HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

eawag
aquatic research

Implementation des HLC-Algorithmus für Clustering von Phytoplankton Daten

Bachelorarbeit

Abteilung Informatik
Hochschule für Technik Rapperswil

Frühjahrssemester 2015

Autor(en):	Richard Schiepek, Dario Andreoli
Betreuer:	Prof. Dr. habil. Ruedi Stoop
Projektpartner:	Eawag Dübendorf
Experte:	Prof. Dr. Thomas Ott
Gegenleser:	Prof. Dr. Josef M. Joller

Abstract

Die Eawag in Dübendorf führt regelmässige Untersuchungen von Phytoplankton aus dem Greifensee durch. Dabei wird versucht anhand von Charakteristika wie z.B. Fluoreszenzwerten Rückschlüsse auf die Spezies zu schliessen. Dazu werden Clustering Algorithmen auf grosse Datensätze angewendet. Gängige Algorithmen wie Ward oder k-Means lieferten bis anhin keine zufriedenstellende Ergebnisse. Der Hebbian Learning Clustering Algorithmus (HLC) ist in der Lage, auch kompliziertere Formen zu clustern und „Background-noize“ auszufiltern. Ziel der Bachelorarbeit war, die Verwendbarkeit des HLC-Algorithmus aufgrund von Cluster Resultaten der Phytoplankton Daten nachzuweisen.

Der HLC-Algorithmus stand in Mathematica, sowie in einer frühen C++ Version zur Verfügung. Durch das Clustern von Sample Daten der ETH Zürich konnten wir ein Gefühl für den Algorithmus und deren Parameter entwickeln. In einem zweiten Schritt verbesserten wir die Performance des C++ Codes durch massives Refactoring, wie bessere Code Struktur, bessere C++ Collections, entfernen von überflüssigen Sortierungen und auch CUDA Parallelisierung. Zu Beginn hatte der Algorithmus keine Konfigurationsmöglichkeiten. Durch das Einsetzen eines externen XML-Config-Files können nun Parameter definiert, Daten gefiltert oder nur ein gewisser Prozentsatz der Daten eingelesen werden, um auch grosse Datensätze analysieren zu können. Um die Usability für alle User zu verbessern, entwickelten wir zudem ein User Interface basierend auf der Qt-Library. Um die Plattformunabhängigkeit zu gewährleisten, wurde bereits zu Beginn der Arbeit diese hergestellt und während des ganzen Projektes Tests auf den verschiedenen Plattformen (Mac, Windows, Linux) durchgeführt.

Wir konnten durch den Einsatz des HLC-Algorithmus und dem Konfigurieren weniger Parameter dieselben, oder sogar bessere Cluster nachweisen, wie die Eawag dies mittels herkömmlichen Techniken und zusätzlicher Handarbeit gemacht hat. Ausserdem konnten wir durch optimierte Programmierung und Parallelisierung eine stark verbesserte Performance erreichen. Um die Parameter des Algorithmus zu konfigurieren, oder sonstige Feinjustierungen vorzunehmen, stellen wir ein intuitives User Interface zur Verfügung. Ausführbare Versionen des HLC-Algorithmus werden auf der Internetseite der Stoop-Group (http://stoop.ini.uzh.ch) für die Plattformen Windows, Mac und Linux zur Verfügung gestellt.

Danksagung

Wir bedanken uns bei folgenden Personen für Ihre Unterstützung während der Bachelorarbeit:

- Prof. Dr. habil. Ruedi Stoop für die kompetente Betreuung unserer Bachelorarbeit.
- Prof. Dr. Josef Joller für die unterstützenden Inputs während der Arbeit.
- Dr. Carlo Albert und Mridul Thomas für die angenehme Zusammenarbeit mit der Eawag.
- Jenny Held für die angenehme Zusammenarbeit in Bezug auf den RHLC.
- Unseren Freunden, Verwandten und Partnerinnen für Geduld und motivierende Worte.

Inhalt

1. Aufgabenstellung

Mit Hilfe automatischer Fluss-Zytometer kann man heute Charakteristiken vieler Millionen von Phytoplankton Partikeln in kurzer Zeit erfassen. Zu diesen Charakteristiken gehören u.a. die Länge, sowie Gehalt und Verteilung von Pigmenten und Chlorophyll. Es ist ein dringendes und noch weitgehend ungelöstes Problem, aus diesen Charakteristiken auf einzelne Spezies oder zumindest funktionelle Gruppen von Spezies zu schliessen. Die gängigen Clustering-Algorithmen (Ward, k-Means, usw.) haben bisher keine zufrieden stellenden Resultate geliefert. Dies mag daran liegen, dass diese Algorithmen implizite Annahmen über die Form der Cluster machen, welche in natürlichen Datensätzen kaum je erfüllt sind. Der von der Natur abgeschaute Hebbian Learning Clustering (HLC) Algorithmus von Landis et al. [1] macht keine solchen Annahmen.

Das Ziel dieser Bachelorarbeit besteht darin, den HLC Algorithmus zu implementieren und seine Verwendbarkeit anhand einiger von der Eawag zur Verfügung gestellten Datensätze zu demonstrieren.

Anforderungen an die Software

Die Software soll:

1. Plattform-unabhängig verwendet werden können
2. Ohne Einsatz kommerzieller Software kompiliert und verwendet werden können
3. Von der Kommandozeile aus aufgerufen werden können
4. Den I/O via Textfiles ermöglichen (Input: Datensatz und HLC Parameter, Output: Cluster in Form von Untermengen von Indizes)
5. Verständlich kommentiert sein, um spätere Änderungen am Quellcode zu ermöglichen
6. Genügend Leistung erbringen, um grosse Datenmengen (mindestens einige Zehntausend Partikel in 20 – 70 Dimensionen) in „vernünftiger Zeit“ zu clustern.

Eine CUDA Version des Codes ist erwünscht wird aber nicht gefordert.

Rapperswil, 24. September 2014

Prof. Dr. Ruedi Stoop
Institute of Neuroinformatics
University of Zurich / ETH Zurich

[1] Landis, Florian, Thomas Ott, and Ruedi Stoop. "Hebbian self-organizing integrate-and-fire networks for data clustering." Neural computation 22.1 (2010): 273-288.

2. Management Summary

2.1 Ausgangslage

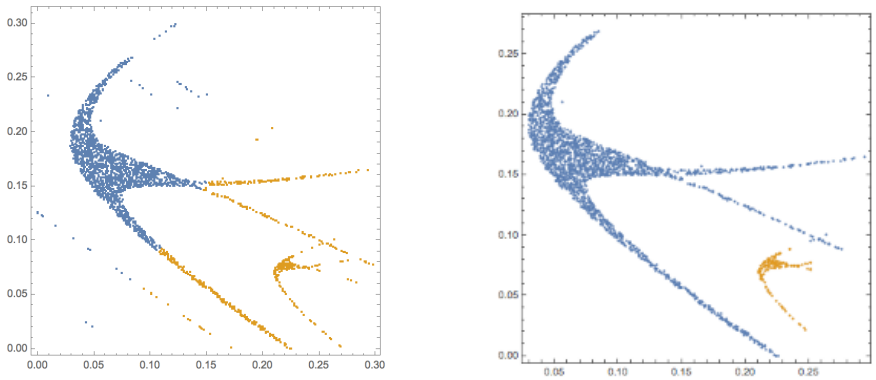
Phytoplankton machen etwa die Hälfte der Primärproduktion der Welt aus. Davon leben Tausende Arten in unseren Seen. Diese unterschiedlichen Phytoplankton unterscheiden sich in Länge, Form, etc. Wenn sich die Zahl einer einzelnen Art schlagartig vergrössert, bilden sich Algenblüten. Diese sind giftig und zurzeit nicht verhersagbar. Genau dieses Problem tritt im Greifensee fast jährlich auf.

Um dieses Problem zu lösen, überwacht das in Dübendorf stationierte Forschungszentrum Eawag laufend die Phytoplankton Ökologie und das Ökosystem und dessen Umgebung (z.B. Nährstoffe, Temperatur).

Abbildung 1: Messstation der Eawag auf dem Greifensee

Die gängigen Clustering-Algorithmen (Ward, k-Means, usw.) liefern für dieses noch weitgehend ungelöste Problem keine zufriedenstellende Resultate. Ziel dieser Arbeit soll also sein, den von der Natur abgeschaute Hebbian Learning Clustering (HLC) Algorithmus zu implementieren und seine Verwendbarkeit anhand der von der Eawag zur Verfügung gestellten Messdaten zu demonstrieren, damit in Zukunft die Bildung solcher Algenblüten vorhergesagt werden kann.

Beispiel Clustering mit k-Means und HLC:



k-Means HLC

Abbildung 2: k-Means im Vergleich mit HLC

Wie auf obenstehenden Abbildungen zu sehen ist, kann k-Means keine korrekten Cluster finden. Mit dem HLC-Clustering ist dies allerdings möglich.

2.2 Vorgehen, Technologien

Zu Beginn der Arbeit ging es darum, sich mit dem Clustering generell, sowie dem HLC Algorithmus vertraut zu machen. Der HLC Algorithmus war bereits in Mathematica und C++ implementiert. Diese beiden Implementationen waren allerdings alles andere als performant. Um mehrere 10'000 Punkte in 20-70 Dimensionen effektiv Clustern zu können, ist die Performance ein sehr wichtiger Faktor. In einem ersten Schritt ging es also darum, die Geschwindigkeit des Algorithmus zu optimieren. Mit Refactoring des C++ Codes konnte eine Performance-Steigerung von ca. 95% erreicht werden.

Um die Performance weiter zu steigern, wurde der Einsatz von CUDA, einer von NVIDIA entwickelten Programmier-Technik, mit der Programmteile durch den Grafikprozessor (GPU) abgearbeitet werden können, eingesetzt. Mit CUDA ist es möglich, Teile des Algorithmus hochgradig zu parallelisieren. Durch CUDA konnte eine weitere Steigerung von ca. 20% im Vergleich zu der non-Cuda Version erreicht werden.

