



ulm university universität
uulm

**Clusteranalyse basierend auf der
Schwarmintelligenz von
Ameisenkolonien**

Hauptpraktikum

Matthias Schneider
matthias.schneider@uni-ulm.de

Sommersemester/Wintersemester 2008

Inhaltsverzeichnis

1	Einleitung	2
2	Ameisenalgorithmen	2
3	Algorithmus	3
3.1	Pseudocode	4
3.2	Implementierung in C++	5
3.3	Das Gitter als Torus	5
3.4	Random Walk der Ameisen	6
3.5	Pheromonspur	7
3.6	Picking und Dropping	9
3.7	Ausreißer und Sackgassen	11
3.8	Eigene Verbesserungen	11
3.9	Parameter	12
3.10	Manual	12
4	Visualisierung	13
4.1	ASCII Output	13
4.2	Pixeltoaster	13
4.3	Ausblick	14
5	Evaluation und Analyse	14
5.1	Allgemeines	14
5.2	Parameterauswahl	17
5.3	Verwendete Datensätze	17
5.4	Vergleich zu anderen Verfahren	17
5.5	Golub Datensatz	18
5.6	Khan Datensatz	19
6	Interessante Beobachtungen	19
6.1	Vollständige Partitionierung der Gitterfläche mit Daten	20
6.2	Der Eistüteneffekt	21
7	Ausblick	22

1 Einleitung

Dieser Bericht beschreibt die Umsetzung eines Schwarmalgorithmus zur Clustering von Daten. Es wird die Schwarmintelligenz von Ameisen als Grundlage für die Arbeitsweise des Algorithmus herangezogen. Dabei wird die praktische Umsetzung und der theoretische Hintergrund erläutert, eigene Verbesserungen und Modifikationen besprochen sowie auf die entsprechenden Forschungsergebnisse von diversen Autoren in diesem Gebiet eingegangen. Die umgesetzten Verfahren werden anschließend evaluiert, bewertet und verglichen. Durch eine Visualisierung der Ameisen- und Datenbewegung wird die Arbeitsweise des Algorithmus grafisch verdeutlicht. Eine Besprechung der beobachteten Phänomene schließt den Bericht ab.

Mit dem Begriff der Schwärme sind wir durch Beispiele in der Natur vertraut, beispielsweise bei Fisch- und Vögelschwärme oder Ameisenkolonien. Jeder Schwarm bildet sich aus vielen einzelnen Teilnehmern, die im Ganzen ein komplexes und emergentes Verhalten erzeugen. Viele Bereiche der Informatik haben sich diese Verhaltensweisen zu nutze gemacht, um damit komplexe Probleme zu lösen [1][2]. Schwärme werden als eine Menge von kleinen, simplen Einheiten betrachtet, die zusammenarbeiten, um ein festgelegtes Ziel zu erreichen. Dabei wirken und kommunizieren die einzelnen Einheiten auf- und miteinander nur in einer lokalen Nachbarschaft und ziehen in ihre Entscheidungen auch nur diese direkte Umgebung (z.B. Landschaft und andere Tiere) mit ein. Im Ganzen ergibt sich daraus ein intelligentes Verhalten.

Das Paper von *Ramos*[3] wurde als Grundlage für diesen Algorithmus verwendet. Das dort vorgestellte Verfahren dient vielen weiteren clusternden Ameisenalgorithmen als Grundlage. Der Algorithmus wurde jedoch an vielen Stellen modifiziert, hauptsächlich bei der Bestimmung der Parameter. Diese Änderungen werden später an den entsprechenden Stellen erläutert. Die meisten Änderungen entstammen Papieren, die Ramos zitieren oder von Ramos zitiert werden und seine Ideen weiterführen bzw. beeinflusst haben.

2 Ameisenalgorithmen

Das soziale Verhalten von Ameisen dient als Vorlage für die Idee hinter Ameisenalgorithmen. In der Natur wandern Ameisen auf scheinbar zufällige Weise durch ihre Umgebung und treffen hierbei beispielsweise auf Nahrungsmittel, Baumaterial oder andere Dinge, die sie aufnehmen und zu ihrem Bau zurück tragen bzw. an geeigneter Stelle aufhäufen.

Dieses Verhalten macht man sich bei den zwei grundlegenden Varianten von Ameisenalgorithmen zunutze.

Ant Colony Optimization (ACO) basiert auf der Beobachtung, dass Ameisen auf ihrem Pfad eine Pheromonspur ablegen, die nach und nach wieder verdunstet. Wenn diese Spur von anderen Ameisen wahrgenommen wird, werden sie mit hoher Wahrscheinlichkeit dieser Spur folgen und sich nicht weiter zufällig bewegen. Dadurch werden die Pheromone verstärkt und mehr und mehr Ameisen werden sich auf diesem Pfad bewegen. Falls eine suchende Ameise einen sehr lange Spur auslegt, wird sie lange brauchen um diese Spur wieder zurück zu verfolgen. Dies führt dazu, dass sich die Spur sehr schnell wieder verflüchtigt. Eine kurze Spur wird hingegen mit hoher Wahrscheinlichkeit von vielen Amei-

sen besucht, so dass solch eine Spur länger anhält. ACO Algorithmen werden bei diskreten Optimierungsaufgaben, beispielsweise dem *Travelling Salesman Problem* angewandt.

Ant Colony Algorithms for Data Mining basieren auf den Verhaltensweisen von konkreten Ameisenspezies. Die Art *Leptothorax unifasciatus* ordnet ihre Larven der Größe nach an. Dabei wird eine Larve dann abgelegt, wenn die Larven in der direkten Umgebung ähnlich in Größe und Form zu der Larve ist, die die Ameise momentan trägt.

Mit diesem Verhalten lassen sich beliebige Daten clustern. Die grundlegende Funktionsweise ist dabei immer gleich: Die Ameisen nehmen Datenobjekte auf, wenn sie sich in einer heterogenen Umgebung befinden, d.h. wenn die anderen Objekte in der direkten Nachbarschaft relativ unähnlich sind. Abgelegt werden die Objekte wieder, falls die Ameise in eine Region läuft, in der eine homogene Umgebung herrscht, d.h. wenn die Nachbarobjekte alle sehr ähnlich zum geladenen Objekt der Ameise sind.

3 Algorithmus

In der Initialisierungsphase des Algorithmus werden die mehrdimensionalen Daten auf einer zweidimensionalen, diskreten Fläche verteilt. Dies geschieht völlig zufällig. Es werden dabei jedoch keine Daten gestapelt, d.h. in einer Zelle der Fläche liegt maximal ein Datenobjekt. Gleich verfährt man mit den Ameisen, die wiederum auf der Fläche zufällig verteilt werden.

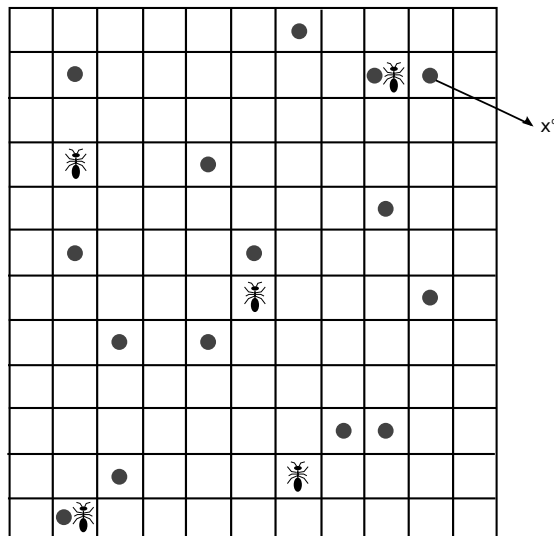


Abbildung 1: Grafische Darstellung der zweidimensionalen Gitterfläche mit verteilten Datenpunkten und Ameisen. Jeder Datenpunkt x hat d Dimensionen/Features.

Es sind jegliche mehrdimensionalen Daten möglich. Der Algorithmus erwartet hier keine besonderen Größenordnungen in Anzahl der Objekte oder Dimensionen. Das Inputformat der Daten für diese Implementierung wird im Verlauf

noch erläutert.

Das Abbruchkriterium ist das Erreichen der maximalen Anzahl an vorher festgelegten Iterationen, d.h. in jeder Iteration bewegt sich jede Ameise einen Schritt und führt eine Entscheidung durch. Eine qualitative Bewertung der Clusterung, beispielsweise durch das Varianzkriterium, geschieht nicht.

Die Ameisen bewegen sich immer genau einen Schritt weit. In welche der acht umliegenden Zellen übergegangen wird, kann auf zwei verschiedene Arten entschieden werden. Entweder die Ameise würfelt die relative Bewegung völlig zufällig aus oder trifft ihre Entscheidung auf Grundlage der Pheromonverteilung in ihrer lokalen Nachbarschaft, siehe dazu auch Abbildung 3. Beide Varianten werden später besprochen.

