FuE-Schwerpunkt Medizintechnik/ medizinische Biotechnologie

Das ‚skalierbare' Bioinformatik-Framework SCABIO

The 'scalable' bioinformatics framework SCABIO

Markus Gumbel, Patrick Meppe

Sie ist oft kaum sichtbar und nie greifbar, und dennoch ist sie überall drin – auch in vielen Geräten der Medizintechnik oder im Computer der Wissenschaftler: *Software*. Biomedizinische Forschung ohne die Unterstützung von Bioinformatik ist undenkbar. Dieser Artikel stellt das Bioinformatik-Framework SCABIO (Scala algorithms for bioinformatics) vor, das an der Hochschule Mannheim entwickelt wurde.

Abstract

This paper introduces the open source bioinformatics framework Scabio which can be used for the development of individual software for the molecular biomedical domain. We present its underlying dynamical programming component, the possibility to run the algorithms concurrently or in the cloud and we explain the design of this library.

Hintergrund

Spätestens seit dem Human-Genome-Project [1], das 1990 begann und offiziell 2003 endete, war klar, dass biomedizinische Forschung ohne Computerunterstützung kaum mehr möglich ist. Dieses Projekt hatte sich zur Aufgabe gesetzt, das komplette Erbgut des Menschen zu entschlüsseln. Unser Genom besteht aus DNA, einer sehr langen Folge von vier verschiedenen Basen, die über Zucker- und Phosphatreste miteinander verknüpft sind. Sequenzierautomaten können nur DNA-Fragmente von nicht mehr als ca. 1000 Basen fehlerfrei lesen. Die gesamte DNA des Menschen, die - wie wir heute wissen - ca. 3,2 Milliarden Basen lang ist, musste daraus aufwändig wie ein Puzzle zusammengesetzt werden. Auch die Firma Celera Genomics um Craig Venter hatte sich an diesem Wettrennen beteiligt. Celeras Ansatz nutzte zum Zusammenbau der vielen DNA-Fragmente Algorithmen, die auf sehr leistungsstarken Rechner liefen, und konnte so etwa gleichzeitig zum offiziellen Projekt das entschlüsselte Genom veröffentlichen [2].

Doch obwohl unser Genom nun bekannt ist, sind noch lange nicht alle Fragen beantwortet. Im medizinischen Kontext sind insbesondere die kleinen Unterschiede zwischen Individuen im Genom interessant, die beispielsweise ausschlaggebend sein können, ob ein Medikament bei einem Patienten wirkt und bei einem anderen nicht. Mittlerweile ist die nächste Generation an Sequenzierautomaten so schnell, dass ein komplettes Genom in wenigen Tagen sequenziert werden kann. Es wird damit gerechnet, dass die Kosten dafür bald unter $ 1000 liegen werden. Dann könnte ein Genom auch für diagnostische Zwecke, z.B. bei mutierten Krebszellen, auf pathologische Unterschiede untersucht werden. Die bei solchen Sequenzierungen anfallenden Datenmengen sind gewaltig, das Genom selbst muss weiter analysiert und die Informationen müssen aggregiert werden. Dies stellt die Informatik und die eingesetzte Analysesoftware vor große Aufgaben.

Obwohl die Biotechnologie und auch die Bioinformatik feste Größe sind, sind sie relativ junge wissenschaftliche Disziplinen. Es gibt zwar zahlreiche standardisierte Bioinformatik-Methoden, die oft über Web-Portale den Wissenschaftlern und Medizinern zugänglich gemacht werden. Dennoch ergeben sich in Projekten immer spezielle Fragestellungen, die nicht über Standardsoftware gelöst werden können. Hier ist maßgeschneiderte Software nötig, um biotechnologische Prozesse optimal unterstützen zu können. Unser SCABIO-Framework hat zum Ziel, eine offene Plattform für die Entwicklung von Bioinformatik-Methoden zur Verfügung zu stellen, die so skalierbar

ist, dass auch mit sehr großen Datenmengen umgegangen werden kann.

Das SCABIO-Framework

SCABIO wurde ursprünglich ab 2010 an der Hochschule Mannheim für Lehrzwecke entwickelt und beinhaltet zahlreiche Algorithmen aus der Bioinformatik wie beispielsweise aus [3]. Dies sind bisher u.a.