Zudem wurde während dem ganzen Projekt die Bedienbarkeit des Algorithmus verbessert. So ist es nun möglich, die verschiedenen, für die Resultate des Algorithmus relevanten, Parameter über eine leicht-verständliche Konfigurationsdatei zu konfigurieren. Um dieses File zu lesen und parsen wurde auf die C++ Bibliothek TinyXML in der aktuellen Version 2 zurückgegriffen.

Um die Plattformunabhängigkeit zu gewährleisten, wurde bereits zu Beginn der Arbeit diese hergestellt und während des ganzen Projektes Tests auf den verschiedenen Plattformen durchgeführt. Besonders das Kompilieren und Einbinden der Bibliotheken auf den verschiedenen Betriebssystemen Linux, Mac und Windows war sehr herausfordernd.

Um sicherzustellen, dass während den ganzen Arbeiten am Algorithmus keine Änderungen vorgenommen werden, welche das Resultat verfälschen, wurden bereits zu Beginn Unit Tests geschrieben. Dies wurde mit der C++ Bibliothek minimal-cpp-test umgesetzt.

Da der HLC Algorithmus bisher keine gute Bedienbarkeit hatte, entschlossen wir uns, ein User Interface zu implementieren, in dem es möglich ist, diverse Parameter zu ändern und weitere Einstellungen vorzunehmen. So ist es z.B. möglich, nur einen gewissen Prozentsatz einzulesen, den Logarithmus auf die Daten anzuwenden, diverse Wertebereiche zu filtern usw. Das User Interface wurde mit der Qt-Library umgesetzt, eines der weitverbreitetsten UI-Bibliotheken im C++ Bereich für Cross-Platform Applikationen.

Zu Ende der Arbeit war noch Zeit, ein anders Neuronen Model als das „Integrate and Fire“ Neuron zu implementieren. Der HLC mit Rulkov Neuron (RHLC) wurde in Matlab von Jenny Held implementiert und von uns in C++ übersetzt. Mit Hilfe der Armadillo Library (lineare Algebra) konnten viele Matlab Funktionen in C++ umgesetzt werden.

2.3 Ergebnisse

Sample Daten der Eawag konnten mit dem HLC Algorithmus in C++ erfolgreich geclustert werden. Die Eawag clusterte die Phytoplankton Daten bisher mit herkömmlichen Methoden (Ward, k-Means) und musste danach noch „von Hand“ Daten aussortieren und manipulieren.
Die Speziesverbreitungen, die durch unsere Implementation festgestellt werden konnten, waren den bisherigen Ergebnissen der Eawag sehr ähnlich, oder diesen in verschiedenen Gesichtspunkten sogar noch überlegen.
Der HLC-Algorithmus kann nun von einem übersichtlichen User Interface gestartet werden. Dieses bietet sehr viele neue Konfigurationsmöglichkeiten, die z.B. auch ermöglichen, sehr grosse Files einzulesen oder Daten zu clustern, bei welchen ein anderes Zeichen als Trennung der Werte verwendet wird.
Zudem kann über das User Interface der gewünschte Clustering Algorithmus (I&F oder Rulkov) gewählt werden. Hier gilt es allerdings anzufügen, dass der Rulkov Algorithmus noch in der Testphase steckt und unsere Implementation noch nicht komplett ausgereift ist.

2.4 Ausblick

Für den HLC-Algorithmus gibt es noch diverse Verbesserungsmöglichkeiten, die wir leider in unserer Arbeit nicht erledigen konnten. Es wäre sicher möglich, die Performance durch weitere Parallelisierung weiter zu verbessern (z.B. den gesamten HLC-Algorithmus mit der Armadillo Library zu implementieren, inklusive Einsatz von OpenBLAS, was für parallele Matrizen-Verarbeitung verwendet wird). Ein weiterer Punkt, bei welchem noch Potential zur Optimierung vorhanden wäre, ist die Verarbeitung grosser Datenmengen. Je nach vorhandenem Memory lassen sich mit der Applikation mehr oder weniger Daten clustern. Da die Eawag jedoch Rechner mit genügend Memory bereitstellen kann, haben wir diesen Punkt nicht weiter untersucht.
Der Rulkov HLC (RHLC) ist sicher noch im Teststadium. Auch hier wäre noch Optimierungs- und Verbesserungspotential vorhanden.

3. Einleitung

3.1 Data Mining & Clustering

Beim Data Mining geht es darum, aus einer grossen Anzahl von Daten, wertvolle Informationen zu extrahieren. Eine untergeordnete Aufgabenstellung des Data Mining ist die Clusteranalyse. Beim Clustern wird versucht, Daten die sich ähnlich sind, in einzelne Gruppen zu unterteilen. Die Daten können aus 1-n Dimensionen bestehen. Es gibt schon diverse Cluster-Algorithmen. Einer der bekanntesten ist der k-Means Algorithmus, welcher sich jedoch eher für sphärische Cluster eignet.

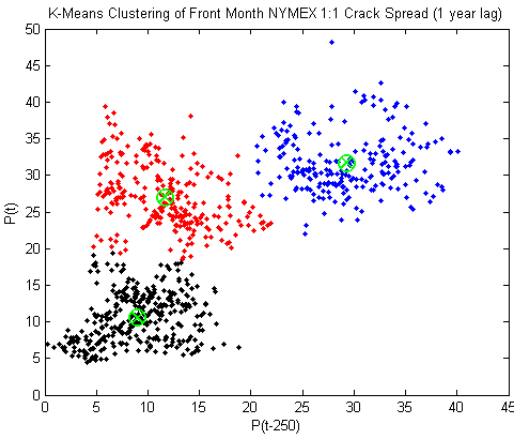


Abbildung 3: Beispiel Clustering

3.2 HLC-Algorithmus

Im Gegensatz zu k-Means oder Ward, ist der HLC-Algorithmus dazu in der Lage, auch kompliziertere Formen multidimensional zu clustern. Er bedient sich dabei der hebbschen Lernregel, die aus der Neurowissenschaft bekannt ist. Diese besagt, je häufiger ein Neuron A gleichzeitig mit Neuron B aktiv ist, umso bevorzugter werden die beiden Neuronen aufeinander reagieren ("what fires together, wires together").

Der Ablauf des HLC besteht aus diversen Abschnitten

1.	Distanzmatrix	Die Daten werden in einer Distanzmatrix abgelegt. Die Grösse der Matrix beträgt n*n
2.	Parameter Initialisierung	Alle nötigen Parameter des HLC werden initialisiert, sowie eine (n*n) Matrix der Aktionspotentiale, die nach dem Zufallsprinzip aufgefüllt wird, bereitgestellt.
3.	Spiking Clustering	Nun wird eine Nachbarschaftsmatrix mit den x nächsten Nachbarn und eine Matrix mit den Verbindungsstärken erstellt. Aus diesen werden anschliessend eine Matrix mit den Neuronen erstellt, die besonders stark zusammenhalten („strongComponents“).
4.	Integration-Loop	Der Integration-Loop bildet den Hauptteil des Algorithmus. Es wird ein neuronales Netzwerk mit dem „Integrate and Fire“ Modell simuliert. Dabei werden die Verbindungsstärken zwischen den Neuronen stets neu berechnet. Sobald alle Verbindungsstärken fast bei 1 oder fast bei 0 sind, wird der Loop beendet.
5.	Find Clusters	Alle Neuronen ohne Connection (Verbindungsstärke 0) werden nun ausgefiltert. Bei den verbleibenden Neuronen werden die Zusammenhängenden in einzelne Cluster gespeichert.

4. Anforderungsspezifikation

4.1 Zweck der Software

Durch die HLC-Software soll es möglich sein, grosse Daten in Gruppen einteilen zu können (Clustering). Der HLC-Algorithmus wurde bereits auf kleinere Datensätze getestet, jedoch noch nicht auf grössere Daten, die aus einem konkreten wissenschaftlichen Umfeld stammen. Ziel der Software ist es, den HLC-Algorithmus auf Daten der Eawag anzuwenden. Die Daten enthalten diverse Angaben über Phytoplankton Partikel. Mittels des HLC sollen nun vernünftige Cluster aus den bereitgestellten Daten gefunden werden.

4.2 Features & Anforderungen

- Die Hauptanforderung an die Software ist, vorhandene Cluster in den Daten der Eawag zu finden.
- Die Software liest ein Input-File ein und generiert einen Output, in dem die Daten in verschiedene Cluster eingeteilt sind.
- Die Output-Daten sollten mit einer Mathematik Software (Mathematica, Matlab) problemlos einlesbar sein.
- Die Konfiguration der Parameter des Algorithmus, sollte ohne Veränderung des Quellcodes möglich sein (z.B. externes XML-File).
- Die Input-Daten könnten so genannte Ausreisser enthalten. Ausreiser sind Punkte, welche sehr weit von den restlichen Punkten entfernt sind. Um diese zu filtern, sollte der analysierte Datenbereich mittels Parameter eingeschränkt werden können.
- Bei einer grossen Datenmenge (Big Data) kann es dazu kommen, dass der Algorithmus eine sehr lange Laufzeit hat (mehrere Stunden/Tage). Aus diesem Grund soll es möglich sein, die Menge der Daten künstlich einzuschränken, indem z.B. nur ein gewisser Prozentsatz der Daten verarbeitet wird.
- Bei der Eawag wird auf normalen Desktop-Rechnern und Notebooks gearbeitet. Falls die Daten zu gross sind, stehen bei der Eawag Cluster zur Verfügung, die genutzt werden können.
- Je nach Zeit und Anforderung, ist eine Implementation von CUDA (Parallelisierung mit Unterstützung der Grafikkarte) vorstellbar.

4.3 Nicht funktionale Anforderungen

Aus den nichtfunktionalen Anforderungen ISO 9126 wurden nur diese Anforderungen beurteilt, die für unser Projekt von Belangen sind.

