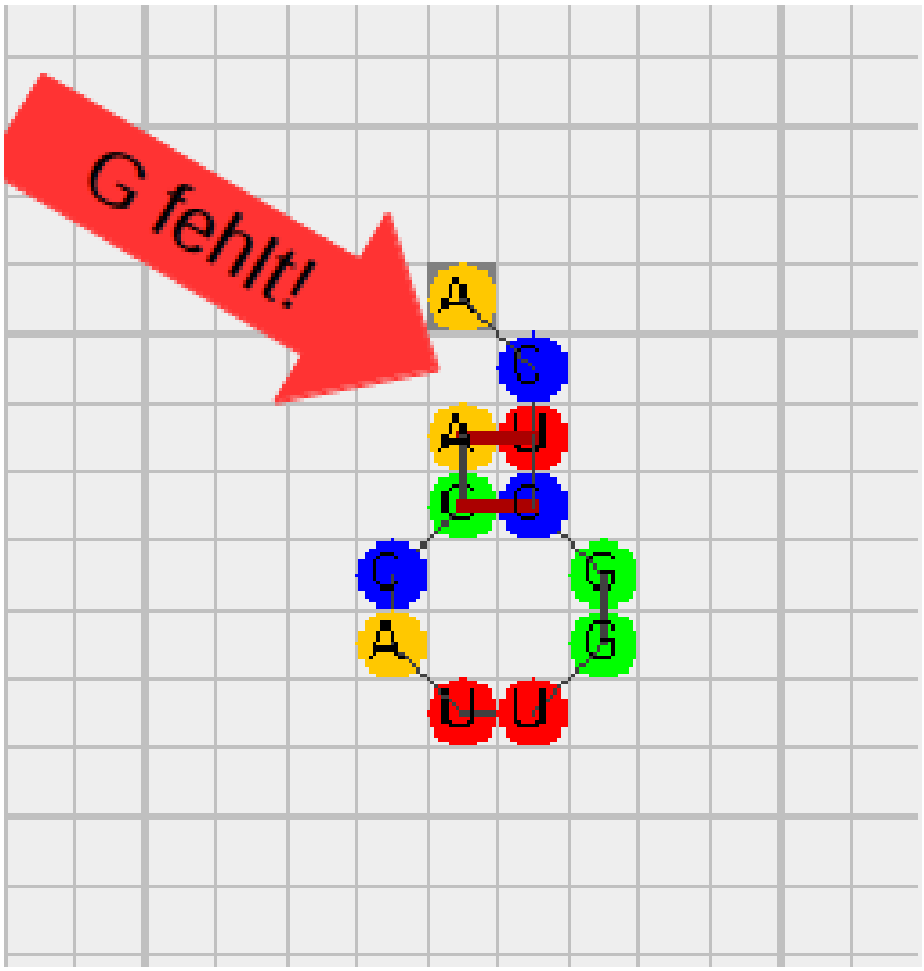
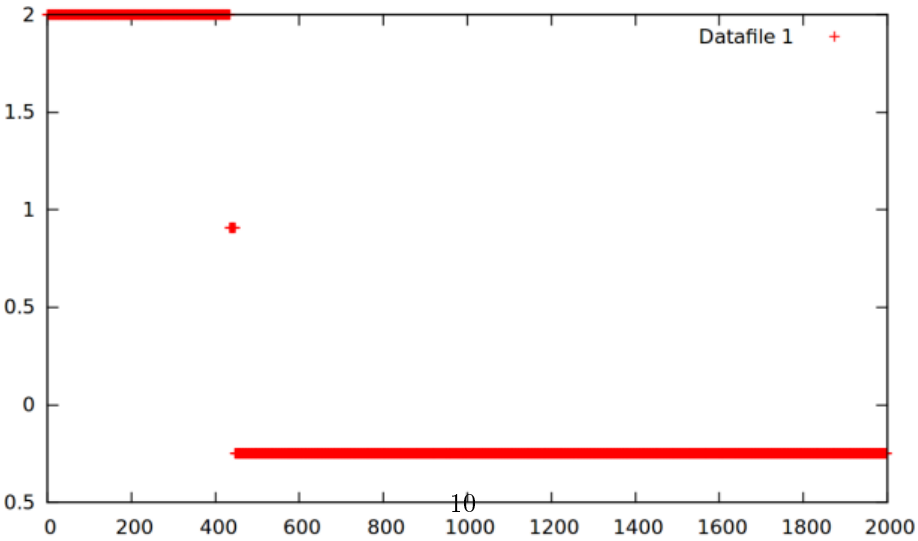


Figure 4: Originale Struktur

Es konnte nur ein Programm gefunden werden, das wenigstens 2 der 3 Basenpaare erzeugt. Das letzte Nukleotid (G) wird von dem Programm verschluckt.