- ein generischer (2D) und erweiterbarer Algorithmus zur dynamischen Programmierung
- globales, semi-lokales und lokales paarweises Alignment
- multiples Alignment
- Mustererkennung mit dem Viterbi-Algorithmus
- RNA 2D Struktur-Berechnung basierend auf dem Nussinov-Algorithmus
- Greedy Superstring Algorithmus für den Zusammenbau von Sequenzen
- und etliches mehr...

SCABIO wurde 2012 auf der Bioinformatics-Open-Source-Conference [4] offiziell als Open-Source-Projekt angekündigt und wird als Library/Tool für Scala gelistet [5]. Es unterliegt der industriefreundlichen Apache-2-Lizenz, so dass es auch in kommerziell verwertbarer Software problemlos eingesetzt werden kann. Die Homepage ist http://www.mi.hs-mannheim.de/gumbel/en/forschung/scabio und das Projekt wird auf GitHub unter https://github.com/markusgumbel/scalabioalg gehostet.

SCABIO ist komplett in Scala geschrieben. Scala [6] ist eine moderne Hochsprache, die in der Java Virtual Machine läuft und somit die zahlreich verfügbaren und ausgereiften Programmbibliotheken der Java-Laufzeitumgebung nutzt. Da Scala zu Java kompatibel ist, lassen sich Java- und Scala-Programme beliebig mischen. Somit können an dem SCABIO-Projekt nicht nur Scala-, sondern auch Java-Entwickler mitwirken. Ein wichtiges Framework ist BioJava [7], das von SCABIO genutzt wird und zahlreiche Funktion aus der Bioinformatik zur Verfügung stellt.

Dynamische Programmierung (DP)

Viele wichtige Algorithmen der Bioinformatik lösen ein Optimierungsproblem für diskrete Zustände, die meist die Basen der DNA oder RNA sind. RNA ist eine leicht abgewandelte Form von DNA, die in einer Zelle vielfältige Funktionen hat und u.a. an der Protein-Synthese beteiligt ist (Translation). So findet der Smith-Waterman-Algorithmus [8] die beste lokale Übereinstimmung (Alignment) von zwei Sequenzen, beispielsweise zweier DNA-Sequenzen. Hierbei wird geschaut, wo sich diese Sequenzen am meisten ähneln. Zur Vorhersage, wie sich RNA-Moleküle in der Zelle räumlich anordnen („falten"), kommen der Zuker- bzw. Nussinov-Algorithmus zum Einsatz [9]. In diesem Fall wird ein Energieminimum gesucht, das durch komplementäre Wasserstoffbrücken-Bindungen zwischen Basen entsteht (siehe auch Abb. 3, *RNA 2D Viewer*). Schließlich wird bei der Suche nach Mustern in DNA-Sequenzen ebenfalls ein Optimierungsverfahren (Viterbi-Algorithmus) eingesetzt [10].

Alle diese genannten Beispiele basieren auf dem Prinzip der dynamischen Programmierung (DP), das zuerst von Bellman [11] beschrieben wurde. Hierbei wird das Finden einer Lösung in einzelne Schritte unterteilt, und der nächste Schritt wird stets durch eine getroffene Entscheidung bestimmt. Bellmans Optimalitätsprinzip besagt vereinfacht: „Wenn ich bereits eine optimale Lösung für Schritt *n*-1 gefunden habe und eine Entscheidung von Schritt *n*-1 zu Schritt *n* treffe, die relativ zum vorigen Schritt (also *n*-1) am besten ist, so ist auch die Lösung im Schritt n optimal".