Interoperabilität	Die Software sollte in einer Programmiersprache implementiert sein, die auf sämtlichen Plattformen (Linux, Windows, Mac) ausführbar ist. Es sollten daher keine plattformspezifischen Bibliotheken benutzt werden.
Fehlertoleranz	Der HLC-Algorithmus liegt zu Beginn der Arbeit in Mathematica vor. Die Ergebnisse, die das finale Produkt liefert, sollten sich von den Ergebnissen des in Mathematica vorhandenen Algorithmus nicht unterscheiden. Ist dies nicht der Fall, liegt womöglich eine fehlerhafte Datenanalyse vor.
Verständlichkeit	Da der Algorithmus von Wissenschaftlern benutzt wird, ist eine Benutzeroberfläche mit verständlichen Buttons oder Icons nicht zwingend.
Bedienbarkeit	Ein User-Interface muss für die Software nicht zwingend vorliegen. Es ist hinreichend, wenn die Applikation über die Konsole bedient werden kann. Der Algorithmus enthält diverse Parameter die konfigurierbar sind. Diese Parameter sollten über ein externes XML-File konfigurierbar sein. Somit muss der Quellcode vom Benutzer nicht, wie bisher, angepasst werden.
Internationalisierung	Da auch internationale Wissenschaftler mit der Software arbeiten werden, sollte der Quellcode selbst, sowie die Config Dateien, komplett in Englisch geschrieben werden.
Zeitverhalten	Das Zeitverhalten zu definieren ist schwierig, da die Gesamtdauer einer Clusteranalyse von dem Dateninput abhängt. Der vorhandene Algorithmus in Mathematica sollte jedoch mindestens um das 20-fache beschleunigt werden.
Verbrauchsverhalten	Der Quellcode des Algorithmus braucht faktisch gar keinen Speicherplatz. (ca. 500 KB). Es soll allerdings berücksichtigt werden, dass die Menge des Outputs, in etwa der Menge des Inputs entspricht. Deshalb sollte genug Speicherplatz für diese Daten bereitstehen.
Modifizierbarkeit	Der Quellcode sollte offen vorliegen. So kann eingesehen werden, wie der Ablauf von statten geht. Durch dies können auch direkt Anpassungen vorgenommen werden.
Installierbarkeit	Auf allen gängigen Systemen (Linux, Windows, Mac) muss die Software ohne grosse Vorkenntnisse installiert werden können. Es muss keine Install-Datei vorliegen. Der Algorithmus sollte mit einem geeigneten Compiler bereitgestellt werden können.

5. Umsetzung in C++

5.1 HLC in Mathematica

Zu Beginn der Arbeit betrachteten wir den HLC-Algorithmus in der Mathematica Version. In Mathematica können Daten mit mehreren Dimensionen eingelesen und analysiert werden.

Um den Algorithmus und seine Parameter kennen zu lernen, führten wir mehre Versuche mit Mathematica aus. Ziel war es, gute Cluster in den von Herrn Stoop bereitgestellten, oder aus dem Internet stammenden Daten zu finden. Mit zunehmender Anwendung des Algorithmus konnten wir immer besser sehen, welche Parameter von Bedeutung sind, um gute Cluster zu finden. Wir erhielten zudem ein Gespür, was das Ändern der einzelnen Parameter für einen Einfluss auf das Ergebnis hat.

Der grösste Knackpunkt bei dem in Mathematica vorhandenen ist die enorm schwache Performance. Eine Clusteranalyse von 3000 Datensätzen mit zwei Dimensionen kann schon einmal mehr 15 Minuten in Anspruch nehmen. Eine Ausführung in einer höheren Programmiersprache ist um ein Vielfaches schneller.

5.2 HLC in C++

Von dem HLC-Algorithmus existierte bereits eine Version in C++, die in einer anderen Bachelorarbeit geschrieben wurde. Diese Version des Algorithmus ist jedoch nicht performanter als die Version in Mathematica. Um eine Performancesteigerung zu erreichen, haben wir einige Optimierungen am Algorithmus vorgenommen, welche unter Punkt 5.3 erläutert werden.

Der Ablauf des ursprünglichen C++ Algorithmus ist folgendermassen:

1. Folgende Parameter werden über die Kommandozeile mitgegeben:
   "C:\InputFolder\data.txt" *(Input File)*
   7 *(Parameter Neighbourhood Neurons)*
   "C:\OutputFolder" *(Output Ordner)*
   clusters *(Output File)*
   .txt *(Output File Extension)*

   Beispiel:
   $ ./hlc "C:\InputFolder\data.txt" 7 "C:\OutputFolder" clusters .txt
2. Daten werden eingelesen
3. Distanzmatrix wird erstellt
4. Integration-Loop
5. Cluster extrahieren
6. Ergebnis in Output schreiben

5.3 Optimierung der C++ Programmierung

Nach ausführlicher Analyse der C++ Version des HLC-Algorithmus änderten wir einige Stellen im Programm um die Performance erheblich zu steigern.

Der Gesamte Algorithmus wurde mit Messpunkten versehen, um zu analysieren, welche Stellen besonders viel Zeit beanspruchen. Besonders in den inneren Schleifen im Integration Loop braucht der Algorithmus sehr viel Zeit.

Hier ein Ausschnitt aus einer unserer Zeitmessungen:

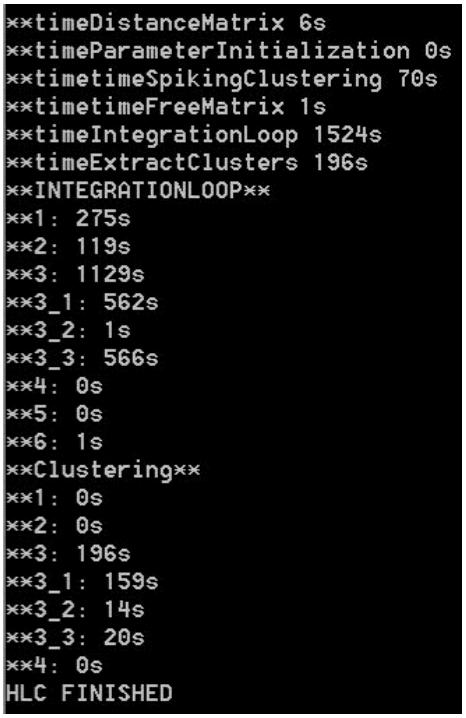


Abbildung 4: Zeitmessung HLC

Besoners Block 3_1 (Aufruf der Funktion HlcOrdering()) braucht viel Zeit:

```
1  while {valueMaxVoltage > theta) {
2    thatLoop++;
3    integrationStart = currentTime(NULL);
4    orderedVoltages = HlcOrdering(voltageOfNeuronsT, allDataPoints);
5    integrationLoop_3_1 += currentTime(NULL) – integrationStart;
6    indexMaxVoltage = orderedVoltages.MaxValueIndex;
7    valueMaxVoltage = orderedVoltages.MaxValue;
8    orderedVoltages.Free();
9  }
```

Auch im Block 3_3 wird diese Funktion HlcOrdering() aufgerufen.

5.3.1 Implementierungen für die Optimierung

Um die benötigte Zeit einer Clusteranalyse zu senken und den Code zu bereinigen, wurden folgende Anpassungen vorgenommen:

Elimination (Naked) Pointers	Nakte Pointer sollten vermieden werden. Anstatt dessen wurden std::vector oder std::array eingesetzt.
Funktion HlcOrdering()	Die Funktion HlcOrdering wird 2-mal in einer inneren Schleife ($O(n^2)$) im Integration-Loop aufgerufen. Bereits nach den ersten Zeitmessungen konnten wir feststellen, dass das vielfache Aufrufen dieser Funktion am meisten Zeit beansprucht. Diese Funktion hat alle Daten in einem Pointer Array aufsteigend sortiert und anschliessend ein Struct mit folgenden Arrays und double Werten zurück geliefert: - Originalvektor - Sortierter Vektor - Array mit Index des sortierten Vektors - Min/Max Value und Index Wir konnten feststellen, dass die Funktion im Integrations-Loop nur benötigt wird, um den grössten Wert im Array auszulesen. Dies kann viel performanter mit der STL-Implementation von std::max_element() gemacht werden.
min_element() / max_element()	An mehreren Stellen konnte anstelle des Funktionsaufrufs HlcOrdering(), die Funktion std::min_element() oder std::max_element() verwendet werden.
Funktion CalculateSpikes-TimesNeurons_SE()	Anstelle des for-Loops wird nun std::generate_n() verwendet, welche den Vektor füllt.
Extract Clusters	Statt vector.push_back() (bedeutet adden von items), wurde vector[i][j] = x benutzt. Bei .push_back() muss bei jedem Aufruf evtl. wieder neuer Speicher alloziert werden, was enorme Performance-Verluste mit sich bringt. Vor allem, wenn die Methode Millionen mal aufgerufen wird.
Vector/Matrix Erstellung	Generell konnte das Erstellen eines Vectors oder einer Matrix optimiert werden, indem beim Erstellen des Objekts im Konstruktor die Länge und wenn vorhanden, ein Default-Wert, mitgegeben wird. Bis anhin wurde jeweils eine (verschachtelte) for-Schlaufe durchlaufen, um den Default-Wert zu setzen.
Erstellung Distance Matrix	Auch beim Erstellen der Distance Matrix wurde bisher die Funktion HlcOrdering() aufgerufen, welche viel Zeit beansprucht. In dieser konkreten Verwendung konnte dies mit der Funktion std::min_element() optimiert werden. Damit jeweils das nächstkleinere Element gelesen werden kann, wird nach jedem Lesen der Wert in einem temporären Vektor auf max-Double gesetzt und anschliessend wieder das kleinste Element gesucht.
Index Ermittlung	Verwendung der Funktion std::distance() um den Index eines Elements/Iterators zu finden.
STL	Generell gilt es zu sagen, dass, wo immer möglich, STL Algorithmen anstelle manuell Programmierten Loops, etc. verwendet werden sollen. Die Implementierungen der jeweiligen STL Algorithmen sind bereits optimiert.

5.4 Optimierung durch CUDA

Um die Performance des Algorithmus weiter zu steigern, prüften wir den Einsatz von CUDA. Dies ist eine Parallelisierungstechnologie von NVIDA. Hierbei werden Berechnungen parallelisiert auf den Prozessorkernen der Grafikkarte ausgeführt (GPU). Gegenüber der Berechnung auf der CPU können auf der GPU massiv mehr Prozesse gleichzeitig ausgeführt werden. Am meisten Performance kann durch eine grosse Anzahl kleiner Berechnungen gewonnen werden. Im Gegensatz zu einem Notebook, das heutzutage 4-16 Cores hat, hat eine Graphikkarte mehrere 100 GPU-Cores.