(a) Struktur - es fehlt das letzte G



(b) Entwicklung über die Generationen des besten Individuums

Figure 5: Bestes Ergebnis des Hairpins

```
>Individual with Fitness -0.25
ACUCGGUUACGAG
#ADF0 20 5
MAX SKIP1 R15 P4 NORTH P0
MIN R6 R14 LEFT P1 ENERGY
R0
MULTIPLY COLLIDE_RIGHT U
R10
SUBTRACT R5 U
IF_GREATER R19 SKIP7 SKIP4 R18
SKIP6
RDIR R3
GETBOND R3 SKIP1
IF_LESS EAST R11 SKIP6 C
MAX P4 RLEFT R10 C RLEFT
R4
P4
SUBTRACT NORTH_EAST R16
IF_EQUAL R18 COLLIDE_LEFT R13 SKIP6
COLLIDE_RIGHT
P1
IF_GREATER STRAIGHT R16 SOUTH_EAST RRIGHT
RLEFT
#MAIN 20 0
NUC G
DIVIDE R18 LENGTH
R1
LOOK_FORWARD NORTH
AVG SKIP3 R2 SKIP2 SKIP9 R10
NORTH_WEST
STACK
R9
GETBOND SKIP9 ENERGY_OLD
IF_LESS R10 SKIP4 SKIP4 STACK
R16
MIN R12 SOUTH ENERGY SOUTH_EAST SOUTH_EAST
PREV
AVG R17 SOUTH COLLIDE_LEFT R8 R4
CALCULATE_ENERGY R3 R12
MAX A SOUTH_EAST R3 R14 NORTH_EAST
SKIP4
DIVIDE C NORTH_WEST
LOOK_FORWARD R8
MAX R3 SKIP4 SKIP8 RSTRAIGHT C
```

Die schlechteste Struktur hat sich in einem lokalen Minimum verfangen, außerdem wurde das 'aneinanderliegen' der zwei Adenosine nicht negativ bewertet.

