Algorithm Engineering-Projekt

String Alignment mit Needleman-Wunsch

Tom Wegener, 18INM/TZ

2019-02-10

Inhaltsverzeichnis

1 Einleitung

Dieses Projekt zum Thema Sequenz-Alignment entstand im Laufe des Moduls "Algorithm Engineering" des Master-Studiengangs Informatik der HTWK Leipzig. Es war ein modulbegleitendes Projekt in dem ein Algorithmus ausprogrammiert werden sollte.

1.1 generelle Problembeschreibung

Am Anfang des Semesters sollte sich für ein Projekt entschieden werden, dieses dann in vier Schritten über den Zeitraum des Semesters ausprogrammiert werden. Zu diesen vier Schritten gehört jeweils eine Abgabe.
Zuerst sollte ein Parser entwickelt werden, der die ausgewählten Daten ausliest und abspeichert. Anschließend eine erste Version des Algorithmus ausprogrammiert werden und erste Laufzeiten gemessen werden, die anschließend auch graphisch dargestellt werden. Aus den Laufzeiten sollten dann als dritte Aufgabe Optimierungsmöglichkeiten aufzeigen. Außerdem wurde die Verwendung eines Profilers empfohlen. In der vierten Aufgabe werden dann zwei Algorithmen auf unterschiedliche Art und Weise verglichen.

1.2 spezielle Problembeschreibung

In dieser Arbeit wird das Problem String-Alignment behandelt. Die Daten werden aus der Datenbank des "National Institute of biotechnical Information"(ncbi) in Form von Fasta-Dateien entnommen und durch einen Parser eingelesen.
Anschließend wird ein Wert ausgegeben, der die Ähnlichkeit von zwei Sequenzen angibt. Dafür gibt es verschiedene Algorithmen.
Das Projekt wurde zuerst teilweise in Python3 umgesetzt und anschließend in go (bzw. golang) übersetzt und fertig gestellt, um die Laufzeiten zu verkürzen.

1.3 Charakteristika der Eingabedaten

Die Fasta-Files haben den Aufbau, dass sie mehrere Blöcke an DNA-Sequenzen enthalten, jeder Block wird über ein ">" eingeleitet, dahinter steht dann eine Beschreibung, die auch eine ID enthält. Die nachfolgenden Zeilen enthalten die DNA-Strings auf mehrere Zeilen verteilt.

```
>ref|XP_021694431.1| gametogenetin-binding protein 2-like [Aedes aegypti]
MAKLTYVYRSDEMNCVKVSKRQLPLIGGENLMMLMDLNSRGLVFDQPPVKGQELDDFAKKYRVLTPAELR
LSLNVPTIEFTSVLSQNVPCVGCRRSVERLFYQLMLSGHPTLDPIVITGRGVLTISEDKMKSPQTLCTLL
HKHKLVLDELLDNQCRNRKNLRCNLHSLDTFRSRPFSETWRDVWNCMKQQCKDELAVIESSELHTMLDGY
LKKHKFCQECRTKVEKAYSLLVHESNPAKEKGYVAHLYSGIKRCLSDKHIHLQTKLEYIDSLIKRAEPEL
NGRNSKHRERHAKTLEIAQEEVLTCIGMCLYERLRRISVCLREEENACQVLAAVAVHALSRSFDMAVERK
QGISNLELLYEEISREERSKEHKKEQKKLKKRRKRNEKKLIDNTSDKAVECSDETDSKLCSCSPDDNEED
DEEADDRVMLCDGTIIDAGPKSTITINRNEADTSKMQIISCSSCEMSNIDKFSNTNICTRSSFDGGYASE
PLQSESLHTSSHMDSTTSSLVSTPEGSEIACSDGLCNHGGSIHSKNRYAPFSNTSFFGTMRSPNMLLNSM
SGLPMTLQEMLDKSSTEDDDAENDVIPNECILEFKSRSNVIKKQREALRQQLLNNFKQLCVKHCKKEDSK
VD
```

Zusätzlich werden zufällige Sequenzen generiert, die einen zehn Zeichen langen Namen haben, während die Sequenz, die verglichen wird, eine veriierende Länge zwischen 0 und 4900 hat.