5.4.1 Berechnung der Distanzmatrix

Der Einsatz von CUDA hat sich bei der Erstellung der Distanzmatrix besonders angeboten und bezahlt gemacht. Bei der Berechnung werden jeweils die Distanzen von Vektor zu Vektor ausgerechnet. Die Laufzeit dieser Berechnung beläuft sich demnach auf $O(n^2)$. Bei grossen Datensätzen konnte die Berechnung schon mehrere Minuten in Anspruch nehmen.

Der Prozess mit CUDA läuft folgendermassen ab.

1. Auf der Grafikkarte wird Speicher für die vorhandenen Daten sowie für die zu erstellende Distanzmatrix alloziert.
2. Die ursprünglichen Daten werden vom Memory (RAM) in den Grafikkarten Memory geladen.
3. Die Berechnung wird auf dem GPU ausgeführt und in einem Vektor gespeichert
4. Der resultierende 2-dimensionale Vektor wird vom Grafikkarten Memory wieder zurück in den RAM geladen und kann jetzt weiter vom Algorithmus verwendet werden.

Vorher (CPU):

```
1  for (int i = 0; i < data.PointCount - 1; i++)
2  {
3    double result;
4    for (int j = 1 + i; j < data.PointCount; j++) {
5      vector<double> sub = Subtraction(data.Data[i], data.Data[j],
6        data.DimensionCount);
7      Power_SE(sub, 2, data.DimensionCount);
8      double total = std::accumulate(sub.begin(), sub.end(), 0.0);
9
10     result = sqrt(total);
11     context.distanceMatrixVector[i][j] = result;
12     context.distanceMatrixVector[j][i] = result;
13     hlcTime.matrix = calculateTime(hlcTime.start);
14   }
15 }
```

Nachher (GPU):

```
1  __global__
2  void createDistanceMatrixKernel(double *values, double *result,
3      int numElements, int dimension)
4  {
5    int col = blockDim.x * blockIdx.x + threadIdx.x;
6    int row = blockDim.y * blockIdx.y + threadIdx.y;
7
8    if ((col < numElements) && (row < numElements)) {
9      double total = 0.0;
10     for (int k = 0; k < dimension; k++) {
11       double diff = values[row * numElements + k] - values[col *
12           numElements + k];
13       diff = pow(diff, 2);
14       total += diff;
15     }
16
17     double resultSqrt = sqrt(total);
18     result[row * numElements + col] = resultSqrt;
19     result[col * numElements + row] = resultSqrt;
20   }
21 }
```

Zeile 5/6: Berechnung der Reihe und Spalte für korrekten Index Zugriff
Zeile 9: Distanz Punkt A zu Punkt B
Zeile 11: Berechne Punkt A minus Punkt B
Zeile 13: Berechne Quadrat der Differenz
Zeile 14: Kumuliere Abstände für jede Dimension
Zeile 17: Ziehe Wurzel um Distanz zu erhalten
Zeile 18/19: Speichere Distanz in Resultatvektor ab

Der zweite Codeblock zeigt die Erstellung der Distanzmatrix auf der GPU. Da mit dieser Methode extrem viele parallele Berechnungen stattfinden, beschleunigt sich der Prozess enorm. Mit der CUDA Methode dauert die Berechnung der Distanzmatrix nicht mehr länger als 2 Sekunden, wobei mit sequentieller CPU Programmierung vorher mehrere Minuten gebraucht wurden.

5.4.2 Benutzung von CUDA im Integration-Loop

Da der Integration-Loop (Hauptbestandteil des HLC-Algorithmus) sehr viel Zeit beansprucht, wollten wir hier auch einige Teile mittels CUDA berechnen. Dazu haben wir eine Vektoraddition im inneren eines Loops mit CUDA programmiert. Nachfolgend die ursprüngliche Vektoraddition:

```
1  for (int i = 0; i < allDataPoints; i++) {
2    if (volageOfNeuronsT[i] != 0) {
3      voltageOfNeuronsT[i] += snConnection[i][indexMaxVoltage];
4    }
5  }
```

Ziel war es, jeweils die Addition der Vektoren parallel auszuführen. Dabei stellte sich heraus, dass die CUDA-Version dieses Codeausschnitts ca. 10mal langsamer lief als die sequentielle CPU-Version. Der Grund dafür ist, dass das Kopieren der Daten in den Memory der Grafikkarte O(n) Zeit benötigt. Die Vektoraddition braucht ebenfalls nur O(n), deshalb lohnt sich das Hin- und Herkopieren der Daten in die Memorys nicht. Bei der Distanzmatrix rechnete sich die Berechnung auf der Grafikkarte, da diese eine Laufzeit von $O(n^2)$ hatte.

Eine Möglichkeit, den Integration Loop noch performanter zu programmieren, wäre herkömmliche Threads auf der CPU zu starten.

5.4.3 Einschränkungen

Es gilt zu beachten, dass die Grafikkarten meist weniger Memory zur Verfügung hat als der Desktop Computer oder das Notebook RAM besitzt. Deshalb kann es schnell passieren, dass bei grossen Datensätzen der Memory auf der Grafikkarte nicht ausreicht. Der Platz, welcher für die Distanzmatrix im Memory gebraucht wird ist: **n * n * sizeOf(double)**.
Falls das Memory zu klein ist, kann in der Config Datei CUDA abgeschaltet werden. Gleiches kann gemacht werden, wenn die Hardware kein CUDA unterstützt.

5.5 Unit Tests

Um während der Manipulation des Codes nicht zu riskieren, dass am Ende der Arbeit der HLC ein anderes Resultat ausgibt als zu Beginn der Arbeit, verwendeten wir Unit Tests. Da wir nur eine kleine Test-Bibliothek einsetzen wollten, welche Open Source ist, haben wir uns für „minimal-cpp-test“ entschieden (https://github.com/vahidk/minimal-cpp-test).

Folgende Tests wurden umgesetzt:

- Einlesen des Test-Config-XML-Files und Prüfung, ob alle Werte korrekt in ein C++ Struct konvertiert wurden.
- Einlesen von Sample Daten. Die Sample Daten werden anschliessend mit dem HLC Algorithmus geclustert und in ein Text File ausgegeben. Dieses Text File wird mit einem bereits vorhandenen Text File abgeglichen. So kann evaluiert werden, ob die Resultate korrekt sind.
- Der gleiche Test wie beim vorherigen Punkt, nur mit aktiviertem CUDA.

6. Performance Messungen

Um die Steigerung der Performance aufzuzeigen, haben wir Zeitmessungen mit dem Algorithmus in Mathematica, der 1. C++ Version, der C++ Endversion und der C++ Endversion mit CUDA durchgeführt.
Für die Messungen wurde folgendes 3-Dimensionales Datenset verwendet:

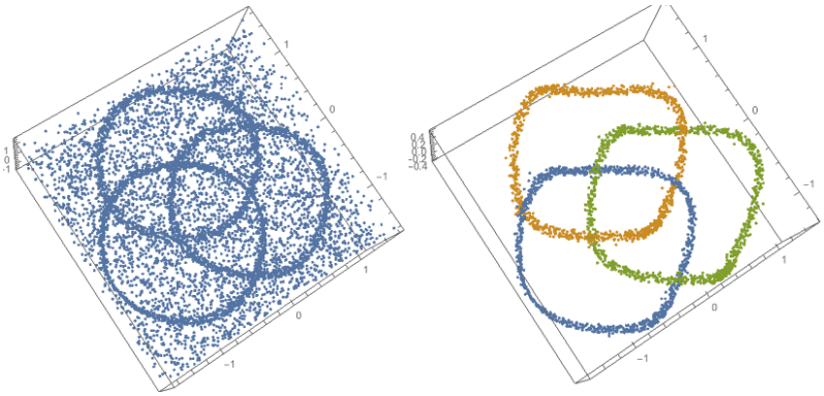


Abbildung 5: Original Dataset und Resultat

Die Messungen wurden auf folgendem Notebook durchgeführt:
Asus N56VB / 16MB RAM / Intel i7-Q6303M / 64Bit Windows 8.1
SDD HDD / Nvidia Gforce GT-740M
Alle Werte sind in Sekunden angegeben.

Anzahl Punkte	Mathematica	C++ Ursprünglich	C++ Aktuell	C++ Aktuel CUDA
400	52	19	7	7
1'000	342	129	24	21
3'000	3'457	2'435	156	132
6'000	14'651	1'9916	668	596
10'000			1'874	1'571