Trifft eine unbeladene Ameise nun auf ein Objekt, sucht sie ihre lokalen Nachbarschaft nach anderen Objekten ab. Findet sie hierbei viele, oder ausschließlich unähnliche Objekte, nimmt sie den Datenpunkt auf, da er in einer heterogenen Umgebung liegt. Ist er jedoch hinreichend ähnlich zur lokalen Nachbarschaft, bleibt der Datenpunkt liegen.

Gleich verhält es sich, wenn eine beladene Ameise nach einem Schritt die Entscheidung treffen muss, einen Datenpunkt an diese Stelle abzulegen oder ihn weiterzutragen. Wenn die Umgebung hinreichend homogen ist, wird das Objekt abgelegt. Falls nicht, wird er noch mindestens einen Schritt weiter getragen und dann eine neue Bewertung durchgeführt.

Auf diese Weise bildet sich nach einer gewissen Zeit eine emergente Struktur, d.h. es können Anhäufungen von ähnlichen Datenpunkten beobachtet werden. Wenn sich keine Objekte mehr über eine gewisse Zeitspanne bewegen, also ein stabiler Zustand erreicht ist, ist die Clusterung abgeschlossen. Die Ameisen finden dann keine Objekte mehr die in einer falschen Umgebung liegen.

3.1 Pseudocode

Der Algorithmus in kompakter und sehr vereinfachter Weise als Pseudocode: Es wurden viele Ausnahmen und Erweiterungen aus Gründen der Übersichtlichkeit weggelassen. Eine genauere Beschreibung des Algorithmus erschließt sich aus den folgenden Kapiteln und anhand der Kommentare im beigelegten Quellcode.

```
// init ants and objects
for (iterate over all objects) {
    place objects on grid randomly
}

for (iterate over all ants) {
    place ants on grid randomly
}

// main loop
while (step < RUNTIME) {
    for (all ants) {
        calculate move according to pheromone map
        move ant to new location
    }
}
```

```

for (all ants) {
  if (ant encounters object) {
    if (ant is unloaded) {
      calculate swarm similarity of local region
      compute picking up probability pp
      and random probability pr
      compare probabilities
      pick up object if pp > pr,
      change state to loaded
    }
  } else {
    if (ant is loaded) {
      drop object if moved longer than L with object
      otherwise:
      calculate swarm similarity of local region
      compute dropping probability pd
      and random probability pr
      compare probabilities
      drop object object if pd > pr,
      change state to unloaded
    }
  }
}
}

```

3.2 Implementierung in C++

Für eine grobe grafische Übersicht siehe Abbildung 2. Die Ameisen bewegen sich auf einem zweidimensionalen Gitter, das als Array realisiert wurde. In jeder Zelle des Arrays wird ein sogenanntes Pocket-Struct gespeichert, welches wiederum mit je einem Pointer auf eine Ameise und ein Datenobjekt zeigen kann. Diese structs sind die eigentlichen Ameisen- und Datenobjekte. Wenn eine Ameise ein Datenobjekt über das 2D-Grid trägt, zeigt ein weiterer Pointer im Ameisenstruct auf eben dieses Datenobjekt, da auf dieses Objekt während es getragen wird kein Pocket-Struct Pointer zeigt. Weiterhin wird in den Pocket-Structs noch die aktuelle Pheromondichte der jeweiligen Zelle gespeichert.

Wenn sich eine Ameise bzw. ein Datenobjekt über das Grid bewegt, werden bei jedem Schritt die Pointer im Pocket-Struct umgehängt. So ist es über das Grid jederzeit möglich die aktuelle Position aller Daten und Ameisen abzurufen.

Am Ende der Laufzeit des Algorithmus werden alle Datenobjekte die zu diesem Zeitpunkt noch von Ameisen getragen werden automatisch gedroppt. So ist sichergestellt das alle Datenobjekte im Ergebnis vorhanden sind.

3.3 Das Gitter als Torus

Um Randeffekte an Ecken oder entlang der vier Kanten zu vermeiden, wurde das zweidimensionale Gitter als Torus realisiert. Wenn eine Ameise beispielsweise am rechten Rand das Gitter verlässt, betritt sie dieses direkt wieder am linken Rand. Das gleiche gilt entsprechend für alle anderen Kanten.

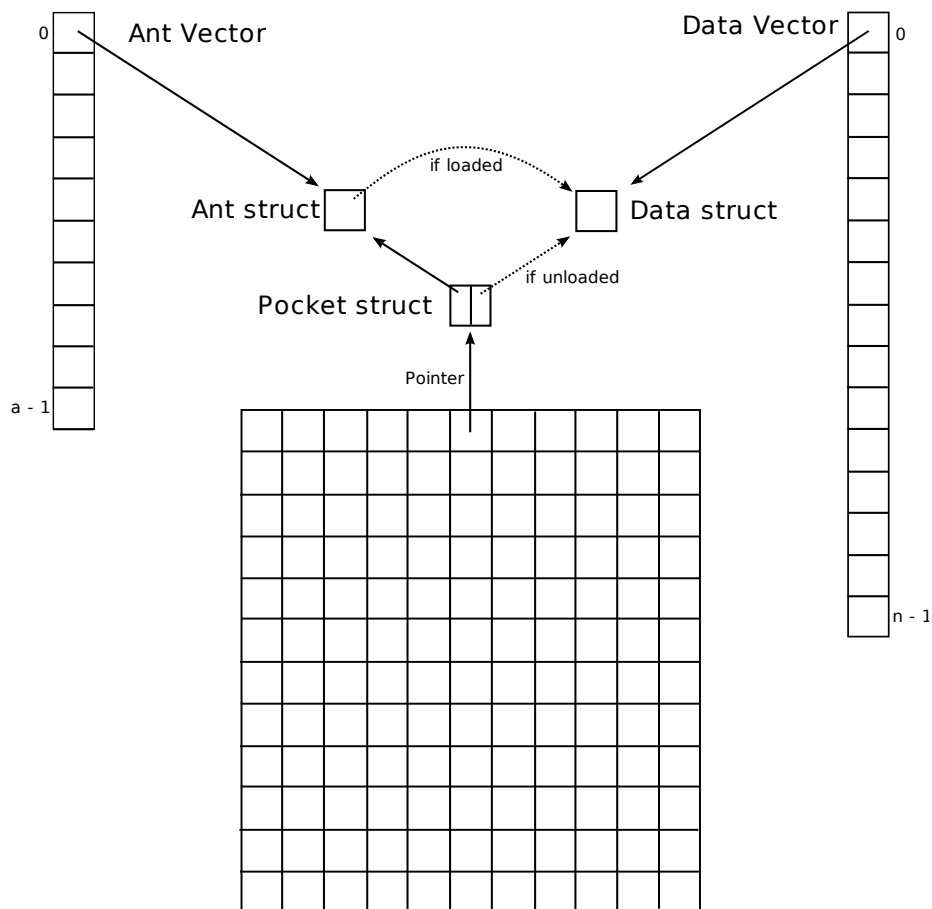


Abbildung 2: Die Datenstruktur des Algorithmus.

Auch die lokalen Nachbarschaften, in der die Ameisen nach Objekten suchen, erstrecken sich hierbei über die Kanten des Gitters.

3.4 Random Walk der Ameisen

Bei der Implementierung wurde festgelegt, dass die Ameisen zufällig die relativen Koordinaten des nächsten Schrittes wählen. Eine Ameise bewegt sich dabei zufällig in eines der acht umliegenden Felder. Jedoch sind nicht alle beliebigen Bewegungen möglich. Es muss verhindert werden, dass sich zwei Ameisen zum gleichen Zeitpunkt in einem Feld aufhalten und es dürfen keine Datenpunkte "gestapelt" werden. Auch nicht erlaubt ist, dass eine Ameise sich nicht bewegt, also eine relative Bewegung von 0,0 gewürfelt wird. Dies würde zu einer unnötigen Verlangsamung des Verfahrens führen, da die neue Zelle gleich der alten Zelle ist und diese im vorherigen Schritt schon von der Ameise untersucht wurde.

3.5 Pheromonspur

Ameisen in der Natur bewegen sich nicht, wie es vielleicht auf den ersten Blick den Anschein hat, völlig zufällig. Jede Ameise sondert Pheromone ab, die andere Ameisen der Kolonie wiederum wahrnehmen und je nach Intensität folgen. Jedoch verflüchtigen sich diese Pheromone mit der Zeit wieder. Es ergibt sich somit eine Karte der Ameisenbewegungen, die man als Geschichte des Ameisen-schwarms ansehen kann. Stellen, an denen es für die Ameise interessante Dinge wie Futter oder Baumaterial gibt, werden von vielen Ameisen frequentiert und es bildet sich somit eine starke Pheromonspur die weitere Ameisen anlockt. In uninteressanten Gebieten bildet sich eine solche Spur nicht. Ameisen kommunizieren daher nicht direkt miteinander, sondern es bildet sich eine indirekte Kommunikation über das Pheromonfeld.