Dieser Ansatz lässt sich in Scala sehr gut in einem generischen Algorithmus zur dynamischen Programmierung implementieren. SCABIO stellt eine solche Komponente zur Verfügung, und die o.g. Algorithmen verwenden diese konsequent wieder. Hier unterscheidet sich SCABIO von anderen Software-Bibliotheken, bei denen die Algorithmen unabhängig voneinander realisiert wurden und keine Wiederverwendung genutzt wurde. Auch ist keine eigene Algebra

```
class Alignment(val s1: String, val s2: String, val mode: AlignmentMode,
... extends DynPro[AlignmentStep] ... {

 def n = s1.length + 1
 def m = s2.length + 1

 def decisions(idx: Idx) = {
   if (idx.i == 0 && idx.j == 0) Array(NOTHING)
   else if (idx.i == 0) Array(INSERT)
   else if (idx.j == 0) Array(DELETE)
   else Array(INSERT, DELETE, BOTH)
 }

 def prevStates(idx: Idx, d: AlignmentStep) = d match {
   case NOTHING => Array()
   case _ => Array(idx + deltaIdx(d))
 }

 def value(idx: Idx, d: AlignmentStep) = d match {
   case NOTHING => 0
   case DELETE => if (idx.j == 0 && allowLeftGapsString2) 0 else values(d)
   case INSERT => if (idx.i == 0 && allowLeftGapsString1) 0 else values(d)
   case _ => score(s1(idx.i - 1), s2(idx.j - 1))
 }
...
}
```

Abbildung 1
Auszug der Smith-Waterman-Algorithmus-Klasse in SCABIO

```
class Viterbi(val s: Array[Char], val alphabet: Array[Char],
 val states: Array[Char],  val transP: Array[Array[Double]],
 val emmP: Array[Array[Double]])  extends DynPro[Int]
 with Backpropagation[Int] with MatrixPrinter[Int] {

 def n = s.length + 1
 def m = states.length

 def decisions(idx: Idx) = {
  if (idx.i == 0) (0 to 0).toArray
  else (1 to states.length).toArray
 }

 def prevStates(idx: Idx, d: Int) =
  if (idx.i > 0) Array(Idx(idx.i - 1, d - 1)) else Array()

 def value(idx: Idx, dState: Int) = (idx.i, idx.j) match {
  case (0, jState) => log(transP(0)(jState + 1))
  case (iChar, jState) => {
   val idxS = alphabet.indexOf(s(iChar - 1))
   val e = emmP(jState + 1 - 1)(idxS) // state, char
   val t = transP(dState)(jState + 1)
   log(e * t)
  }
 }
 ...
}
```

Abbildung 2
Auszug der Viterbi-Klasse in SCABIO

wie in [12] nötig, um DP-Probleme zu formulieren. Der Vorteil mit SCABIO ist, dass nun versucht werden kann, diesen wichtigen Teil zu optimieren und zu parallelisieren, was im Abschnitt weiter unten erläutert wird. Neue Fragestellungen, die über ein solches Optimierungsverfahren gelöst werden können, lassen sich nun in SCABIO sehr schnell beantworten, da kaum neuer Source-Code geschrieben werden muss.

In den Abbildungen 1 und 2 wird auszugsweise und exemplarisch die Implementierung für den Smith-Waterman- und den Viterbi- Algorithmus gezeigt. Alle Klassen überschreiben die Funktionen `decisions()`, `prevState()` und `value()` der Oberklasse *DynPro*, und die Zustände werden in einer Matrix gehalten, die über einen Index (*Idx* mit den Eigenschaften Zeile *i* und Spalte *j*) adressiert werden. In diesen Funktionen sind die jeweiligen Eigenschaften der Algorithmen beschrieben. `decisions()` gibt an, welche Entscheidungen im Zustand *n*-1 getroffen werden können, um in Zustand *n* – dargestellt durch den Index (*i, j*) - zu kommen. `prevState()` (für previous) gibt den Vorgängerzustand von Zustand (*i, j*) in Abhängigkeit der getroffenen Entscheidung *d* (für decision) an. Schließlich zeigt `value()` die Kosten an, die entstehen, wenn mit der Entscheidung *d* in den Zustand (*i, j*) gewechselt wird. Eben diese Kosten gilt es je nach Problemstellung zu minimieren bzw. zu maximieren. Im letzten Fall spricht man dann nicht mehr von Kosten sondern von Nutzen oder allgemein Wert. Wie aus den Abbildungen 1 und 2 ersichtlich, ermöglicht die kompakte Syntax von Scala, dass diese Algorithmen mit sehr wenigen Zeilen Source-Code beschrieben sind.