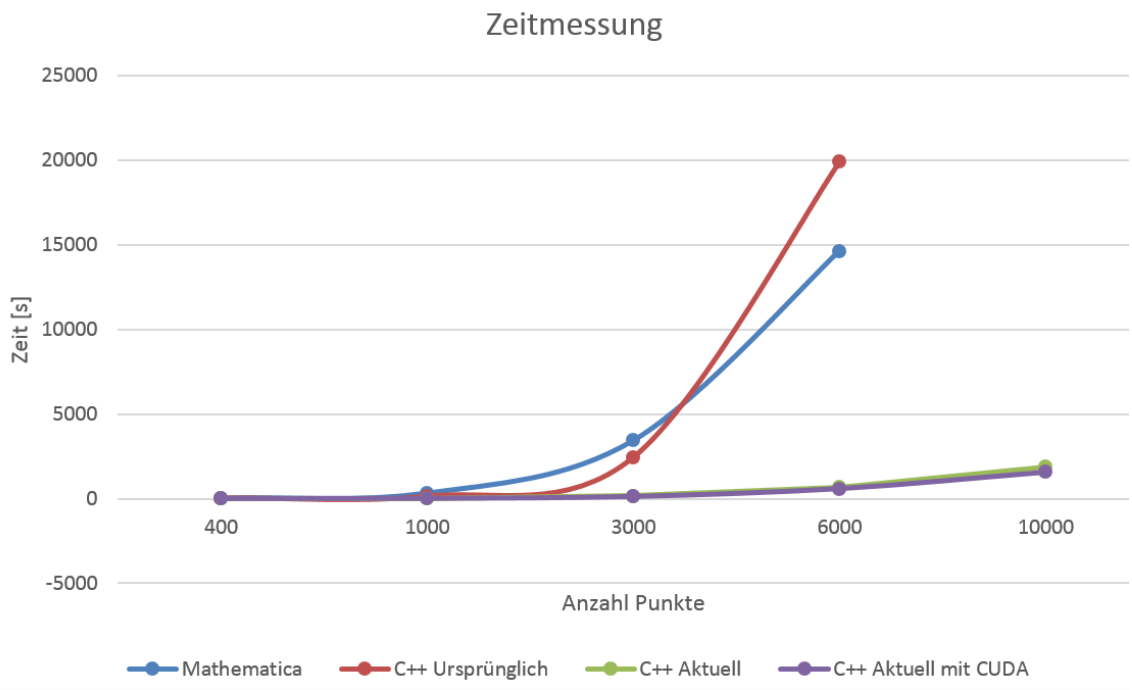


Abbildung 6: Diagramm Zeitvergleich

7. Zusätzliche Features

7.1 Externe Config

In der zu Beginn der Arbeit erhaltenen C++ Version, waren nur drei der Parameter von extern konfigurierbar. Dies waren die Anzahl „Neighbourhood Neurons“, sowie die Pfade der Input- Und Output Ordner. Bei einer kompilierten Version des Algorithmus war es also nicht möglich, die restlichen Parameter ohne Neukompilation anzupassen.
Um den HLC Algorithmus bedienbarer und flexibler zu machen, wollten wir den Benutzern viele neue Konfigurationsmöglichkeiten bieten. Diese sind nachfolgend kategorisch aufgelistet.

7.1.1 Path

InputFilePath	Absoluter Pfad der Datei mit den zu analysierenden Daten.
OutputFolder	Absoluter Pfad des Ordners, in den die Resultate geschrieben werden sollen.
OutputFileName	Name der Datei, in welche das Resultat geschrieben wird.
OutputFileExtension	Die Dateierweiterung der Resultatdatei.
WriteFilteredDataToFile	Falls Daten rausgefiltert wurden, ist der wirklich analysierte Datensatz ein anderer als der Originaldatensatz. Der wirkliche Datensatz wird hiermit speziell in ein File geschrieben.
ColumnSeparation	Wie werden Spalten im Input File separiert (z.B. “,“ für Komma oder “\t“ für Tabulator).
IgnoreFirstLine	Vielfach stehen in der ersten Zeile keine Werte, sondern Spaltenüberschriften. Mit dieser Einstellungen kann die erste Linie des Input Files übersprungen werden.
CudaActivated	Hiermit aktiviert man CUDA Parallelisierung für die Distanzmatrixberechnung. Eine Nvidia Grafikkarte mit CUDA ist dafür erforderlich.

7.1.2 Data Range

AnalyzedDataInPercent	Wie viel Prozent der Daten wirklich analysiert wird. Bei 5% z.B. wird nur jede 20. Line fürs Clustern beachtet. Besonders für Big Data sehr praktisch
AnalyzedColumns	Man will aus einem File nur bestimmte Spalten analysieren.
minPointCountToBeACluster	Es kann passieren, dass viele Cluster gefunden werden. Hier kann eingestellt werden, dass erst ab einer gewissen Anzahl Punkte eine Datenmenge als Cluster erkennt wird.
LogarithmActivated	Auf jeden Wert des eingelesenen Datenpunktes wird der Algorithmus angewendet.
TanimotoActivated	Wenn Tanimoto Aktiviert ist, wird die Tanimoto Distanz zur Berechnung der Distanzmatrix verwendet. So wird das Problem mit extrem abweichenden Datenpunkten behoben. Ist allerdings nur für I&F implementiert.
Dimension	Mit dieser Option können einzelne Dimensionen im Wertebereich eingeschränkt werden. Dies braucht man wenn man nur einen Teil der Daten clustern möchte. Dabei kann die Nr. der Dimension, sowie ein minimal- und maximal-Wert definiert werden.

7.1.3 Parameter

NeighbourhoodNeurons	I & F und Rulkov Parameter
D0Constant	I & F Parameter
TAUConstant	I & F Parameter
VoltageIR	I & F Parameter
Theta	I & F Parameter
TimeRC	I & F Parameter
AvarageDistance	I & F Parameter
Thresh	I & F Parameter und Rulkov Parameter
TimeCounter	I & F Parameter
StopCoeffizient	I & F Parameter
StopHere	I & F Parameter
Multiplikator	I & F Parameter
Timestep	I & F Parameter
MU	Rulkov Parameter
Alpha	Rulkov Parameter
Tau	Rulkov Parameter

Damit sich der Benutzer schnell zurecht findet, kann er sich eine Sample Config generieren lassen. Dies kann mit folgendem Befehl gemacht werden:

Unix: $ hlc –dump-sample-config
Windows: > hlc.exe –dump-sample-config

7.2 User Interface

Um den Algorithmus besser konfigurierbar für die Allgemeinheit zu machen, haben wir uns entschlossen, ein User Interface zu implementieren. Das UI wurde mit der Qt-Library erstellt. Dies ist die weitverbreitetste C++ Bibliothek für Oberflächen von Cross-Platform Applikationen.

Das UI unterstützt folgende Aktivitäten:

- Öffnen einer bestehenden Config Datei
- Speichern einer Config Datei
- Laden einer vordefinierten Sample Config
- Möglichkeit zu Konfiguration sämtlicher Parameter, welche auch die Config.xml bietet: Path Optionen, Data Range Optionen und Konfiguration von Parametern
- Wahl zwischen Integrate & Fire oder Rulkov Neuronen Model
- Ausgabefenster
- Start und Stop Button

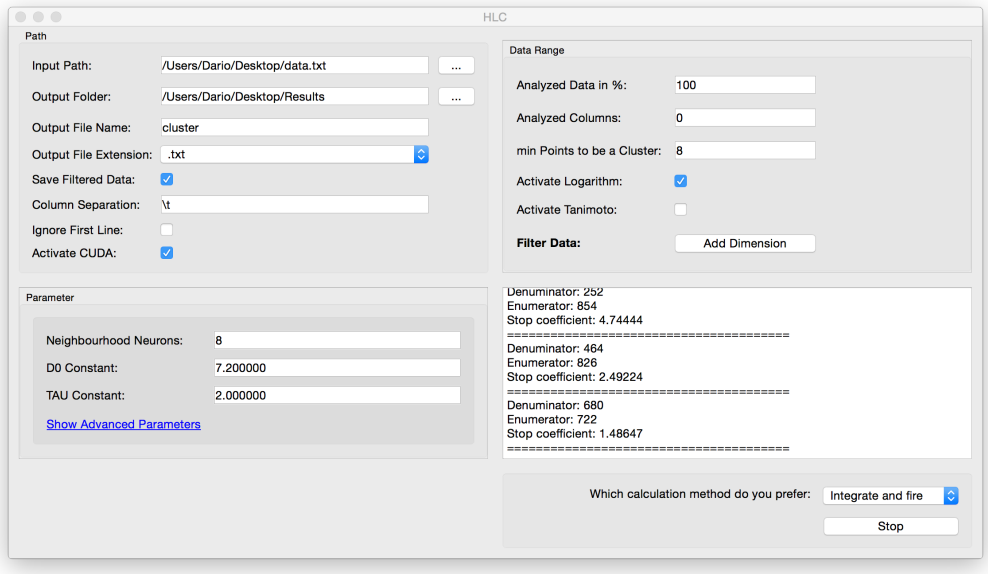


Abbildung 7: User Interface

7.3 Rulkov HLC (RHLC)

Eine Alternative zum Bilden eines neuronalen Netzes mit Integrate and Fire Neuronen ist die Rulkov Map. Der HLC Algorithmus mit Rulkov-Map wurde von Jenny Held während ihrer Masterthesis an der ETH entwickelt.

Der Hauptunterschied vom HLC zum RHLC ist im Learning-Loop zu finden. Die dynamische Gleichung ist wesentlich leichter aufzulösen und man gewinnt deshalb sehr viel Performance.
Der RHLC wurde von uns von einer Matlab Version nach C++ portiert werden.

Dazu verwendeten wir die die Armadillo Library, die performant ist und über sehr viele Funktionen der linearen Algebra verfügt. Die Library ist zwar nicht ganz so umfangreich wie Matlab, bietet aber schon mal die wichtigsten Matrix Transformationen an. Da wir nur sehr wenig Zeit zu Verfügung hatten, um den Algorithmus zu implementieren, ist der RHLC in C++ noch nicht komplett fertig (z.B. wird das Extrahieren der Cluster noch mit dem I&F Code gemacht). Liefert aber mit den ersten Testdaten schon ansprechende Ergebnisse.

Das Finden der Cluster ist noch nicht mit Armadillo umgesetzt. Hier wurde die Tiefensuche verwendet, die bereits von Anfang an beim HLC implementiert wurde. In Matlab wurde für das Finden der Cluster der Tarjan Algorithmus verwendet, den man in C++ in einer iterativen Version noch implementieren sollte.

Armadillo verwendet für die Operationen der linearen Algebra die BLAS Library. Diese läuft aber leider nicht parallel. Um noch wesentlich mehr Performance zu gewinnen, sollte man hier BLAS mit OpenBLAS ersetzen. Dies konnte aus Zeitgründen leider nicht mehr umgesetzt werden.
Im Moment ist die Version in Matlab noch wesentlich performanter als in C++.

8. Clustering Resultate

Während praktisch der ganzen Bachelorarbeit wurde versucht, in mehrdimensionalen Daten Cluster zu finden. Die Datensätze haben wir entweder von Herrn Stoop und der Eawag zugeschickt bekommen, oder vom Internet heruntergeladen.