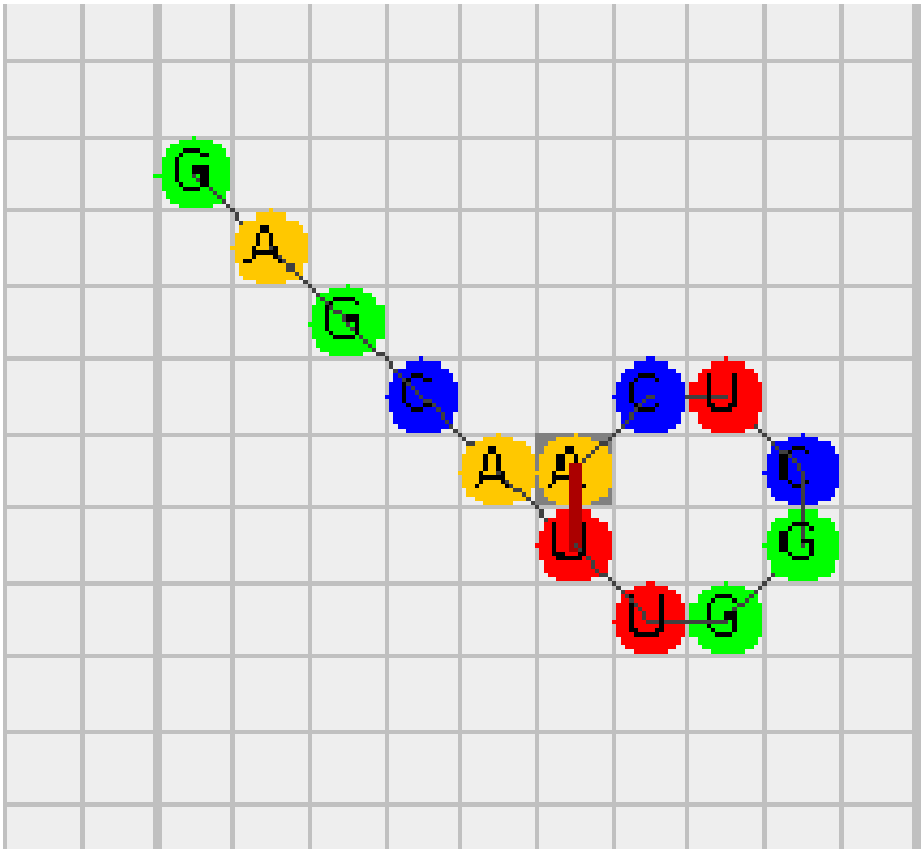


Figure 6: Schlechtestes Ergebnis des Hairpins

```
>Individual with Fitness 1.0769230769230769
ACUCGGUUACGAG
#ADF0 20 5
ADD ENERGY SOUTH_WEST
NORTH_WEST
SUBTRACT COLLIDE_STRAIGHT SOUTH_EAST
PURINE R17
MIN RSTRAIGHT R13 WEST SKIP4 SKIP5
AVG R13 SKIP5 SOUTH_WEST COLLIDE_LEFT SKIP8
MIN SKIP4 SOUTH_WEST P3 RSTRAIGHT SKIP7
ADD NORTH RSTRAIGHT
LOOK_BACK R6
R16
```

```
MIN ENERGY STACK STRAIGHT R4 R10
IF_LESS R16 U ENERGY_OLD C
P3
RDIR R18
AVG R0 R10 R0 P0 STRAIGHT
IF_EQUAL R17 NORTH_WEST ENERGY R13
IF_EQUAL R18 R2 SOUTH R16
DIVIDE SKIP8 SKIP4
R13
R1
#MAIN 20 0
U
ADF0 SKIP8 SKIP8 RSTRAIGHT RLEFT ENERGY
AVG SOUTH_EAST R2 SKIP6 R13 SKIP7
MIN ENERGY_OLD COLLIDE_STRAIGHT SKIP9 STACK SKIP9
R11
SKIP9
R1
LEFT
IF_GREATER R0 SOUTH R17 R9
IF_LESS SOUTH R13 RIGHT R18
SKIP6
NORTH
SKIP4
EAST
MIN R12 SKIP1 R4 R13 R9
R1
COLLIDE_RIGHT
SKIP3
IF_EQUAL SOUTH_EAST RLEFT R13 SKIP8
IF_GREATER SKIP8 PREV NORTH SKIP7
```

Lernt das Programm?

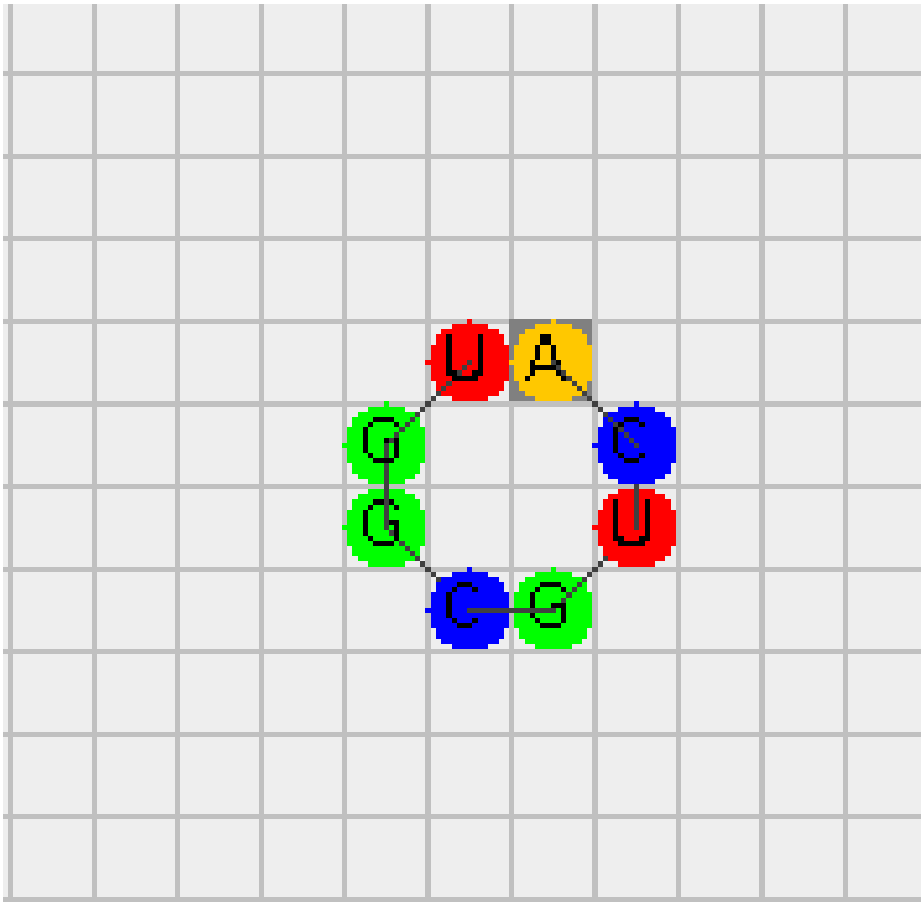


Figure 7: Verhalten auf Hinzufügen eines weiteren G-C-Stacks (G nach der 3., C nach der 12. Base)

Ähnlich wie beim GC-Test versagt das Programm hier.

Conclusion

Wie man an den Ergebnissen sehen kann, wird selbst bei einfachen Strukturen selbst nach 2000 Generationen nicht immer ein Optimum gefunden (Beim GC-Test sind nur zwei der besten Individuen nach 2000 Generationen im Optimum von -154 Energie!), obwohl zumindest die meisten Individuen bereits in einem

lokalen Minium sind. Wenn nun noch das "Lernen" des Programms betrachtet wird, indem getestet wird, wie es auf Veränderungen der Sequnz reagiert, wird klar, dass die evolvierten Programme hier nur einen kleinen Spielraum haben.

Beim G-C-Test können z.B. beliebig viele G's und C's hinzugefügt werden, solange sie ins gelernte Schema passen.

Eine Lösung hierfür kann sein, nicht nur eine Sequenz zu betrachten, sondern nur Individuen zu nehmen, die auf mehreren Sequenzen gute Ergebnisse bringen. Die Evolvierung eines solchen Programms muss sorgfältig ausgeklügelt werden, da sonst eventuell gute Teillösungen verschwinden.

Zusammenfassend funktioniert die Lösung des Problems mit der aktuellen Umsetzung aber nicht.