1.4 Messumgebung

Getestet wurden zwei Fasta-Dateien, Aedes Aegypti und Aedes Albopictus. Zusätzlich werden bei Programmstart mehrere String zufällig generiert.
Programmiert wurde zuerst in Python3 mit der Version 3.6.7 und anschließend in go (golang) mit der Version 1.10.4. Ausgeführt wurde der Code hauptsächlich auf einem Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz mit 16GB Arbeitsspeicher. Das Betriebssystem ist eine Linux-Distribution mit Ubuntu 18.10 als Basis.
Bei jedem durchlaufen werden zwei Dateien aus dem data-Ordner, die in der Konfigurations-Datei angegeben wurden, eingelesen.

2 Algorithmen und Optimierungen

Als erster Algorithmus wurde der Needleman-Wunsch-Algorithmus (NWA) gewählt, der die Laufzeit O(mn) besitzt. Zusätzlich wurde ein splitted-Needleman-Wunsch (SNWA) hinzugefügt, der jedoch nur eine Annäherung an den Score bietet.
In der Konfigurationsdatei (config.yml) können diese durch ihre Abkürzungen (nwa, snwa) eingestellt werden. Zu beachten ist dabei, dass diese klein geschrieben werden müssen.

2.1 Needleman-Wunsch

Der NWA hat eine Laufzeit von O(mn) und ergibt einen genauen Score. Der Algorithmus läuft so ab wie im folgenden Pseudo-code beschrieben, zusätzlich wird eine Matrix angelegt in der die Herkunft des Wertes zusätzlich gespeichert werden.

```
match = 1
mismatch = -1
gap = -1
for i=0 to length(A)
  F[i,0] = gap*i
for j=0 to length(B)
  F[0,j] = gap*j
for i=1 to length(A)
  for j=1 to length(B)
  {
    if (A[i] == A[j])
    {
      score = 1
    } else {
      score = -1
    }
    Match = F[i-1,j-1] + score
    Delete = F[i-1, j] + d
    Insert = F[i, j-1] + d
    F[i,j] = max(Match, Insert, Delete)
  }
```

Im Quelltext wird dieser Algorithmus durch die Funktion nwa dargestellt und kann in der Konfigurationsdatei (config.yml) unter dem label algorithm eingestellt werden.

2.1.1 Optimierungen

Der Algorithmus wurde zuerst in Python3 ausprogrammiert, jedoch wurden dort sehr hohe Laufzeiten für den Vergleich von mehreren Sequenzen erreicht, weshalb das Programm nach go (golang) umgeschrieben wurde.
Im NWA wird das Maximum von drei Zahlen benötigt. In einigen Programmiersprachen wird eine Funktion, die das Maximum zurückgibt über eine Bibliothek bereitgestellt. Bei go muss das Maximum selber berechnet werden. Dafür gibt es zwei Möglichkeiten, einerseits kann man bei einer unbekannten Menge von Zahlen von denen das Maximum zurückgegeben werden soll, über ein Array dieser Zahlen iterieren. Andererseits kann man, wenn die Menge der Zahlen bekannt und niedrig ist, mehrere bedingte Anweisungen nutzen. Die bedingten Anweisungen haben eine deutlich geringere Laufzeit, da der Zugriff auf Arrays aufwendiger ist als der Vergleich von Integer-Zahlen.

2.2 Splitted Needleman-Wunsch

Der SNWA hat eine Laufzeit zwischen O(mn) und O(mn/2), ergibt aber keinen korrekten Score, sondern nur eine Annäherung an einen Score.
Die Optimierung basiert auf der Annahme, dass an einer Stelle innerhalb des Vergleiches der Sequenzen der Wert diagonal ermittelt wird. Dafür werden die gleichen Buchstaben bei beiden Sequenzen benötigt. Durch die diagonale Ermittlung des Vegleiches werden die weiteren Werte nicht mehr benötigt.
Für den Algorithmus werden die Sequenzen an einem gleichen Buchstaben möglichst nah zur Mitte geteilt. Anschließend wird der NWA auf beide Hälften ausgeführt und die beiden Scores addiert. Dadurch

wird im Optimalfall die Laufzeit halbiert, da nur noch die Hälfte der Matrix berechnet werden muss. Theoretisch kann dieser Algorithmus noch verbessert werden, sodass die Sequenz in mehr als zwei Teile aufgespaltet wird. Zusätzlich kann die Ausführung des NWA auf die verschiedenen Teilsequenzen parallelisiert werden.