8.1 Resultate mit Testdaten

Name: Shrimp **Quelle**: Prof.Dr. Stoop **Parameter**
NEIGHBOURHOOD_NEURONS = 9
D0_CONTSTANT = 0.5;
TAU_CONSTANT = 6
VOLTAGE_IR = 25
THETA = 16

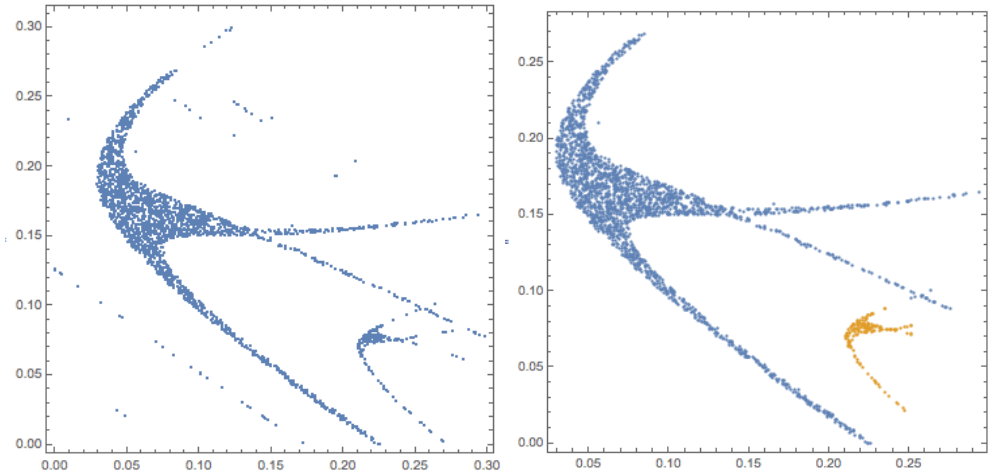


Abbildung 8: Clustering Ergebnis Shrimp

Name: Atom **Quelle**: http://cs.joensuu.fi/sipu/datasets/ **Parameter**
NEIGHBOURHOOD_NEURONS = 70
D0_CONTSTANT = 0.5;
TAU_CONSTANT = 6
VOLTAGE_IR = 25
THETA = 16

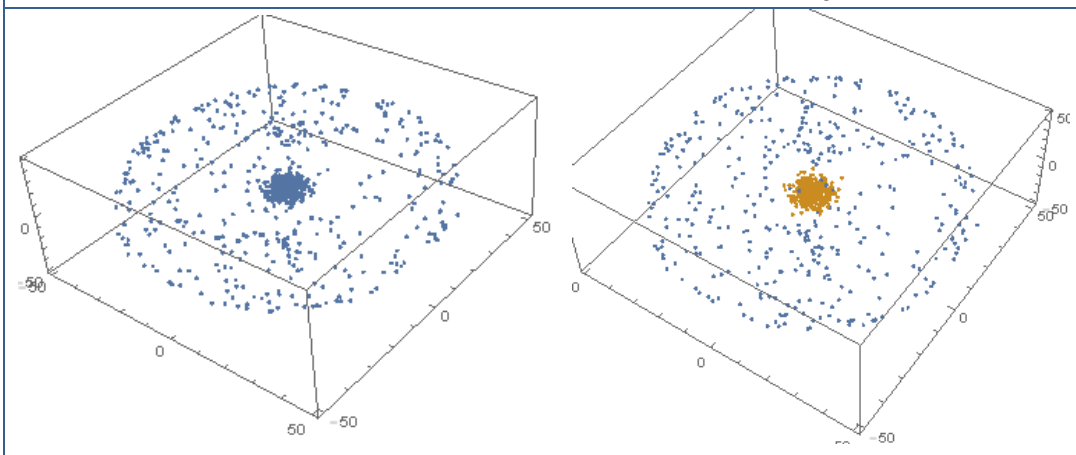


Abbildung 9: Clustering Ergebnis Atom

Name: Borrowmean wave ring	**Quelle**: Prof.Dr. Stoop	**Parameter** NEIGHBOURHOOD_NEURONS = 7 D0_CONTSTANT = 2.9; TAU_CONSTANT = 6 VOLTAGE_IR = 25 THETA = 16

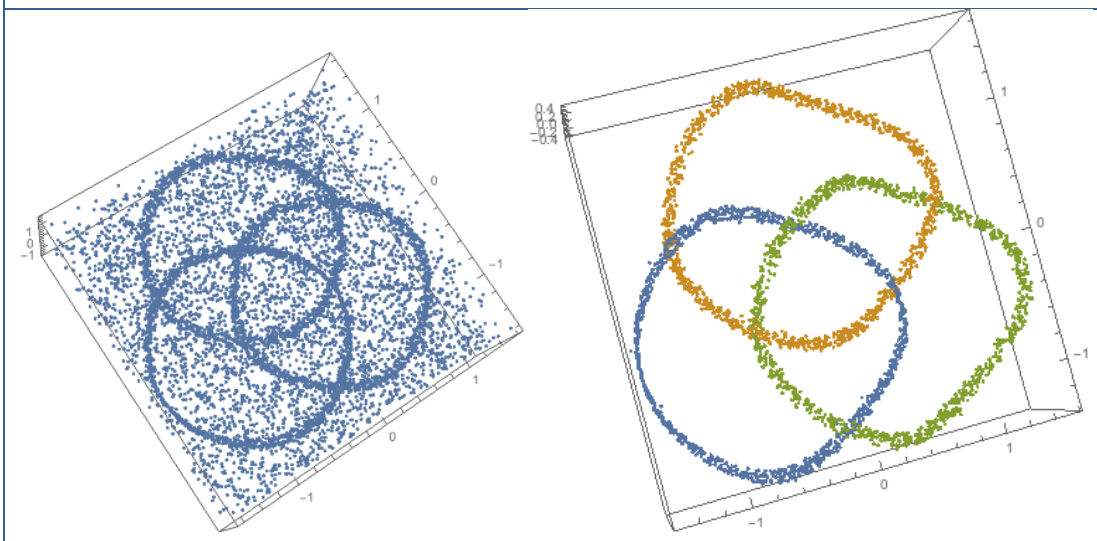


Abbildung 10: Clustering Ergebnis Borrowmean wave ring

Name: Spiral_2d	**Quelle**: http://cs.joensuu.fi/sipu/datasets/	**Parameter** NEIGHBOURHOOD_NEURONS = 36 D0_CONTSTANT = 2.1; TAU_CONSTANT = 2.9 VOLTAGE_IR = 70 THETA = 22

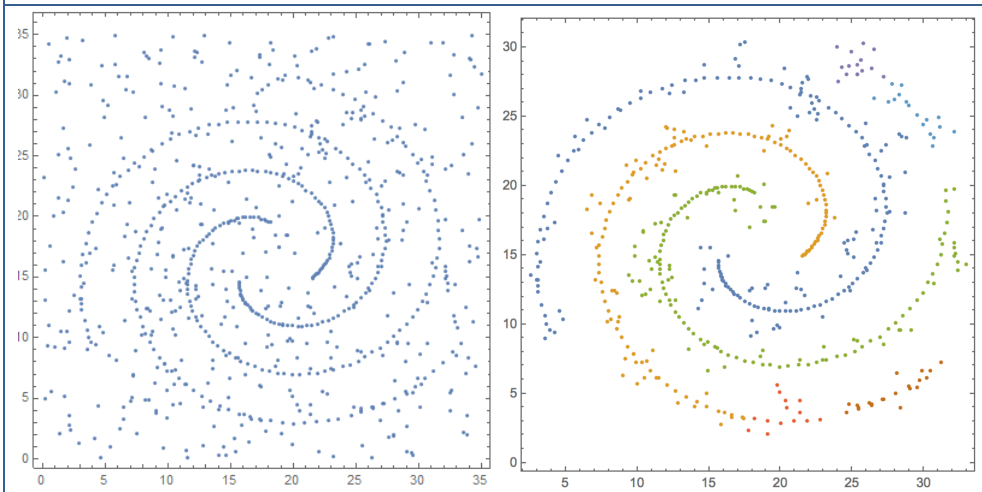


Abbildung 11: Clustering Ergebnis Spiral 2D

8.2 Resultate im Vergleich

Um den Vorteil des HLC-Algorithmus zu veranschaulichen, möchten wir gerne nachfolgende Vergleiche mit den Resultaten des k-Means Clustering veranschaulichen:

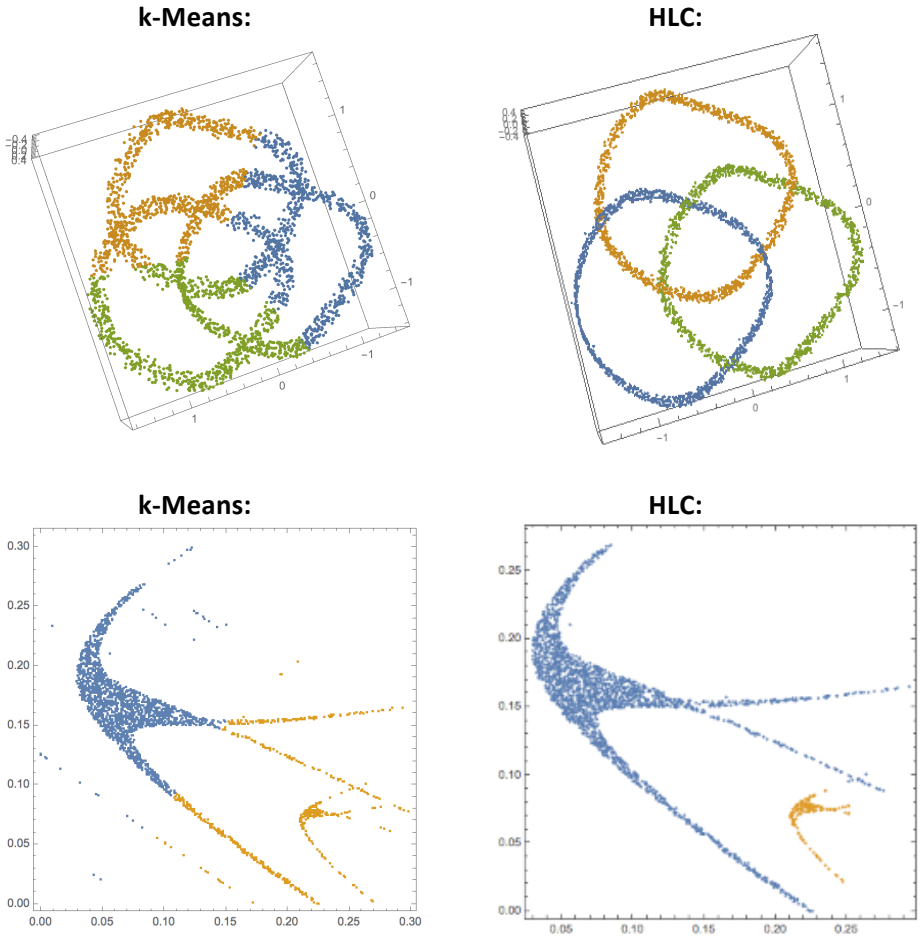


Abbildung 12: Clustering Resultate im Vergleich

Dabei ist der Nachteil des k-Means Algorithmus klar erkennbar. Da bei k-Means mit einem Schnitt die Cluster getrennt werden sollen, ist es in obenstehenden Figuren unmöglich, die korrekten Cluster zu finden. Der HLC-Algorithmus hingegen, hat die Cluster wie gewünscht gefunden.