Chialvo und Millonas [4] übertragen dieses Verfahren auf künstliche Ameisenschwärme und entwickeln ein stochastisches Modell, das das Verhalten der Ameisen abbildet und dazugehörige Konstanten benennt. Jede Ameise sondert in jedem Schritt in ihrer Zelle eine konstante Menge an Pheromonen $\eta = 0.07$ ab. Nachdem alle Ameisen sich bewegt und ihre Entscheidungen getroffen haben, verflüchtigen sich die Pheromone wieder um die Konstante $k = 0.015$. Negative Pheromonwerte treten nicht auf. Zu Beginn des Verfahrens sind in keiner Zelle Pheromone von finden (Pheromondichte $\sigma = 0$). Die Pheromondichte gibt die Anzahl der Pheromone in einer Zelle an.

Der Zustand einer Ameise zu einem Zeitpunkt kann durch ihre Position r und Orientierung θ bestimmt werden. Daraus ergibt sich eine Gewichtsfunktion, die angibt, mit welcher Wahrscheinlichkeit sich eine Ameise zur Zelle r mit der Pheromondichte $\sigma(r)$ bewegt:

$$W(\sigma) = \left(1 + \frac{\sigma}{1 + \delta\sigma}\right)^\beta$$


Der Parameter β bestimmt die osmotropotaxische Sensitivität und $\frac{1}{\delta}$ die sensorische Kapazität der Ameise, die aussagt, dass die Fähigkeit der Ameise Pheromone zu riechen bei hohen Konzentrationen abnimmt.


Ameisen bevorzugen im Mittel geradeaus zu gehen statt sich häufig um 180 Grad zu drehen. Daher wird die Übergangswahrscheinlichkeit mit Faktoren gewichtet, die genau dies bewirken. Für jede Richtung aus der eine Ameise kommen kann wird eine entsprechende Matrix erstellt, die die konstanten Werte für die entsprechenden Folgeschritte enthält. Siehe Abbildung 3.

Dies führt zu einer normalisierten Funktion, die die Wahrscheinlichkeit des Übergangs von Zelle k nach i bestimmt.

$$P_{ik} = \frac{W(\sigma_i)w(\Delta_i)}{\sum_{j/k} W(\sigma_i)w(\Delta_i)},$$

wobei die Notation j/k die Summe über alle Nachbarschaftszellen bedeutet. In der praktischen Umsetzung wird nun eine unabhängig gewählte Zufallszahl aus dem Intervall $(0, 1)$ gezogen und mit der ersten der acht berechneten Wahrscheinlichkeiten verglichen. Falls die Zufallszahl kleiner ist, wird diese Bewegung ausgeführt. Ist die Zufallszahl größer, werden die ersten beiden Wahrscheinlichkeiten addiert und es wird wieder verglichen, dieses mal jedoch mit der Summe

1/2	1	1/2
1/4		1/4
1/12	1/20	1/12

1/4	1/2	1
1/12		1/2
1/20	1/12	1/4


1/20	1/12	1/4
1/12		1/2
1/4	1/2	1

Abbildung 3: Gewichtsmatrix für die Bewegung einer Ameise in ein umliegendes Feld. Die Richtung, aus der die Ameise auf das mittlere Feld getreten ist, ist grau gekennzeichnet.

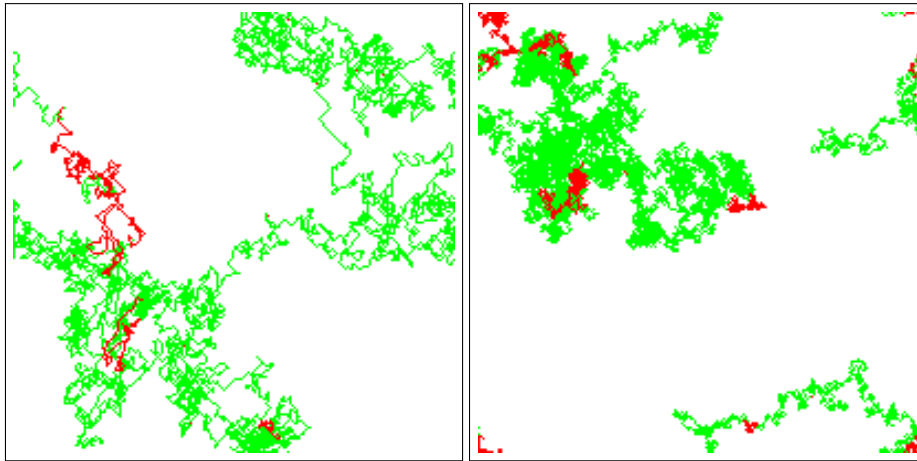


Abbildung 4: Links: Bewegung einer Ameise mit Pheromonspur, Rechts: Ohne Einwirkung von Pheromonspuren anderer Ameisen.

der ersten zwei Wahrscheinlichkeiten, usw. Auf diese Weise wird die Richtung des nächsten Schrittes vorgegeben.

Folgende Werte haben sich für die Variablen der obigen Formeln als kohärent erwiesen: $\beta = 3.5$, $\delta = 0.2$. Für den biologischen Hintergrund dieser Werte siehe[4].

Das Pheromonspurmodell von *Chialvo* wurde von *Ramos*[5] in einem Detail verändert. Statt eine konstante Pheromonspur abzusondern wird eine dynamisch angepasste Menge an Pheromonen abgelegt, die abhängig von der Menge der Objekte in der lokalen Nachbarschaft der betreffenden Ameise ist. Es ergibt sich folgende neue Rate der Absonderung:

$$\eta_{dyn} = \eta + p\Delta$$

mit konstantem Wert $p = 0.0025$ und Δ gleich der Anzahl der Objekte in der lokalen Nachbarschaft.

Erreicht wird damit, dass eine Ameisen ihre Bewegungen nicht rein nach der Pheromonspur anderer Ameisen entscheidet, sondern die Menge der umliegenden Objekte mit einbezogen wird. Dadurch entsteht ein emergentes und

selbstregelndes Massenverhalten um Objektgruppen herum.

Die Veränderung der Laufwege der Ameisen lässt sich auf den folgenden Auszügen erkennen:

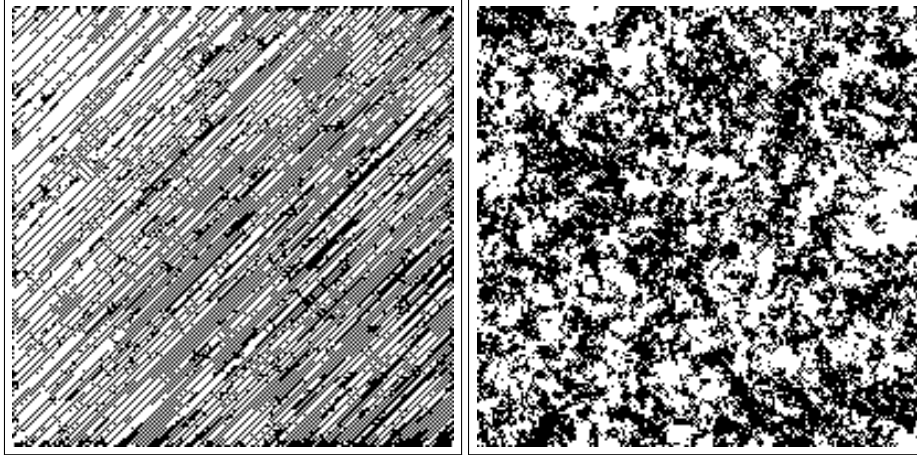


Abbildung 5: Links: Mit dynamischer Anpassung, Rechts: Ohne. Links kann eine Richtung erkannt werden, in die sich die Ameisen bevorzugt fortbewegen.

3.6 Picking und Dropping

Die Entscheidung, ob ein Datenpunkt aufgenommen oder abgelegt werden soll, ist eine der wichtigsten Komponenten dieses Algorithmus. Hier wird entschieden, ob sich Datenpunkte bewegen sollen und damit überhaupt die Grundlage für eine mögliche Clusterung gelegt. Es wurden zwei Varianten implementiert, die im Folgenden vorgestellt werden.