3 Laufzeitmessungen und Vergleich der Algorithmen

Für die Laufzeit-Messungen werden die beiden Algorithmen hintereinander ausgeführt auf jeweils zwei mal 100 Sequenzen aus einem fasta-File und zwei mal 100 zufällig generierte Sequenzen. Am Ende wird die Grafik 1 ausgegeben, die die vier Ausführungen darstellt.
Dabei wird folgende Grafik als Ergebnis ausgegeben:

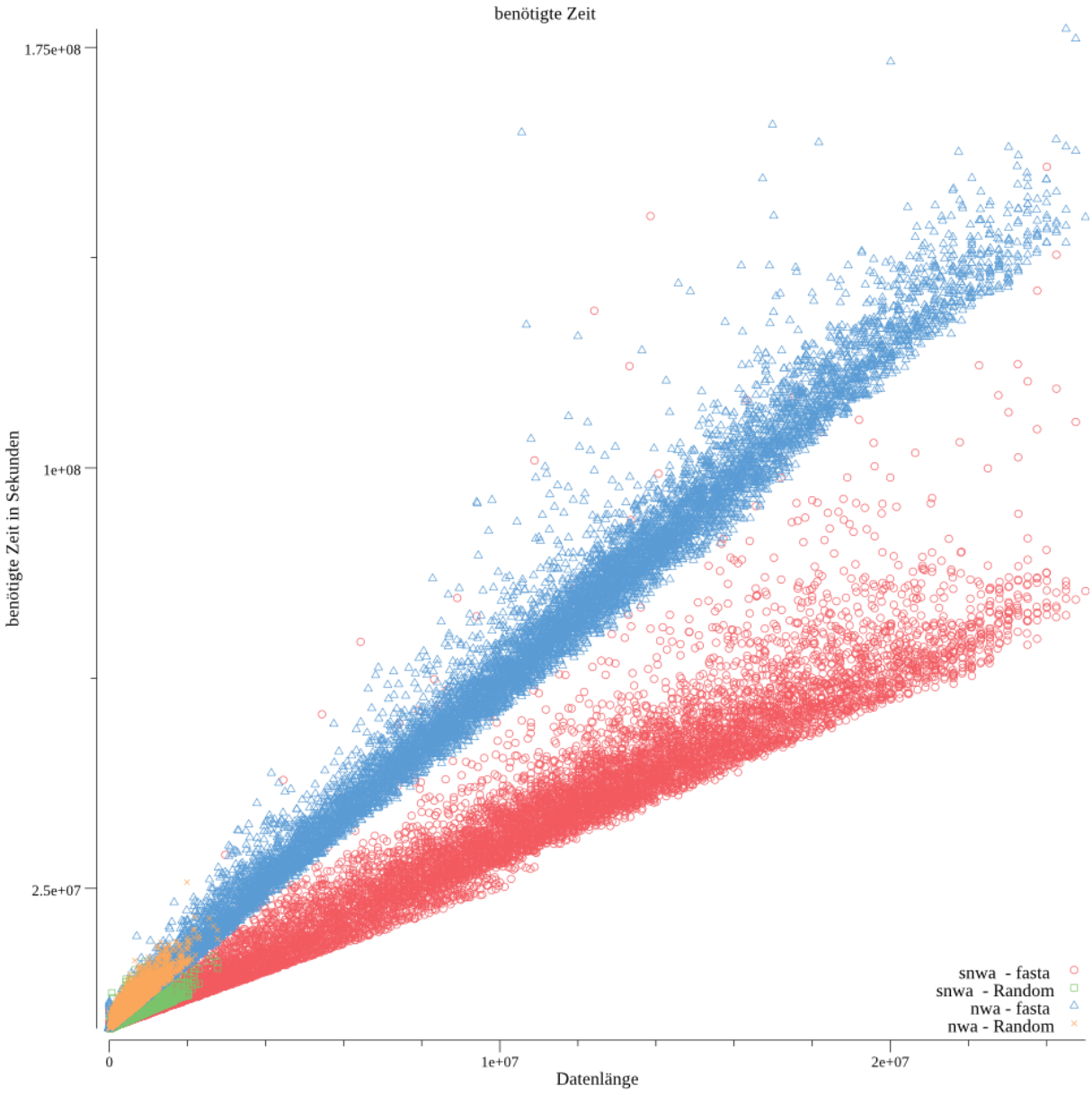

Abbildung 1: Laufzeit der Algorithmen mit Fasta-Dateien und zufällig generierten Sequenzen

Wie in der Grafik zu sehen ist, ist der Splitted NWA ungefähr doppelt so schnell wie der NWA. Auch bei längeren Vergleichen ist der Algorithmus ungefähr doppelt so schnell.

3.1 Hypothesentest

Durch einen statistischen Hypothesentest soll ermittelt werden, ob der SNWA schneller ist als der NWA.
$\tilde{\mu}_1$: Erwartungswert der Messwerte mit Needleman-Wunsch-Algorithmus
$\tilde{\mu}_2$: Erwartungswert der Messwerte mit Splitted Needleman-Wunsch-Algorithmus
Der Stichprobenumfang ist n = 100.
Angenommen wird, dass H_0: Nullhypothese: $\tilde{\mu}_d \leq 0$
H_1: Alternativ-Hypothese: $\tilde{\mu}_d > 0$
μ_1 = 1626178,78 , σ_1 = 498683,49
μ_2 = 1503446,66 , σ_2 = 896867,24
μ_d = 122732,12 , σ_d = 398183,74
Daraus ergibt sich ein t-Wert von $t = \sqrt{100}\frac{122732,12}{398183,74} = 3,0823$
t(0,95, 99) $\approx$ 1,984 < t