8.3 Resultate mit Daten der Eawag

8.3.1 Lab Cultures Subset

Die Eawag schickte uns ein Phytoplankton Daten-Set mit ca. 60‘000 Datenpunkten, welche ca. 70 Dimensionen besitzen. Für das Clustern waren allerdings nur deren 5 Dimensionen relevant. Um zu erkennen, ob unsere Versuche Cluster zu finden erfolgreich sind, wurde uns eine Abbildung von bereits erfolgten Clustering geschickt. Dort ist sichtbar, welche Spalten auf welchen Achsen abgebildet wurden und welche Cluster mit dem k-Means gefunden wurden. Dabei gilt es zu erwähnen, dass von der Eawag noch manuell geclusterte Daten aussortiert und aufbereitet wurden. So hatten wir einen direkten Vergleich und einen Anhaltspunkt, welche Cluster gefunden werden sollten.

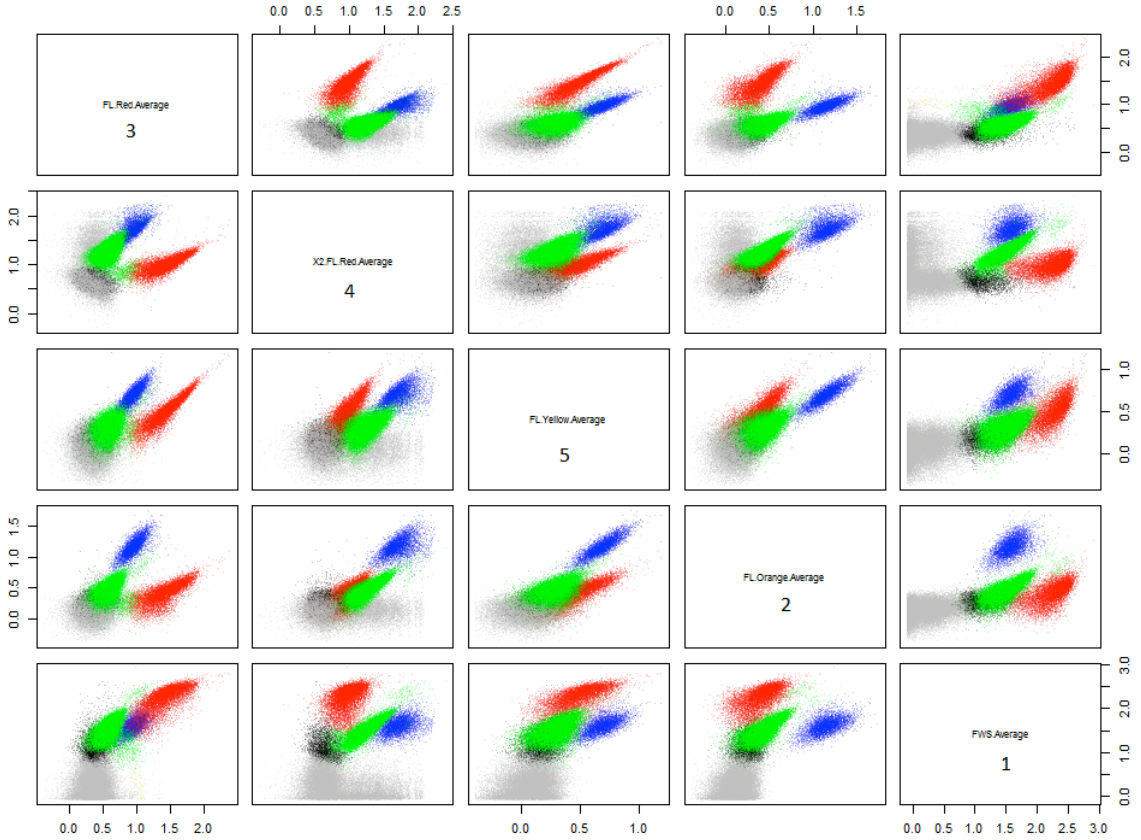


Abbildung 13: Resultate Clustering Lab Cultures Subset der Eawag

Die farbigen Cluster stellen folgende Spezies oder Gattungen dar:
Cyanobakterien: Gattung der Synechococcus und Microcystis aeruginosa.
Grüne Algen: Gattung der Scenedesmus

Der graue Cluster ist für die Untersuchungen irrelevant. Dies sind Staub Partikel, tote Zellen, Bakterien und falsche elektronische Signale, ausgehend vom Flusszytometer.
Das Clustering wurde aufgrund von Fluoreszwerten durchgeführt.

Nachfolgend sind nun die durch den von uns implementierten HLC-Algorithmus gefundenen Cluster abgebildet:

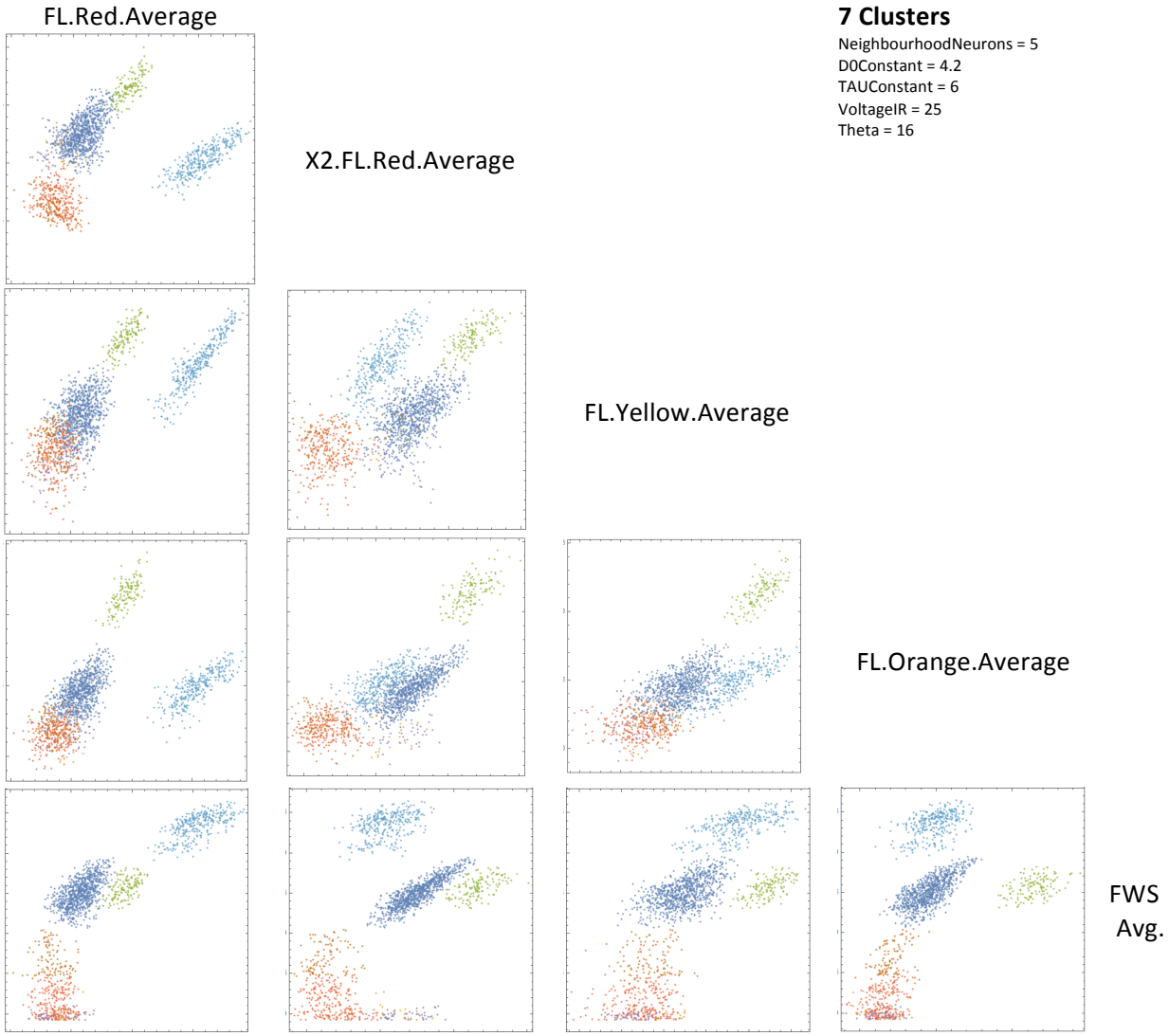


Abbildung 14: Resultate Clustering Lab Cultures Subset des HLC

Dabei ist klar zu erkennen, dass die gesuchten Cluster gefunden werden konnten.