Parallelisierbarkeit

Im Gegensatz zum Trend der vergangenen Jahrzehnte sind heutzutage neue Mikroprozessoren nicht mehr wesentlich schneller als ihre Vorgängermodelle. Stattdessen besitzen die Prozessoren mehrere so genannte Kerne, die paralleles und somit auch möglicherweise schnelleres Verarbeiten ermöglichen. Parallele Algorithmen sind aber nach wie vor schwierig zu entwickeln und prinzipiell gilt, dass nicht alle Algorithmen parallelisierbar sind bzw. die parallele Ausführung zu einer Geschwindigkeitssteigerung führt. Dennoch ist klar, dass die Methoden für die parallele Programmierung möglichst gut von der Programmiersprache unterstützt werden sollten.

Scala bietet neben der objekt-orientierten Programmierung auch eine vollwertige Unterstützung der funktionalen Programmierung. Dieser – eigentlich bereits sehr alte Ansatz – eignet sich sehr gut für parallele und damit skalierbare Ansätze. Man sagt, funktionale Programme sind *zustandslos*, weil bei diesem Ansatz Funktionen keine veränderlichen Variablen nutzen dürfen. Es ist somit möglich, die Funktionen auf beliebigen Prozessoren bzw. auf verteilten Rechnern parallel aufzurufen, ohne Seiteneffekte bedingt durch Zustandsänderungen zu erhalten, da Funktionen (im mathematischen Sinne) stets – in Abhängigkeit der Parameter – das gleiche Ergebnis liefern. Das Akka-Framework [13] ist das Standardframework unter Scala, das verteiltes Rechnen u.a. auch in einer Cloud-Umgebung direkt unterstützt. In aktuellen Arbeiten untersuchen wir, ob und wie gut sich die in SCABIO verwendeten Algorithmen parallelisieren lassen. Diese Parallelisierung ist Voraussetzung dafür, dass die Antwortzeit für einen Algorithmus, wie beispielsweise eine Suchanfrage für ein Alignment, möglich kurz und konstant gehalten werden kann, auch wenn die Datenmenge immer größer wird. Somit sollte es auch möglich sein, die Datenmengen der aktuellen und zukünftigen Sequenziermethoden schnell durchsuchbar und auswertbar zu machen.

Scripting/Workflows

Neben der Entwicklung neuer Algorithmen spielen Workflows in der Bioinformatik eine große Rolle. Hierbei lösen Tools, d.h. Programme, die üblicherweise in C/C++, Java, Python oder Perl geschrieben sind, ein Teilproblem und werden nacheinander aufgerufen, so dass insgesamt ein größeres Problem gelöst wird. Ein typischer Workflow ist 1) das Sequenzieren eines Genoms, 2) die Aufbereitung der Daten und 3) die – aus biologischer und medizinischer Sicht – abschließende wichtige Annotation der Daten, d.h. das Verbinden der Daten mit dem bekannten Wissen der biomedizinischen Datenbanken oder dem Wissen von Experten vor Ort. Scala und somit auch SCABIO bieten die Möglichkeit, sehr leicht eine eigene domänenspezifische Sprache zu entwickeln (domain specific language, DSL). Solche DSLs lassen sich gut in Workflows integrieren, weil so z.B. die Programmierung stark vereinfacht wird, so dass sie von einem Anwender (Mediziner, Biologen usw.) übernommen werden kann.