Die Nullhypothese wird abgelehnt und dementsprechend ist die Alternativ-Hypothese wahr, somit ist der Splitted Needleman-Wunsch-Algorithmus schneller als der Needleman-Wunsch-Algorithmus.

4 Ausblick

Der NWA löst das Problem des Sequenz-Alignment, das jedoch bei einer Laufzeit von O(mn). Die Laufzeit des Algorithmus kann durch das Aufteilen der Sequenzen verbessert werden, das wurde im Splitted-NWA realisiert, jedoch wird dadurch die Genauigkeit des Scores verschlechtert. Der SNWA kann zusätzlich durch eine Aufteilung in mehr als zwei Teilsequenzen noch in seiner Laufzeit verbessert werden, jedoch wird mit jeder Teilung auch die Ungenauigkeit vergrößert. Außerdem können die Teilsequenzen auch parallel ausgewertet werden, was die Geschwindigkeit auch verbessert.

5 Anhang

5.1 kurze Programmdokumentation

5.1.1 Nutzung

Um das Programm auszuführen werden nur das bin-file mit dem Namen ae-string-alignment-go, sowie das config-file mit dem Namen config.yml und die Dateien benötigt. Die Dateien können in einem Ordner liegen, der Pfad und Name muss nur in der Config-Datei angegeben werden.
Die Config-Datei ist folgendermaßen aufgebaut:

```
files:
  first: data/aedes_aegypti_protein.fa
  second: data/aedes_albopictus_protein.fa
  scatter: points.png
  scores: score
  times: times
algorithm: nwa
profile: true
test: true
```