Model von Deneubourg & Lumer/Faieta Das Basismodel von *Deneubourg*[6] sieht folgende Wahrscheinlichkeiten vor, mit der Daten aufgenommen bzw. abgelegt werden:

$$P_p = \left(\frac{k_1}{k_1 + f} \right)^2, \quad P_d = \left(\frac{f}{k_2 + f} \right)^2, \quad (1)$$

wobei k_1 und k_2 benutzerdefinierte Schwellwerte darstellen und f die Anzahl der Objekte in der lokalen Umgebung. Wenn nun $f \ll k_1$, wird P_p nahe bei 1 liegen, d.h. das Objekt wird sehr wahrscheinlich aufgenommen. Wenn $f \gg k_1$, wird P_p nahe an 0 liegen, d.h. das Objekt wird wahrscheinlich nicht aufgenommen. Analog verhält es sich beim Ablegen der Objekte.

f kann auf viele verschiedene Arten definiert werden und muss nicht zwingend die Anzahl der benachbarten Element entsprechen. Allgemein sollte es ein Maß für die lokale Nachbarschaft sein, mit deren Hilfe eine Entscheidung getroffen werden kann. *Lumer* und *Faieta*[7] schlagen eine Methode zur Bestimmung von f vor, die die durchschnittliche Ähnlichkeit von einem Objekt o_i mit allen anderen Objekten o_j in der Nachbarschaft bestimmt:

$$f(o_i) = \max \left\{ 0, \frac{1}{s^2} \sum_{o_j \in \text{Neigh}(r)} \left[1 - \frac{d(o_i, o_j)}{\alpha} \right] \right\},$$

wobei s die Anzahl der Objekte in der lokalen Umgebung und α den maximalen Abstand d zwischen zwei Objekten angibt.

Damit ist eine Methode gegeben, die nicht die Anzahl der Elemente sondern die lokale Ähnlichkeit als Maß verwendet. *Lumer* und *Faieta* verwenden zusätzlich eine veränderte Wahrscheinlichkeit für das Ablegen von Elementen:

$$P_d = \begin{cases} 2f(o_i) & \text{für } f < k_2 \\ 1 & \text{sonst} \end{cases} \quad (2)$$

Model Vitorino Ramos *Vitorino Ramos*[3] schlägt hier einige Verbesserungen vor, die als Alternative zu den obigen Verfahren implementiert wurden.

Da die obigen Verfahren nicht in Betracht ziehen, wieviele Elemente im Verhältnis zur Fläche in einer lokalen Nachbarschaft liegen, kann es durchaus vorkommen, dass zwei unterschiedliche Verteilungen eine numerische Ähnlichkeit aufweisen. *Ramos* schlägt hier vor, dass eben auch dieses Verhältnis in die Berechnung mit aufgenommen wird. Weiter kann in der Natur ein Verhalten von Ameisen beobachtet werden, das als Motivation für dieses Verfahren dient. Ameisen der Gattung *Pheidole* besitzen eine Art Schwellwert, ab welchen sie sich einer Aufgabe widmen. Dieser Schwellwert wird überschritten, wenn ein Reiz s eine bestimmte Intensität erreicht. Dies kann z.B. die Anzahl an Elementen in der lokalen Nachbarschaft oder die Menge an abgesonderten Pheromonen sein.

Aus diesen Beobachtungen werden die folgenden Wahrscheinlichkeiten abgeleitet:

$$\chi = \frac{n^2}{n^2 + 5^2}, \quad \delta = \left(\frac{k_1}{k_1 + d} \right)^2, \quad \epsilon = \left(\frac{d}{k_2 + d} \right)^2, \quad (3)$$

mit

$$d = \left(\frac{1}{d_{max}} \right) \left[\frac{1}{F} \sum_{i=1}^F (f_a(i) - f_b(i))^2 \right]^{\frac{1}{2}}. \quad (4)$$

Die erste Formel in (3) bestimmt den Aufgabenschwellwert χ für die Ameisen in einer 3x3-Nachbarschaft, wobei n die Anzahl der Datenpunkte in dieser Region bestimmt. *Ramos* gibt hier keine genauen Angaben über die Bestimmung der Konstante 5 im Nenner der Gleichung. Es wird vermutet, dass er die Konstante als die aufgerundete Hälfte des Flächeninhalts (Anzahl der Zellen) der lokalen Nachbarschaft bestimmt.

Gleichung (4) bestimmt die Unterschiede der Objekte zueinander in einer lokalen Nachbarschaft. Damit ist ein weiteres Ähnlichkeitsmaß definiert, das wie f im vorherigen Abschnitt für die anstehenden Entscheidungen verwendet wird. Die zweite und dritte Gleichung in (3) bestimmen nun, ähnlich wie bei *Deneubourg*[6], die Wahrscheinlichkeiten ein Objekt abzulegen bzw. aufzunehmen, jedoch Nenner und Zähler, im Vergleich zu *Deneubourg*, vertauscht

Ramos gibt in [3] verschiedene Kombinationen dieser Gleichungen an, die zu verschieden guten Ergebnissen führten. Für eine genauere Analyse siehe dort.

3.7 Ausreißer und Sackgassen

Durch die probabilistische Natur eines Ameisenalgorithmus können sich Ameisen in Regionen des Gitters festlaufen, da sie keinen geeigneten Punkt zur Ablage ihres Datenpunktes finden. Um eine endlose Suche der Ameise zu vermeiden wird eine Schwelle verwendet, die, wenn überschritten, die Ameise zwingt den Datenpunkt abzulegen. In der Praxis haben sich Werte als brauchbar erwiesen, die ungefähr der Breite des Gitters entsprechen.

3.8 Eigene Verbesserungen

Overstep Bei der Bewegung von Objekten und Ameisen gibt es die Beschränkungen, dass sich keine zwei Ameisen zur gleichen Zeit in einer Zelle befinden sowie keine zwei Objekte zur gleichen Zeit in einer Zelle abgelegt (“gestapelt”) werden dürfen. Es ist aber möglich, eine beladene Ameise auf ein belegtes Feld zu bewegen, es der Ameise in diesem Feld aber zu verbieten, dass es sein Objekt dort ablegt (da schon ein anderes Objekt dort abgelegt wurde). Die Alternative wäre, die Ameise nicht zu bewegen bzw. ein freies Feld (falls es dies überhaupt gibt) zu suchen. Dies ist aber mit einigem Mehraufwand verbunden und verlangsamt die Bewegung der Ameise, was wiederum den Algorithmus verlangsamt.

Wichtig ist jedoch, dass am Ende kein Objekt vergessen wird, was zu einem unvollständigen Endzustand führen würde. Dies kann passieren, wenn die Laufzeit des Algorithmus beendet ist und es noch Ameisen gibt, die Objekte mit sich tragen. Normalerweise werden diese Objekte an der aktuellen Stelle abgelegt. Wenn nun aber eine beladene Ameise genau in diesem Moment in obigem Zustand ist, würde das Objekt in der Zelle mit dem geladenen Objekt überschrieben werden. Dies muss man verhindern, indem man entweder eine freie Stelle in der Nachbarschaft sucht oder diese Objekte in besonderer Weise behandelt.

Region Penalty Diese Verbesserung ist von der Idee gleich der, die *Ramos* in [3], Kapitel 4 vorschlägt. Bei den bisherigen Verfahren zur Berechnung der Pickup und Dropping Wahrscheinlichkeiten wird nicht die Objektdichte der Nachbarschaft betrachtet. Dies ist jedoch wichtig, um dichtere bzw. kompaktere sowie weniger räumlich getrennte Cluster zu erhalten.

Eine einfache, aber im Vergleich zum Vorschlag von *Ramos* qualitativ schlechtere Methode, ist, die berechnete Ähnlichkeit in einer Nachbarschaft umso mehr zu betrafen, wenn das Verhältnis von belegten zu freien Zellen reaktiv groß ist.

Praktisch wurde dies gelöst, indem die Ähnlichkeit um das inverse Verhältnis der belegten zu freien Zellen verringert wurde. Ein kleines Beispiel: Sind in einer 10x10-Nachbarschaft nur 8 Felder mit Objekten belegt, wird die Ähnlichkeit um $100 - 8 = 92\%$ verringert.

Dieses Verfahren führte insgesamt dazu, dass sich kompaktere Cluster gebildet haben. Jedoch ist der Vorschlag von *Ramos* diesem vorzuziehen, da er in der Herleitung sowie dem theoretischen Hintergrund fundierter ist. Eine Kombination der beiden Verfahren sollte noch untersucht werden.

3.9 Parameter

Die verschiedenen Parameter des Algorithmus sind von entscheidender Bedeutung für eine gute Clusterung. Da manche Parameter stark vom verwendeten Datensatz abhängig sind, müssen diese teilweise von Hand eingestellt werden.

Die Bezeichnungen der Parameter beziehen sich auf die im Quellcode verwendeten Variablennamen.