Architektur und Features

Abbildung 3 zeigt die Komponenten von SCABIO im Überblick sowie zwei Beispielanwendungen. Das Modul core enthält die Komponente zur dynamischen Programmierung (`*.dynpro`) und die bisher umgesetzten Algorithmen aus dem Bereich der Bioinformatik (`*.bioinf`). Im Modul demo sind Beispiele für typische Probleme der dynamischen Programmierung enthalten wie beispielsweise das Rucksack-Problem, bei dem es darum geht, Gegenstände, die verschieden schwer und verschieden wertvoll sind, in einen Rucksack zu packen, so dass dessen Trage-Kapazität zwar nicht überschritten wird, aber dessen Inhalt maximal wertvoll ist. Der *RNA 2D Viewer* zeigt das Ergebnis einer RNA-Faltung. Hier faltet sich die Kette aus Basen mit sich selbst. Die andere Beispielanwendung Edit Distance zeigt ein einfaches Alignment, das hier allerdings aus Zeichenketten der englischen Sprache und nicht aus DNA besteht. Neben den bereits oben erwähnten Algorithmen enthält SCABIO auch noch Algorithmen für das multiple Alignment und einen Superstring-Algorithmus für die Sequenzierung. Das Modul biojava enthält Klassen zur Integration mit BioJava – dies sind v.a. Funktionen zum Einlesen und Konvertieren von Sequenzen aus öffentlichen biologischen Datenbanken.

SCABIO lässt sich mit dem Konfigurationsmanagementtool Maven 2 (oder höher) [14] bauen und definiert in der aktuellen Version 0.1 die drei Unterprojekte *core*, *biojava* und *demo*. Es braucht eine Java 5