In dem Abschnitt "files" werden die Dateien angegeben, dabei sind "first" und "second" die Dateien, die eingelesen werden. "scatter" ist der Dateiname für ein Laufzeit-Diagramm, die gleichen Daten werden auch in dem unter "times" angegebenen file abgespeichert. Unter "score" muss der Dateiname angegeben werden unter dem die Scores abgespeichert werden.
Unter "algorithm" wird der Algorithmus angegeben, da gibt es die Auswahl zwischen nwa und snwa für Needleman-Wunsch-Algorithmus oder Splitted Needleman-Wunsch-Algorithmus. Wenn "profile" auf true gesetzt ist, wird das Programm geprofiled, dabei wird eine Datei im tmp-Ordner angelegt, die über 'go pprof' ausgewertet werden kann. Wenn "test" auf true gesetzt ist, werden nur zwei Sequenzen miteinander verglichen und die Angabe eines Algorithmus ignoriert.

5.1.2 Dependencies und Profiling

Das Programm wurde in go 1.10 bzw. 1.11 geschrieben, dementsprechend wurde in diesem Projekt noch der $GOPATH genutzt. Die dependencies können über "go get" mit den folgenden dependencies nachgeladen werden:

```
"github.com/olebedev/config"
"github.com/pkg/profile"
"gonum.org/v1/plot"
"gonum.org/v1/plot/plotter"
"gonum.org/v1/plot/plotutil"
"gonum.org/v1/plot/vg"
```

Anschließend kann das Programm mit "go build" gebaut werden.
Wenn das profiling genutzt werden soll, können folgende zwei Befehle die Datei in entweder Text oder eine Grafik umwandeln (Paket graphviz wird benötigt).

```
go tool pprof --text ~/path/to/executable /tmp/profile054282181/cpu.pprof > file.txt
go tool pprof --png ~/path/to/executable /tmp/profile054282181/cpu.pprof > file.png
```

Es ist zu beachten, dass anstatt dem ersten Dateipfad der richtige Pfad zur ausgeführten Datei angegeben werden muss und statt dem zweiten Pfad der Pfad zur Datei, die beim profiling ausgegeben wurde.