- **GRIDSIZE** - Größe (Seitenlänge) des 2D Feld auf dem sich die Ameisen bewegen. Es hat sich, auch von Ramos bestätigt, ein Wert von $GRIDSIZE = \sqrt{40 * \text{Anzahl der Ameisen}}$ bewährt. Leichte Abweichungen von diesem Maß sind jedoch kein Problem. Eine Verschlechterung der Clusterung ist erst ab etwa einer doppelten Seitenlänge zu beobachten.
- **RUNTIME** - Anzahl Iterationen bis zum Abbruch des Algorithmus. Genauer Wert hängt stark von der Anzahl der Ameisen, Objekte und Features ab.
- **Thresholds K1 und K2** - $K1$ ist der Grenzwert für die Entscheidung, ob ein Objekt aufgenommen wird oder nicht. Ist die Ähnlichkeit merklich kleiner als $K1$, wird das Objekt sehr wahrscheinlich aufgenommen. Ist sie merklich größer als $K1$, wird das Objekt wahrscheinlich nicht aufgenommen. Für $K2$ analog. Siehe auch Kapitel 3.6. Die Werte für diese zwei Parameter müssen für jeden Datensatz von Hand eingestellt werden um optimale Ergebnisse zu erzielen.
- **ALPHA** - Gibt den maximal möglichen Unterschied (euklidischer Abstand) unter allen Objekten an. Er wird zur Normalisierung der Werte verwendet und muss a priori bekannt sein oder separat bestimmt werden.
- **DIAMETER** - Bestimmt die Breite bzw. Höhe der lokalen Nachbarschaft einer Ameise. Sollte eine ungerade Zahl sein, da ein Mittelpunkt (die Position der Ameise) benötigt wird.
- **L** - Anzahl von Schritten bevor eine Ameise sein geladenes Objekt ablegen muss. Verhindert das mögliche Festsitzen in Ecken oder zwischen anderen Objekten.

Weiter gibt es noch diverse Switches zum Um- und Abschalten von verschiedenen Varianten des Algorithmus. So können die Pheromonspur, die Dropping-Methoden, die eigenen Verbesserungen (siehe dazu 3.8) und die verwendete Metrik zur Ähnlichkeitsberechnung definiert werden.

3.10 Manual

Die Implementierung erwartet auf stdin eine Datei (oder eine Abfolge von Strings die genau dem Muster der Datei entsprechen), die den Pfad zum Datensatz sowie diverse Parameter beinhaltet. Die Reihenfolge der aufgeführten Variablen ist dabei wichtig. Desweiteren darf nach dem ersten Whitespace kein weiterer Whitespace folgen. Ein Beispiel für den Golub Datensatz:

```
golub_72_100.data #data_filename
50 #gridsize
```

```

500000 #iterations
8      #population_size
72     #number_of_objecst
100    #number_of_features
3      #diameter
200    #forced_drop
26.1529 #alpha_26.1529
0.18   #k1
0.3    #k2
0      #ssim_quad
1      #drop_method
1      #penalty
2      #metric
1      #pheromone_trail
1      #swarmsim_method

```

In diesem Fall liegt der Datensatz im gleichen Verzeichnis wie das Programm selbst.

4 Visualisierung

Das vorliegende Verfahren eignet sich gut um visuell dargestellt zu werden. Vor allem die entstehende Clusterung auf der 2D-Fläche steht dabei im Blickpunkt, da hier direkt zu sehen ist, ob die Clusterung funktioniert oder nicht.

4.1 ASCII Output

Eine sehr simple Visualisierung ist die Ausgabe der Verteilung der Datenpunkte auf der 2D-Fläche zu Beginn und am Ende der Laufzeit. Dabei wird jeweils jeder Datenpunkt als “*” in einer Textdatei an entsprechender Position gespeichert. Diese Visualisierung erlaubt keine Echtzeitverfolgung der Clusterung, ist aber für die Weiterverarbeitung der Daten hilfreich, da Textdateien leicht geparsed werden können.

4.2 Pixeltoaster

Um eine Visualisierung in Echtzeit zu erlauben, wurde die freie Grafikkbibliothek *Pixeltoaster*¹ eingesetzt. Damit lassen sich beliebige Bewegungen der Ameisen anzeigen. Jeder Pixel in der Anzeige entspricht einer Stelle in einem C++ Vector, womit jede Visualisierung der Ameisen und deren Bewegungen sehr einfach dargestellt werden konnten. Folgende vier Anzeigen (siehe Abbildung 6) wurden implementiert:

- Verteilung aller Datenpunkte
- Bewegung/Spur einer einzelnen Ameise
- Bewegung/Spur aller Ameisen
- Verteilung der Pheromone auf dem Gitter

¹<http://code.google.com/p/pixeltoaster/>

Damit lässt sich neben dem Endprodukt des Verfahrens, der Verteilung der Datenpunkte, auch die Arbeit der Ameisen und deren Hilfsmittel verfolgen.

Mit diversen Parametern lassen sich, bei entsprechender Vorkenntnis über den Datensatz, die Daten einfärben. Auch eine Verlangsamung der Anzeige ist möglich, da in Echtzeit viele Details zu schnell ablaufen würden.

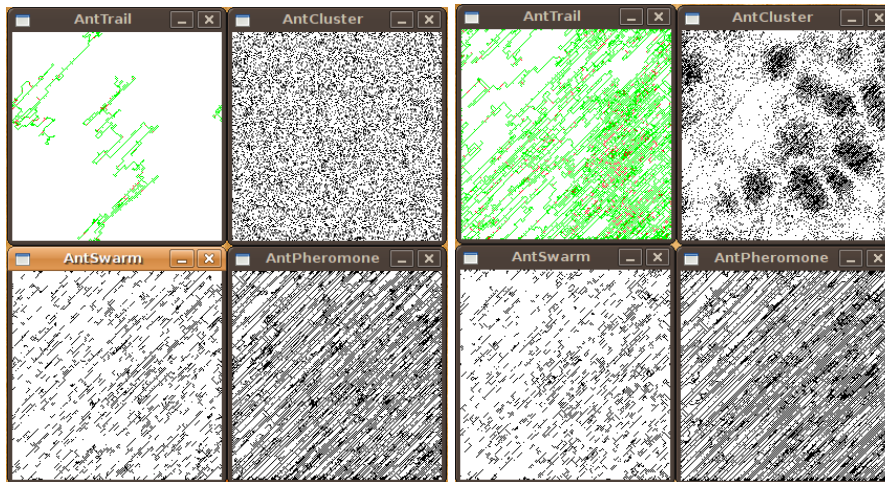


Abbildung 6: Die vier Fenster der Pixeltoaster Visualisierung. Links zu einem frühen Zeitpunkt der Laufzeit, rechts zu einem späteren.

4.3 Ausblick

Eine weitere Visualisierung ist mit der *Cairo* Grafikkbibliothek angedacht (in Verbindung mit GTK+ bzw. GTKmm). Damit könnten die Daten auf beliebige Art und Weise per Vektorgrafiken gezeichnet werden. Auch eine einfache GUI zur grafischen Einstellung der verschiedenen Parameter könnte mit vertretbarem Aufwand entwickelt werden.

Eine weitere, jedoch nur sehr grob angedachte Visualisierung könnte mit *Processing* durchgeführt werden. Dieses Framework kann dreidimensionale und effektreiche Visualisierungen ermöglichen. Der Algorithmus müsste dafür aber in die *Java*-ähnliche Skriptsprache von *Processing* portiert werden. Einwände sind jedoch, dass die damit zu erzielenden Effekte die Aufmerksamkeit von dem eigentlich Ziel des Algorithmus, der Clusterung, ablenkt. Als Alternative, d.h. zusätzliche, Visualisierung würde *Processing* jedoch einen gewissen Mehrwert bieten.

5 Evaluation und Analyse

5.1 Allgemeines

Im folgenden werden die Ergebnisse und Erkenntnisse, die im Laufe des Praktikums entstanden sind, besprochen.

Fragmentierte Cluster Eine grundlegende Erkenntnis ist die, dass sich Gruppen von Datenpunkten, die von einem nicht-probabilistischen Clusteralgorithmus in einen einzigen Cluster einsortiert werden würden, sich bei dem vorliegenden Verfahren über mehr als einen Cluster erstrecken können. Dies ist damit zu begründen, dass die Clusterung im räumlichen Sinne auf einer zweidimensionalen Fläche geschieht und sich dabei an zwei oder mehr räumlich getrennten Orten konkurrierende Cluster bilden können. Bestehende, aber weniger dominante Cluster werden auch nicht mit der Zeit abgebaut und einem dominanteren Cluster zugefügt, da beispielsweise in einer 3x3-Nachbarschaft der Datenpunkt in der Mitte mit sehr hoher Wahrscheinlichkeit nicht mehr bewegt wird, wenn die umliegenden acht Datenpunkte entsprechend ähnlich sind.



Abbildung 7: Fragmentierte Cluster. Ein Datensatz mit zwei klar getrennten Datencluster (a priori bekannt, hier gelb und blau) sind eindeutig geclustert worden, jedoch jeweils in zwei räumlich getrennte Cluster.