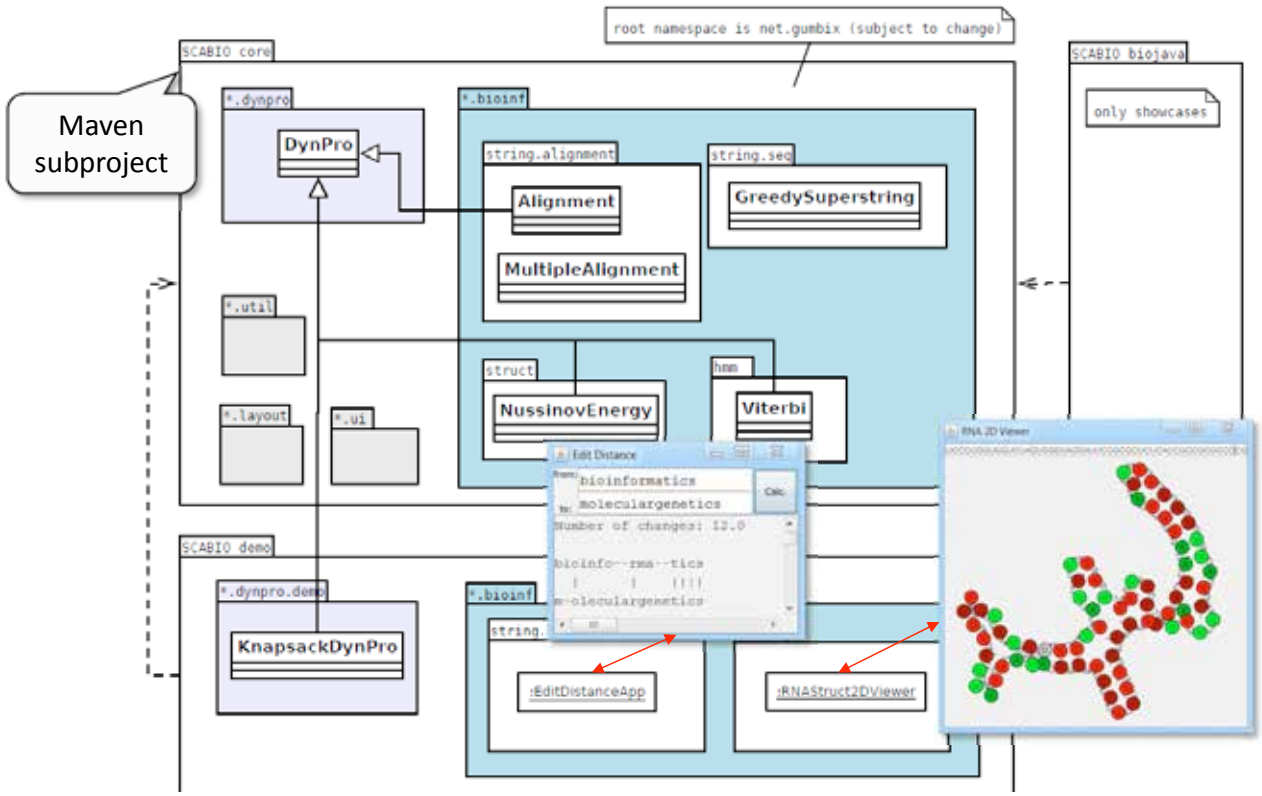

Abbildung 3
Module, Packages und die wichtigsten Klassen von SCABIO

(oder höher) Laufzeitumgebung und mindestens Scala 2.9. Weiterhin werden BioJava 3 im Modul *biojava* und die Java Universal Network/Graph Framework (JUNG) Bibliothek [15] im Modul *demo* für die Beispielanwendung *RNA 2D viewer* benötigt. Da diese Komponenten alle kostenfrei sind, kann SCABIO ohne Invest direkt genutzt werden.

Diskussion und Ausblick

SCABIO ist ein relativ junges Framework, das seine Praxistauglichkeit in großen Projekten sicherlich noch unter Beweis stellen muss. Es zeigt, wie Entwicklungsarbeiten an Hochschulen für angewandte Wissenschaften auch ohne begleitendes Projekt quasi *nebenher* bis zur Veröffentlichung gebracht werden können. Die Freigabe als Open-Source bietet die Möglichkeit, das Ergebnis sofort einem großen Benutzerkreis zur Verfügung zu stellen.

SCABIO ist neben bioscala [16] eines der wenigen Projekte im Bereich der Bioinformatik, das bereits auf die aktuelle Sprache Scala setzt. Hier wird es interessant sein zu sehen, ob und wie gut sich die Features von Scala auch in praktisch nutzbare Bioinformatik-Algorithmen umsetzen lassen. Der geringe Umfang von nur ca. 3.500 Zeilen Source-Code des SCABIO-Projekts bei der bereits vorhandenen Funktionalität zeigen unserer Meinung nach deutlich, wie gut sich die neuartigen Konzepte Scalas nutzen lassen und präzisen und wenig redundanten Source-Code ermöglichen. Dies wiederum ist für die Erweiterung und Entwicklung neuer Methoden eine wichtige Voraussetzung.

Referenzen

[1] International Human Genome Sequencing Consortium: Initial sequencing and analysis of the human genome. Nature 409 (860), 2001.

[2] J. Craig Venter et al. The Sequence of the Human Genome. Science 291(1304), 2001.

[3] H. J. Böckenhauer, D. Bongartz. Algorithmic Aspects of Bioinformatics. Springer, 2007.

[4] M. Gumbel. SCABIO – a framework for bioinformatics algorithms in Scala. 12th bioinformatics open source conference (as part of http://www.iscb.org/ismb2012). 14.07.2012, Long Beach.

[5] http://www.scala-lang.org/node/1209

[6] M. Odersky, L. Spoon, B. Venners. Programming in Scala. artima developer, 2 edition, 2010

[7] R. C. G. Holland, T. A. Down, M. Pocock, A. Prlic, D. Huen, K. James, S. Foisy, A. Dräger, A. Yates, M. Heuer, M. J. Schreiber. Biojava: an open-source framework for bioinformatics. Bioinformatics, 24(18):2096–2097, Sep 2008.

[8] T. F. Smith, M. S. Waterman: Identification of Common Molecular Subsequences. Journal of Molecular Biology 147, 1981, S. 195–197.

[9] R. Nussinov et al.: Algorithms for loop matchings. In: SIAM Journal of Applied Mathematics. 35, 1978, S. 68-82

[10] A. J. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory 13 (2): 260–269. 1967.

[11 R. E. Bellman. Dynamic Programming. Princeton University Press, Princeton, NJ. 1957, Neuauflage 2003.

[12] R. Giegerich, C. Meyer, P. Steffen. A discipline of dynamic programming over sequence data. Science of Computer Programming, 51, 2004.

[13] http://akka.io/

[14] http://maven.apache.org/

[15] http://jung.sourceforge.net/

[16] Pjotr Prins. bioscala. https://github.com/bioscala/bioscala.

Prof. Dr. Markus Gumbel ist Professor am Institut für Medizinische Informatik an der Hochschule Mannheim.

Patrick Meppe ist wissenschaftlicher Mitarbeiter am Institut für Medizinische Informatik.