5.2 Profiling-Ergebnisse

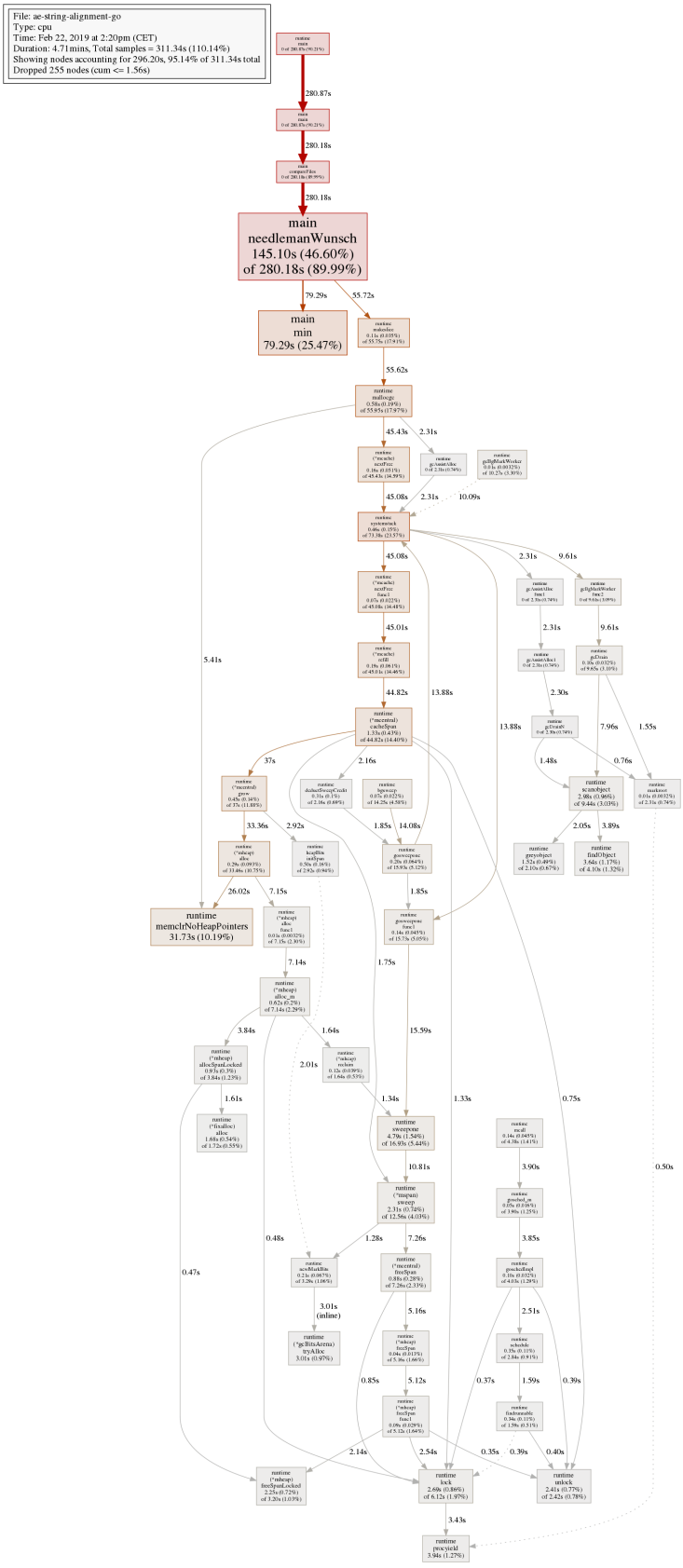

Abbildung 2: Profiling-Grafik mit der ersten Maximal-Funktion (hier ”min”)