Es existieren diverse Vorschläge um dieses Verhalten zu minimieren. Zum einen werden mit der Verwendung von Pheromonspuren die Ameisen in interessantere Bereiche (d.h. mit mehr Datenpunkten) geleitet und bilden somit weniger kleine Cluster. Zum anderen kann das Verhältnis der belegten zu den nicht belegten Zellen in einer lokalen Nachbarschaft in die Berechnungen der Auf- und Ablegewartrscheinlichkeiten mit einbezogen werden und damit weniger dichte bzw. kleine Cluster zu vermeiden. Siehe dazu auch Abschnitt 3.6.

Laufwege der Ameisen Die relative Bewegung der Ameisen kann auf zwei verschiedene Arten bestimmt werden: Zufällig gewürfelt, mit der Einschränkung, dass keine Ameisen kollidieren oder über die entstehende Karte der Pheromonspuren. Jeder der beiden Varianten erzeugt eine andere Art der Bewegung der Ameisenkolonie.

Bei der zufällig gewählten Bewegung der Ameisen kann für eine einzelne Ameise eine nahezu Brown'sche Bewegung angenommen werden (mit Ausnahme der besagten Einschränkungen). Wenn der Laufweg einer Ameise nachverfolgt wird, kann man erkennen, dass der zurückgelegte Pfad durch die vielen Richtungswechseln sehr dicht und kompakt ist. Die Ameise erforscht hier eher die direkte lokale Umgebung und begibt sich nur langsam in weiter entfernte Gebiete.

Wenn die Ameise jedoch auf Pheromonspuren reagiert, erhält man einen Pfad, der viel länger eine bestimmte Richtung beibehält. Dies kann einfach mit der Natur der Pheromonspuren begründet werden. Die Ameise erforscht hier die erweiterte Umgebung und lässt die direkte lokale Nachbarschaft aussen vor. Dies ist natürlich ein gewollter Effekt der Pheromonspuren, da diese Bereiche für die Ameise uninteressant sind, da sie in der näheren Vergangenheit von anderen Ameisen nicht besucht wurden und daher keine oder keine hohe Pheromonspur

aufweisen können.

Eine Kombination dieser beiden Bewegungsarten könnte zu Verbesserungen in der Clusterung führen. Man startet mit der zufälligen Bewegung und wechselt nach einer gewissen Zeit zur Pheromonvariante. Damit wird zuerst lokal nach einzelnen Datenpunkten und später im größeren Maßstab nach ganzen Clustern gesucht. Dies müsste jedoch noch untersucht werden.

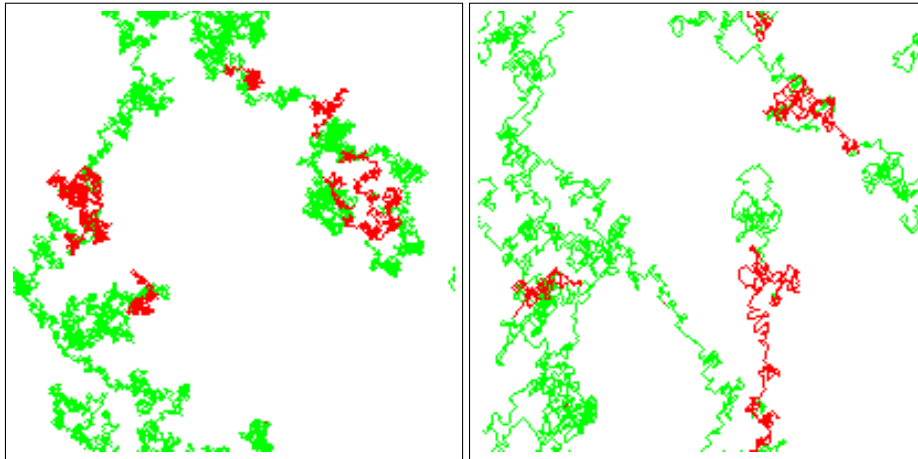


Abbildung 8: Links der Pfad einer einzelnen Ameise mit zufälliger Bewegung, rechts der Pfad unter Verwendung der Pheromonkarte. Rot kennzeichnet eine beladene Ameisen, grün eine unbeladene Ameise.

Parameteranpassung für die vorliegenden Daten Die beiden Schranken k_1 und k_2 müssen in der Regel von Hand für den jeweiligen Datensatz eingestellt werden. Alle zitierten Paper geben keine genaue Begründung an warum sie die Werte so wählen. Die Standards die diese Paper angeben sind zwar oft hinreichend gut, aber wenn die Werte leicht nach oben oder unten verändert werden, kann oftmals eine bessere Clusterung entstehen.

Auch der maximale Euklidische Abstand, der zwischen zwei beliebigen Objekten entstehen kann, muss vorher bekannt sein. Dies erfordert einen kompletten Durchlauf des Datensatzes und einen Vergleich von jedem Objekt mit jedem anderen Objekt (was eine zusätzliche Komplexität von $O(n^2)$ ergibt, mit $n = \text{Anzahl der Objekte}$).

Da die Laufzeit des Algorithmus durch die Anzahl der Iterationen fest vorgegeben ist und kein anderes Abbruchkriterium besteht, muss auch dieser Wert für den jeweiligen Datensatz angepasst werden. Eine Abbruchbedingung, die beispielsweise die Qualität oder den Fortschritt der Clusterung berechnet, ist nicht vorgesehen. In der Regel ist eine minimale Laufzeit von 10.000 bis 100.000 Iterationen bei kleineren Datensätzen notwendig, bis eine Clusterung visuell wahrgenommen werden kann. Bis alle Objekte eine stabile Position eingenommen haben, muss durchaus noch ein vielfaches der genannten Iterationen gewartet werden. Bei großen Datensätzen mit über 1000 Objekten kann eine visuelle Wahrnehmung der Clusterung nochmals länger dauern. Hier sind mehrere Stunden bis Tage keine Seltenheit.

5.2 Parameterauswahl

Die Hauptparameter, wie Anzahl der Ameisen oder Größe der Gitterfläche, können nur durch empirische Erfahrungswerte früherer Läufe gewählt werden. Es haben sich hierbei einige Regeln herausgebildet, die in vielen Fällen gute Ergebnisse erzielen.

Laut [3] sollte die Grundfläche des Gitters in etwa viermal so groß wie die Anzahl der Datenpunkte/Objekte sein, also $A = 4 * n_o$. Die Anzahl der Ameisen kann ungefähr ein vierzigstel der Fläche des Gitters betragen, also $n_a = A/40$. Dies ergibt, dass die Anzahl der Ameisen ca. 10% der Anzahl der Datenpunkte entsprechen sollte.

Diese Werte haben sich in meinen Tests als gute bis sehr gute Werte erweisen. Wenn man die Anzahl der Ameisen bei gleichbleibender Objektanzahl verringert, wird die Clusterung deutlich verlangsamt. Eine Erhöhung der Ameisenanzahl bringt jedoch keine merklichen Geschwindigkeitszuwächse, da sich die Ameisen hier eher gegenseitig blockieren. Weiter steigt auch mit jeder Ameise der Berechnungsaufwand des Algorithmus, womit bei zuvielen Ameisen eher wieder mit Geschwindigkeitseinbußen zu rechnen ist.

5.3 Verwendete Datensätze

Für die Tests wurden folgende Datensätze verwendet:

- Golub mit 72 Objekten und je 100 Features
- Khan mit 83 Objekten und je 96 Features
- Selbst generierter Datensatz mit zwei Features (x,y) und vier a priori bekannten Klassen. Mit diesem Datensatz wurde während der Programmierung das Verhalten der Ameisen getestet.

5.4 Vergleich zu anderen Verfahren

Zum Vergleich des vorliegenden Verfahrens mit bestehenden, nicht-probabilistischen Verfahren wurde der Datensatz zusätzlich mit *k-means* geclustert.

Dazu wurde zuerst der Ameisenalgorithmus gestartet und erst abgebrochen, wenn die Bewegung der Datenpunkte zum Stillstand gekommen sind. Dies bedeutet, dass die Ameisen keine bessere Position für jeden der Datenpunkte mehr finden können. Die Anzahl k der dann gezählten Cluster sowie die 2D-Koordinaten auf der Gitterfläche jedes einzelnen Punktes ergeben den Input für einen ersten Lauf von *k-means*. Damit haben wir jedem räumlichen Cluster auf der 2D-Fläche ein Label zugewiesen.

Nun wird dieses Ergebnis mit dem eines normalen Laufes des *k-means* Verfahrens mit dem Ursprungsdatensatz verglichen. Sind die Labels der einzelnen Datenpunkte gleich, hat das ameisenbasierte Verfahren den Datenpunkt in den gleichen Cluster wie das *k-means*-Verfahren einsortiert.