9. Schlussfolgerung

9.1 Was wurde erreicht?

Das Hauptziel der Bachelorarbeit war das Finden von Clustern in Daten von Phytoplankton Charakteristiken. Wir konnten die Spezies aus den multidimensionalen Daten heraus clustern und genauso darstellen, wie die Eawag dies bereits in ihrer bisherigen Forschungsarbeit getan hat.
Der Algorithmus lief sehr stabil. Mit der Konfiguration von nur wenigen Parametern, konnten schnell sehr gute Resultate erzielt werden.

Der C++ Algorithmus wurde strukturiert und die Performance massiv erhöht, so das auch grössere Datensätze in vernünftiger Zeit eingelesen und geclustert werden können. Um den Algorithmus besser bedienbar zu machen, wurde zusätzlich eine externe Config-XML-Datei zur Verfügung gestellt. Darin können Parameter des Algorithmus, sowie Einstellungen über den Datenbereich, der Input/Output Ordner usw. konfiguriert werden.

Die Plattformunabhängigkeit zwischen Mac, Windows und Linux herstellen war weit schwieriger als gedacht. Wir brauchten teilweise mehre Stunden oder Tage, um die neuen Bibliotheken einzubinden und kompilieren zu können. Schlussendlich können wir aber eine Kompilierungs-, sowie Ausführungsanleitung für alle relevanten Betriebssysteme präsentieren.

Da wir beim letzten Meeting bei der Eawag (11.05.2015) schon die gefundenen Cluster präsentieren konnten, blieb uns gegen Ende der Arbeit noch genug Zeit, zusätzliche Features zu implementieren.
Wir konnten noch ein User Interface implementieren, in dem man die Konfigurations-Datei editieren und den Algorithmus bedienen kann.

Ausserdem konnten wir einen Prototypen des RHLC Algorithmus realisieren, welcher auf der Masterarbeit von Jenny Held (ETH) basiert. Beim Learning Loop des HLC Algorithmus wurde das Integrate & Fire Neuronen Model mit dem Rulkov Map Model ausgetauscht. Die Hauptaufgabe war, den Matlab Code in C++ zu portieren. Die C++ Version liefert schon passable Ergebnisse, ist aber noch nicht komplett fertiggestellt.

9.2 Was wurde nicht erreicht?

Das aufgetretene Big Data Problem konnten wir leider nicht wirklich beheben. Wir haben eine Option, dass z.B. nur jede 20. Zeile geclustert wird. Dadurch sind wir fähig, auch grössere Datenmengen zu analysieren. In den meisten Fällen spielt es keine Rolle, wenn die Daten weniger dicht sind. Dies traf in unserem Fall auch auf die Daten der Eawag zu. Wenn man jedoch wirklich jeden der z.B. 60'000 Punkten analysieren will, läuft während der Ausführung des Programms der Memory Speicher oder der Grafikkarten-Speicher voll. Ausserdem hat man bei derartig grossen Datenmengen eine lange Laufzeit, da der HLC in $O(n^2)$ läuft. Hier gäbe es sicher noch Verbesserungspotential.

Der Rulkov Algorithmus ist zwar im Learning Loop implementiert, das Extrahieren der Cluster ist jedoch immer noch mit der ursprünglichen Variante des I&F implementiert und nicht mit dem Tarjan Algorithmus, wie es in der Matlab Version genutzt wird. Leider konnten wir in der restlich verbliebenen Zeit den RHLC Algorithmus nicht mehr in einer hundertprozentig stabilen Version abliefern. Die Ergebnisse des Clusterns sind deshalb nicht immer sehr ansprechend. Hier gilt es aber zu erwähnen, dass die Implementierung des RHLC nicht Teil der Aufgabenstellung war.

9.3 Was würden wir nächstes Mal anders machen?

Zu Beginn hatten wir einen C++ Algorithmus zur Verfügung, der zwar funktionierte aber aus Sicht des Software Engineering nicht sehr sauber programmiert war. Wir versuchten den Code zu strukturieren

und optimieren, was sehr viel Zeit kostete. Beim nächstem mal würden wir von Anfang an den kompletten Algorithmus neu coden, denn nur so kann man die optimale Struktur hinbekommen.

Ausserdem hätten wir schon zu Beginn die Armadillo Library verwenden können. Für jede Matrix Transformation, oder Allgemeine Berechnung der linearen Algebra, wäre der Code mit Armadillo wesentlich kompakter und das Programm auch wesentlich performanter. Denn mit Armadillo hat man die Möglichkeit, Parallelisierungs-Libraries wie OpenBLAS einzusetzen. Zudem wäre bei einer Neuentwicklung das Verständnis des Codes noch besser vorhanden gewesen.

Was sicher auch gut gewesen wäre, wenn wir zu Beginn die Boost-Library eingesetzt hätten. Diese Library stellt sehr viele C++ Funktionen zur Verfügung, mit der man unabhängig vom Betriebssystem diverse Low-Level Systemfunktionen aufrufen kann.

9.4 Was gibt es an diesem Projekt noch zu verbessern?

- Der bisherige C++ HLC (Integrate & Fire) könnte komplett mit der Armadillo Library umgesetzt werden. Inklusive Einbindung einer Parallelisierungs Library (OpenBLAS). Dies würde die Performance und die Lesbarkeit des Codes enorm steigern.
- Der RHLC (Rulkov HLC) Algorithmus ist noch nicht ausgereift. Das Einlesen der Daten, die Erstellung der Coupling Matrix und das Erstellen der Distanz Matrix sind noch nicht parallelisiert. Der Learning Loop sollte so weit in Ordnung sein. Aber auch dort gibt es sicher Stellen, bei denen noch mehr Parallelisierung möglich wäre.
  Das Extrahieren der Cluster ist immer noch mit dem ursprünglichen Code implementiert. Hier wäre eine Suche der Strong Components mit dem Tarjan Algorithmus (wie bei der Matlab Version) sicher angebracht um die gleichen Clustering Resultate wie Jenny Held zu erhalten.
- Es gäbe womöglich noch mehr Stellen, an denen CUDA eingesetzt werden könnte. Durch das könnte man nochmals an Performance gewinnen.
- Interessant wäre eine Möglichkeit zu finden, wie man sehr grosse Daten einlesen kann (mehr als 20000 Punkte) ohne das der Memory vollläuft. Dies passiert da die n*n Matrizen enorm viel Platz im Arbeitsspeicher benötigen.

11. Literaturverzeichnis

[1]	F. Landis, Th. Ott, R. Stoop , „Hebbian Self-Organizeing Integrate-and-Fire Network for Data Clustering“, Neural Computation 22, 273-278, 2010
[2]	Ott et al., „Sequential Superparamagnetic Clustering for Unbiased Classification of High-Dimensional Chemical Data“, J. Chem. Inf. Comput. Sci., Vol. 44, No. 4, 1358-1364, 2004
[3]	C. Albert, Präsentation „Phytoplankton“, 2014
[4]	S. Glüge et al., „The Challange of Clustering Flow Cytometry Data from Phytoplankton in Lakes“, 2010
[5]	F. Gomez, R. L. Stoop, R. Stoop, „Universal dynamical proporties preclude standard clustering in a large class of biochemical data“, Bioinformatics, Vol. 30 No. 17, 2486-2493, 2014
[6]	Master Thesis „Hebbian learning clustering with Rulkov neurons“, J. Held, 30.04.2015

12. Abbildungverzeichnis

13. Glossar

Armadillo	Hoch performante C++ Bibliothek für lineare Algebra
Bibliothek (Software)	Eine Programmbibliothek bezeichnet in der Programmierung eine Sammlung von Unterprogrammen/-Routinen.
BLAS	BLAS ist eine Programmbibliothek zur Verarbeitung von Berechnungen der linearen Algebra
C++	Weitverbreitete Objekt orientierte maschinennahe Programmiersprache. Obermenge von C.
Clustering	Verfahren zur Entdeckung von Ähnlichkeitsstrukturen in Datenbeständen. Die so gefundenen Gruppen von „ähnlichen“ Objekten werden als Cluster bezeichnet, die Gruppenzuordnung als Clustering
Cmake	Ein plattformunabhängiges Programmierwerkzeug für die Entwicklung und Erstellung von Software.
CPU	Central Processing Unit. Bezeichnet den Prozessor (Recheneinheit) des Computers
CUDA	Parallelisierungs Tool für Nvidia Grafikkarten. Benutzt für einzelne Programmierteile, die Grafikkerne zur parallelen Verarbeitung
Eawag	Forschungsanstalt der ETH in Dübendorf zum Thema Wasser
Executable	Ausführbare Programmdatei
Fluoreszenz	Spontane Emission von Licht kurz nach der Anregung eines Materials
GPU	Speziell für die Berechnung von Grafiken spezialisierter und optimierter Prozessor für Computer
GUI	Grafische Benutzer Oberfläche
HLC	Hebbian Learning Clustering. Ein von der ETH entwickelter Algorithmus, der auf der hebbschen Lernregel basiert.
Integrate and Fire	Model für ein neuronales Netzwerk
k-Means	Weit verbreiteter Algorithmus zur Clusteranalyse
Kompilieren	Übersetzen des Source Codes in ein ausführbares Programm
Mathematica	Ein kommerzielles von Wolfram Research entwickeltes mathematisch-naturwissenschaftliches Programmpaket.
Matlab	Ein kommerzielles von Mathworks entwickeltes mathematisch-naturwissenschaftliches Programmpaket.
Noize	Bezeichnet das Rauschen in Daten oder Bildern
Nvidia	Eine der grössten Hersteller für Grafikprozessoren und Chipsätzen
OpenBLAS	Optimierte BLAS Bibliothek
Phytoplankton	Unterart von Plankton, enthält beispielsweise diverse Algensorten sowie Cyanobakterien. Ist für einen Grossteil der Photosynthese verantwortlich.
QT	Weit verbreitetste Bibliothek für C++ User Interfaces
Rulkov Map	Model für ein neuronales Netzwerk
TinyXML	C++ Bibliothek zum parsen von XML Dateien
Unit Test	Ist dazu da funktionalen Einzelteile von Computerprogrammen zu testen
Ward	Hierarchische Clusteranalyse
XML	Auszeichnungssprache zur Darstellung hierarchisch strukturierter Daten in Form von Textdateien