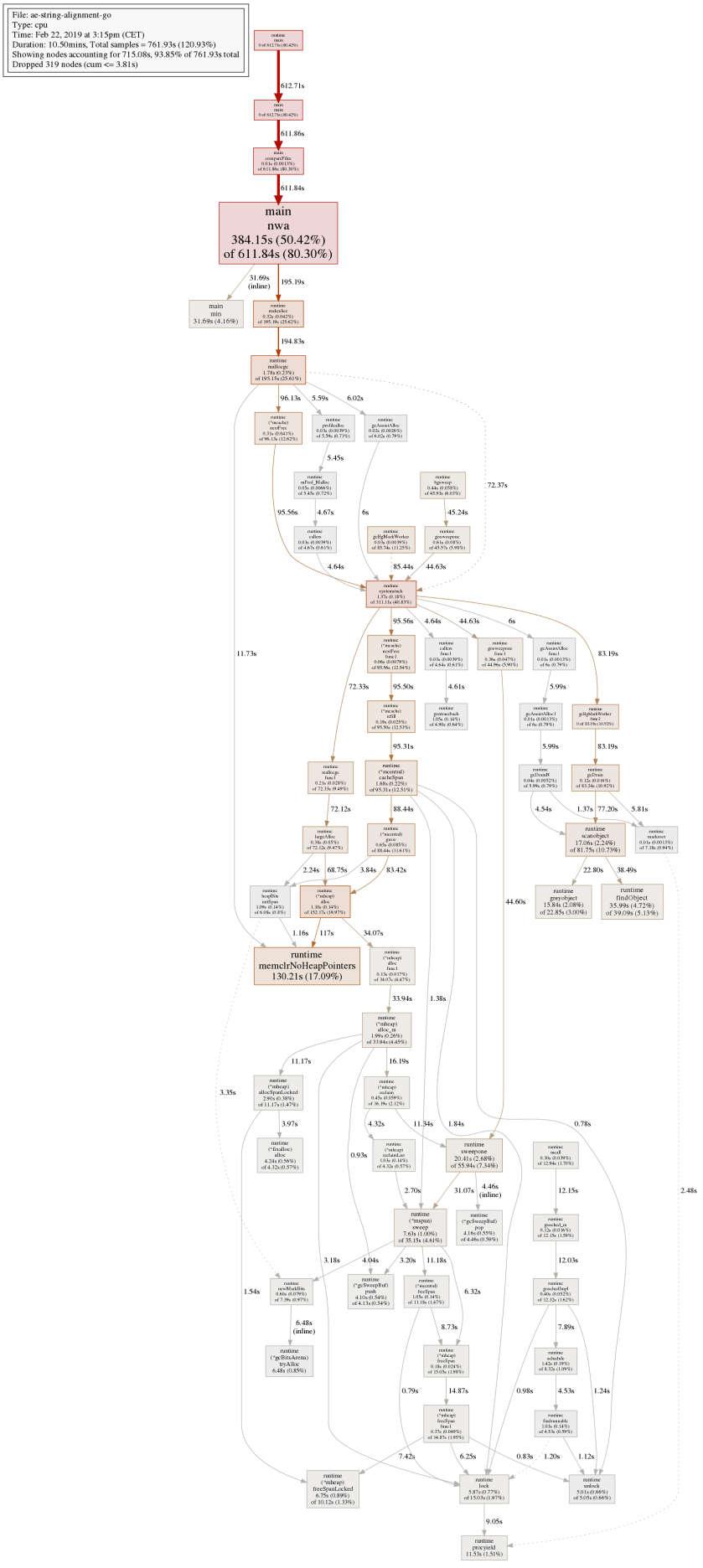

Abbildung 3: Profiling-Grafik nach der Verbesserung der Maximal-Funktion (hier "min")

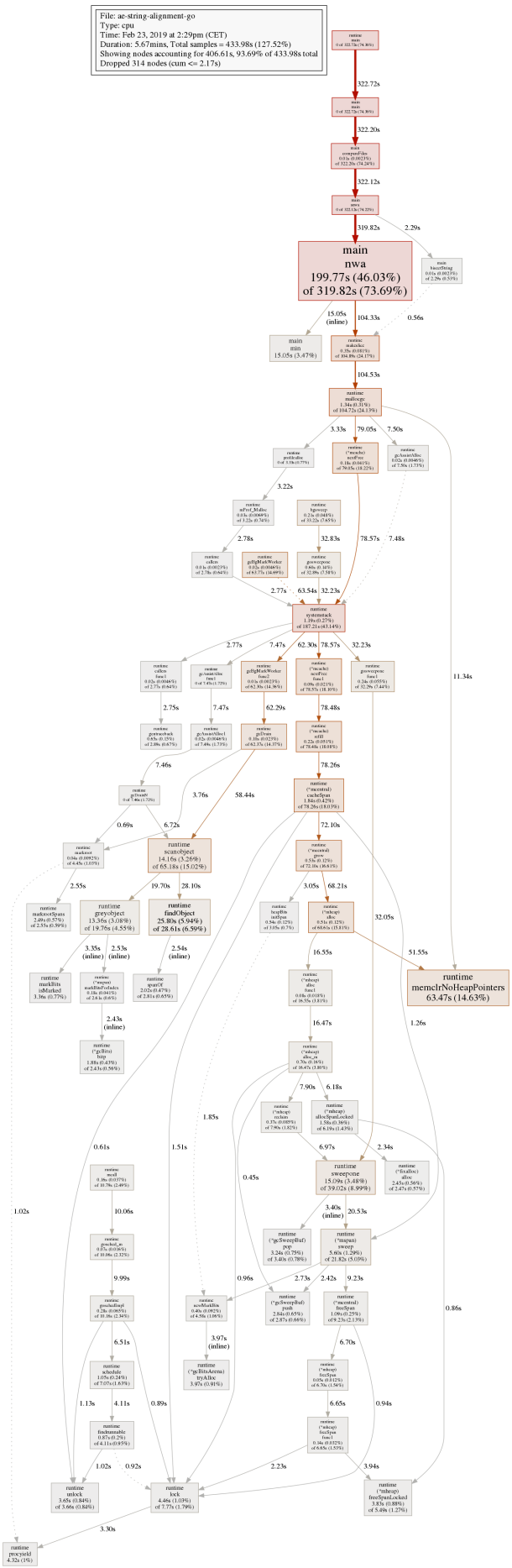

Abbildung 4: Profiling-Grafik mit dem SNWA