In der Praxis ist dies jedoch nicht zu 100% der Fall. Nur ca. 80% der Datenpunkte werden gleich gelabelt. Diese Erkenntnis kann vor allem auf die Fragmentierung der Cluster zurückgeführt werden, sowie auf die probabilistische Natur des hier vorgestellten Verfahrens. Es kann durchaus passieren, dass ein Datenpunkt an einer Stelle bzw. in einem Cluster liegen bleibt, da nur seine direkte

Nachbarschaft der Ameise bekannt ist und ein wahrscheinlich besser geeigneter Cluster, der räumlich auf der anderen Seite der Gitterfläche liegt, der Ameise nicht bekannt ist und somit nicht aufgenommen wird.

Alle externen Verfahren (wie etwa *k-means*) wurden mit Matlab durchgeführt. Für die genaue Implementierung dieser Verfahren siehe ebendort. Der Matlab-Code für die Tests ist beigefügt.

5.5 Golub Datensatz

Die Clustering mit dem Golub Datensatz erzeugt in der Regel zwischen zwei und maximal vier Cluster. Eine typische Clustering mit Golub ist auf Abbildung 9 zu sehen. Optimale Ergebnisse sind mit folgenden Einstellungen zu erreichen:

```
golub_72_100.data #data_filename
50      #gridsize
5000000 #iterations
8       #population_size
72      #number_of_objects
100     #number_of_features
3       #diameter
200     #forced_drop
26.1529 #alpha_26.1529
0.18    #k1
0.3     #k2
0       #ssim_quad
1       #drop_method
1       #penalty
2       #metric
1       #pheromone_trail
1       #swarmsim_method
```

Die Anzahl der Iterationen ist absichtlich hoch gewählt, damit sicher ein stabiles Ergebnis zustande kommt. Hier könnte man auch einen niedrigeren Wert wählen.

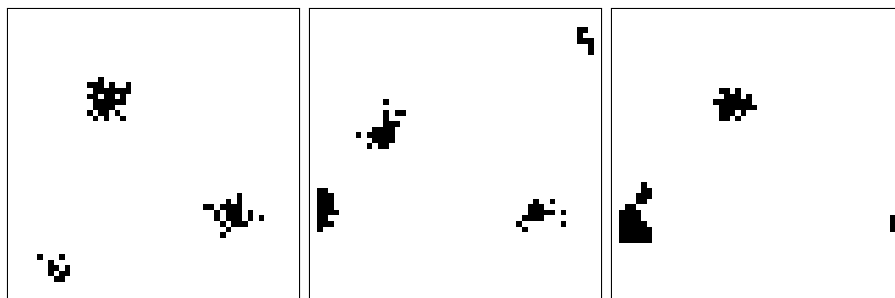


Abbildung 9: Drei Ergebnisse der Clustering des Golub Datensatzes.

Mit der gegebenen Labelung des Golub Datensatzes lässt sich das Ergebnis der Clustering verifizieren. Dabei wurde jeder Datenpunkt entsprechend seines Labels eingefärbt, so dass sich im optimalen Fall drei Cluster mit jeweils

drei unterschiedlichen Farben ergeben würden, wobei in keinem der Cluster ein fremder Farbpunkt auftauchen dürfte. Ergebnisse hierzu siehe Abbildung 10, mit Label *AML* in Rot, Label *B-ALL* in Grün und Label *T-ALL* in Blau. Es ist zu beobachten, dass Datenpunkte mit *B-ALL* Label fast immer einen reinen Cluster bilden, wobei die Label *AML* und *T-ALL* auch gemischt vorkommen können.

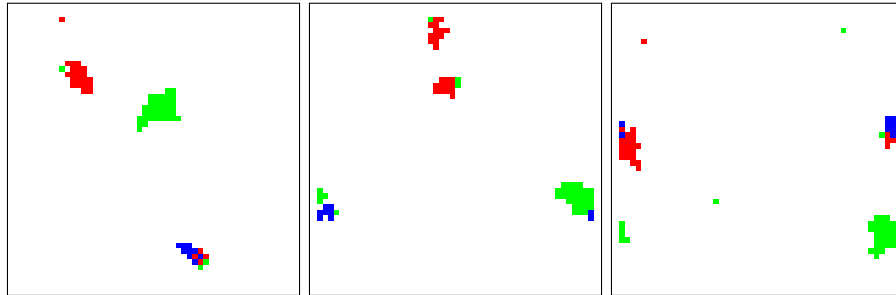


Abbildung 10: Drei Ergebnisse mit eingefärbten Datenpunkten. Rot entspricht Label *AML*, Grün *B-ALL* und Blau *T-ALL*. Die vereinzelt alleine liegenden Datenpunkte sind auf eine noch nicht vollständig abgeschlossene Clustering zurück zu führen.

5.6 Khan Datensatz

Der Khan Datensatz ergibt in der Regeln mehr Cluster als Golub, ca. 5 bis maximal 7 Cluster, siehe Abbildung 11. Die optimalsten Ergebnisse sind mit folgenden Werten zu erzielen:

```
khan_83_stand_96.data #data_filename
50      #gridsize
1000000 #iterations
20      #population_size
83      #number_of_objecst
96      #number_of_features
3       #diameter
200     #forced_drop
24.1642 #alpha
0.18    #k1
0.3     #k2
0       #ssim_quad
1       #drop_method
1       #penalty
2       #metric
1       #pheromone_trail
1       #swarmsim_method
```

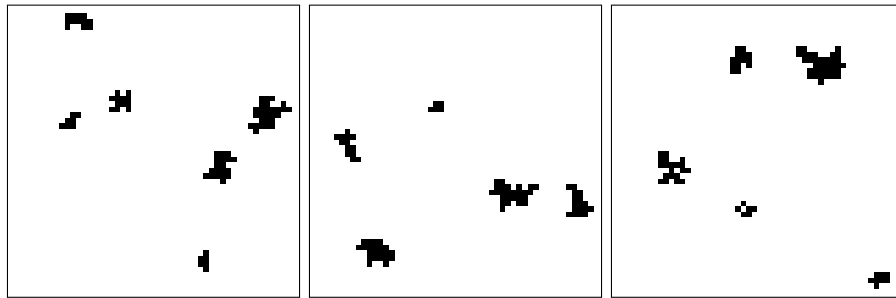


Abbildung 11: Drei Ergebnisse der Clustering des Khan Datensatzes.

6 Interessante Beobachtungen

Während diverser Test mit verschiedenen Datensätzen und Parametereinstellungen sind immer wieder bemerkenswerte sowie unerklärliche Clusterungen und Schwarmbewegungen entstanden. Im folgenden eine unvollständige Auflisten und mögliche Erklärungsversuche.

6.1 Vollständige Partitionierung der Gitterfläche mit Daten

Ein interessanter Effekt ergibt sich, wenn auf einem künstlichen generierten Datensatz, der während der Programmierung zum Testen des Algorithmus generiert wurde, mit speziell gewählten Parametern getestet wird. Der Datensatz besteht aus 2400 Objekten mit je vier disjunkten 600er Gruppen. Innerhalb dieser vier Gruppen sind alle Daten gleich, d.h. es gibt 600 Objekte mit den gleichen Features x und y .

Dieser Datensatz wurde mit den folgenden Parametern getestet:

```
Gridsize: 100x100
Anzahl Ameisen: 240
Lokale Nachbarschaft: 9x9 oder 7x7
Alpha: 552 (maximal möglicher euklidischer Abstand)
K1: 0.1
K2: 0.15
Dropping: Methode von Lumer und Faieta
Region Penalty: Ja
Pheromonspur: Nein
```

Es ergibt sich nun eine Partitionierung der vier Gruppen auf der Gitterfläche. Gleiche Datenpunkte bilden einen eigenen Bereich mit einer schmalen Abgrenzung zum benachbarten Bereich. Innerhalb der einzelnen Bereiche bleiben die Datenpunkte in Unruhe, d.h. sie bewegen sich weiter und enden nicht in einem stabilen Endzustand. Dies ist wahrscheinlich damit zu begründen, dass die Wahrscheinlichkeiten für das Liegenlassen der Datenpunkte zu niedrig bleiben. Dieses Verhalten ließ sich nicht mit Anpassungen der Parameter beheben.

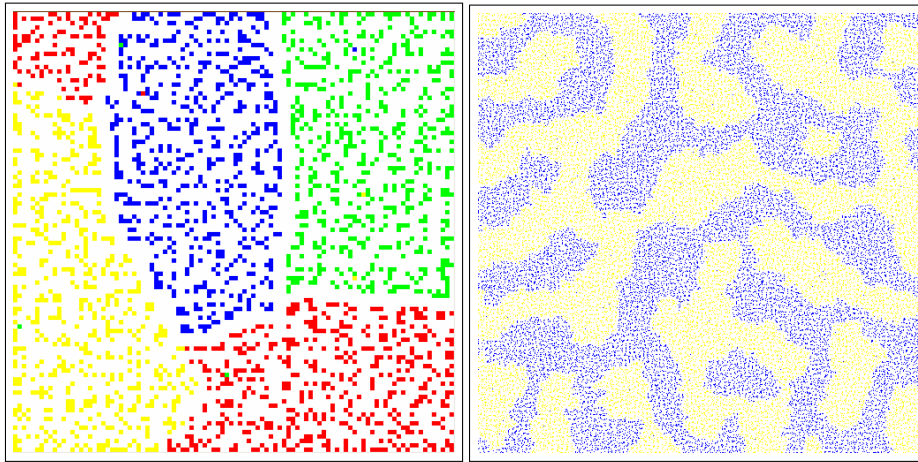


Abbildung 12: Die Daten werden bei besonderer Wahl der Parameter und mit einem künstlichen Datensatz auf die gesamte Fläche verteilt, aber nach Gruppen getrennt.

6.2 Der Eistüteneffekt

Ein spezieller Effekt, der mit einem bestimmten Datensatz und unter bestimmten Parametern aufgetreten ist, habe ich aufgrund des Aussehens des Endergebnisses den “Eistüteneffekt” genannt. Dieser Effekt bietet keine besondere Beachtung um Sinne der Qualität der Clusterung.

Die Daten gruppieren sich hierbei langsam um ein oder mehrere Zentren herum und separieren sich nach und nach in ihre bekannten Cluster, “kleben” aber weiterhin zusammen, siehe Abbildung 13. Eine besondere Formel für die Region Penalty (siehe dazu Abschnitt 3.8) ist für diesen Effekt zuständig. Beim Dropping wie auch beim Picking von Objekten wird die Ähnlichkeit nach folgender Formel “bestraft”:

$$\text{lokale Ähnlichkeit} = \text{lokale Ähnlichkeit} * \left(\frac{\text{Anzahl Elemente}}{\text{Fläche der Nachbarschaft}} * 2 \right)$$

Die weiteren Parameter sind:

```
Datensatz: verySimpleData2k (ähnlicher Datensatz wie oben)
Gridsize: 100x100
Anzahl Objekte: 2400
Anzahl Ameisen: 240
Lokale Nachbarschaft: 9x9
Alpha: 1117.23
K1: 0.5
K2: 0.3
Region Penalty: Ja (siehe oben)
Dropping: Deneubourg
Pheromonspur: Nein
```

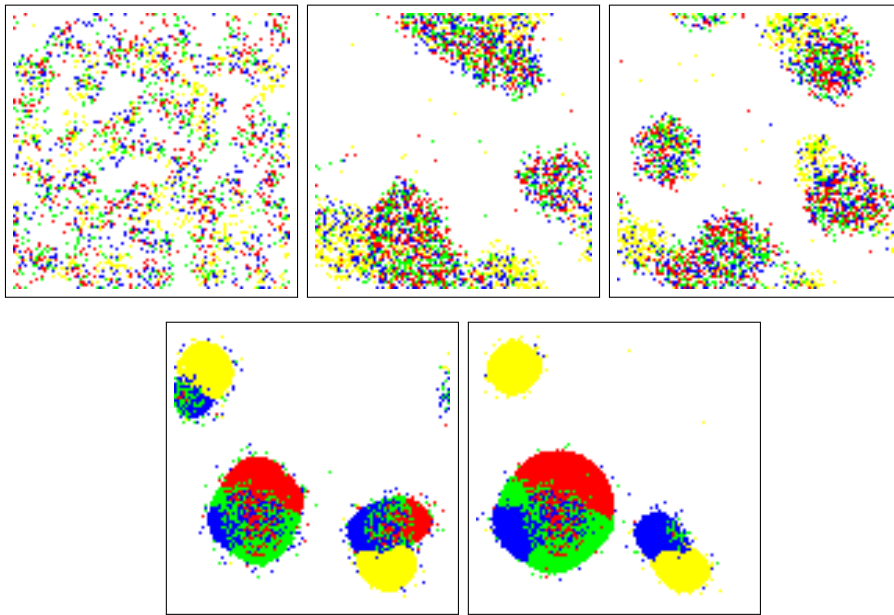


Abbildung 13: Der *Eistüteneffekt* in verschiedenen zeitlichen Zuständen.

7 Ausblick

In mehreren Bereichen dieses Praktikums können noch Verbesserungen sowie Erweiterungen vorgenommen werden. Auf einige möchte ich hier eingehen und dabei beschreiben, warum diese entstanden sind und wie sie umgesetzt werden könnten.

Neben den im Abschnitt 2 vorgestellten Verfahren, kann ein Clusterverfahren auch mit *Particle Swarms* realisiert werden[8], basierend auf dem Verfahren von *Kennedy und Eberhart*[1]. Da diese Vorgehensweise sehr von der hier verwendeten Idee abweicht, wäre es interessant zu sehen, wie sich diese beiden Varianten im Vergleich verhalten würden.

Der hier implementierte Algorithmus verwendet keine Parallelisierung der Aufgaben, somit können moderne Mehrkernprozessoren nicht davon profitieren. Hier müsste nach einer Methode gesucht werden, die es erlaubt, dass das Verfahren parallele Aufgaben gleichzeitig berechnet, aber dabei die Arbeitsweise der Ameisen nicht gestört wird. Es darf also beispielsweise nicht passieren, dass eine Ameise einen Datenpunkt A aufnimmt und eine andere Ameise in einem parallelen Thread auch versucht diesen Datenpunkt A aufzunehmen. Lösen könnte man dieses Problem, indem man die Gitterfläche in mehrere Bereiche (deren Anzahl gleich der Anzahl der gewünschten Threads ist) aufteilt und jeden Bereich für eine Iteration einem Thread zuweist. Die Randbereiche zwischen den einzelnen Bereichen müssten hier jedoch erst einmal ausgelassen werden und in einem nachfolgenden Schritt bearbeitet werden. Damit liese sich, soweit die Theorie, eine Nebenläufigkeit realisieren. Inwieweit dies in der Praxis funktioniert ist aber noch zu untersuchen.

Weiter könnte noch genauer untersucht werden, auf welche Art man die

Bewegung der Ameisen und der Datenpunkt optimieren könnte. Wäre es beispielsweise möglich, mehrere Datenpunkte in einer Zelle abzulegen? Und wie müssten die Ameisen dann entscheiden welche Punkte sie aufnehmen? Oder kann man die Ameisen ausser per Zufall oder Pheromonspur noch auf einer andere Weise bewegen und damit vielleicht besser zu den Clusterzentren locken? Ist es vielleicht sogar besser, wenn man zuerst die Ameisen zufällig bewegt und danach auf das Pheromonmodell wechselt? Alle diese Frage bedürfen weiterer Untersuchungen, da sie eventuell Verbesserungen versprechen.

Literatur

- [1] James Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm Intelligence*. Evolutionary Computation Series. Morgan Kaufman, San Francisco, 2001.
- [2] Ajith Abraham, He Guo, and Hongbo Liu. Swarm intelligence: Foundations, perspectives and applications. In Nadia Nedjah and Luiza de Macedo Mourelle, editors, *Swarm Intelligent Systems*, volume 26 of *Studies in Computational Intelligence*, pages 3–25. Springer, 2006.
- [3] Vitorino Ramos, Fernando Muge, and Pedro Pina. Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies. In Ajith Abraham, Javier Ruiz del Solar, and Mario Köppen, editors, *HIS*, volume 87 of *Frontiers in Artificial Intelligence and Applications*, pages 500–512. IOS Press, 2002.
- [4] Dante R. Chialvo and Mark M. Millonas. How swarms build cognitive maps. In Luc Steel, editor, *The Biology and Technology of Intelligent Autonomous Agents*, volume 144, pages 439–450. Nato ASI Series, 1995.
- [5] Vitorino Ramos and Filipe Almeida. Artificial ant colonies in digital image habitats - a mass behaviour effect study on on pattern recognition. In *In Dorigo, M., Middendorf, M., Stuzle, T. (Eds.): From Ant Colonies to Artificial Ants - 2 nd Int. Wkshp on Ant Algorithms*, pages 113–116, 2000.
- [6] Jean-Louis Deneubourg, Simon Goss, Nigel Franks, A. Sendova-Franks, C. Detrain, and L. Chrétien. The dynamics of collective sorting robot-like ants and ant-like robots. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, pages 356–363, Cambridge, MA, USA, 1990. MIT Press.
- [7] Erik D. Lumer and Baldo Faieta. Diversity and adaptation in populations of clustering ants. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 501–508, Cambridge, MA, USA, 1994. MIT Press.
- [8] Sandra C. M. Cohen and Leandro N. de Castro. Data clustering with particle swarms. In Gary G. Yen, Simon M. Lucas, Gary Fogel, Graham Kendall, Ralf Salomon, Byoung-Tak Zhang, Carlos A. Coello Coello, and Thomas Philip Runarsson, editors, *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 1792–1798. IEEE Press, 16-21 July 2006.