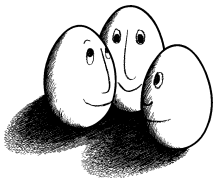
Diplomarbeit

Ein Algorithmus zur Lösung des Farthest-Pair-Problems

Marco Stolpe



Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

9. April 2003

Betreuer:

Prof. Dr. Katharina Morik
Dipl.-Inform. Stefan Rüping

Inhaltsverzeichnis

Einleitung

Hintergrund

Die vorliegende Diplomarbeit ist in Zusammenarbeit mit dem Fachbereich Statistik der Universität Dortmund entstanden. Im Rahmen des Sonderforschungsbereiches 475 ist am Lehrstuhl für Wirtschafts- und Sozialstatistik durch die beiden Mitarbeiter M. Zerbst und L. Tschiersch die Desertifikation der nordafrikanischen Landschaft analysiert und erforscht worden.

Ein besonderer Einflussfaktor ist die Erosion der Böden. Erosion ist ein Vorgang, durch den Gesteine und Mineralien der Erdoberfläche — und damit häufig auch fruchtbarer Mutterboden — abgetragen werden. Natürliche Erosion wird hervorgerufen durch fließendes Wasser, durch Gletscher (Exaration), durch Wind (Deflation) und durch die Brandung an Meeresküsten (Abrasion). Aber auch der Mensch verursacht Erosion durch die landwirtschaftliche Nutzung von Böden. So entstehen Steppen und Trockengebiete, deren Ausbreitung spätestens seit der großen Hungerkatastrophe im Sahel Mitte der 70er und Anfang der 80er Jahre im Mittelpunkt öffentlichen Interesses steht. In diesem Zusammenhang seien verschiedene Aufrufe der UN erwähnt, das wachsende Problem der Erosion in den Griff zu bekommen [UNC94].

Messungen vor Ort können ergänzt werden durch die Erstellung und Analyse von Luftaufnahmen der betroffenen Gebiete. Das langfristige Ziel ist die automatisierte Entdeckung von Erosion. Ein erster Schritt in diese Richtung sind die Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02], in denen die Clusteringverfahren *Maximum-Linkage* (*ML*) und *Advanced-Maximum-Linkage* (*AML*) behandelt werden.

Im Rahmen meiner Tätigkeit als studentische Hilfskraft sind beide Algorithmen in Zusammenarbeit mit M. Zerbst und L. Tschiersch in der Anwendung PICANA (Picture Analysis) implementiert worden. Als alternativer Ansatz für die Clusterung wurden *neuronale Netze* betrachtet.

Problemstellung und Zielsetzung

Anhand verschiedener *Gütekriterien* kann gezeigt werden, dass die Verfahren ML und AML im Bereich der Erosionsproblematik zu besseren Clusterungen als neuronale Netze führen. Empirische Vergleiche mit anderen bekannten Clusteringverfahren blieben bisher aus.

Bei einer geringen Anzahl von Farben in den zu analysierenden Bildern schneiden ML und AML auch bezüglich der Laufzeit besser als neuronale Netze ab [Zer01]. Eine umfassende Analyse der Laufzeit für den besten (*best case*) und schlechtesten Fall (*worst case*) oder die zu erwartende Laufzeit (*average case*) wurde bisher jedoch nicht durchgeführt.

Aus den genannten Punkten ergeben sich die Ziele dieser Diplomarbeit:

1. Analyse der Laufzeit von AML für die Fälle *best case, worst case* und *average case,*
2. Verbesserung der Laufzeit durch Modifikation von AML und
3. empirischer Vergleich (Laufzeit, Güte) mit anderen bekannten Clusteringverfahren.

In den einführenden Kapiteln 1, 2 und 3 werden zunächst die Grundlagen der Clusteranalyse, der digitalen Bildverarbeitung und der algorithmischen Geometrie vorgestellt.

Kapitel 4 geht den zuvor genannten Zielen 1 und 2 nach. Es wird nachgewiesen, dass ML und AML für eine kleine Anzahl zu bestimmender Cluster k quadratische und für eine große Anzahl k im *worst case* sogar kubische Laufzeit haben. Eine Verbesserung der Laufzeit hat große Bedeutung für praktische Anwendungen, insbesondere in komplexeren Problemgebieten als dem der Erosionserkennung. Der Schwerpunkt des restlichen Kapitels (und dieser Arbeit insgesamt) liegt daher auf einer entsprechenden Modifikation von AML.

Dazu wird zunächst gezeigt, dass der erste Schritt des Verfahrens dem aus der algorithmischen Geometrie bekannten *Farthest-Pair-Problem* entspricht. Dafür existieren effiziente Lösungen in der Ebene, die auf Algorithmen zur Berechnung der *konvexen Hülle* einer Punktmenge zurückgreifen. Im dreidimensionalen Raum führt ein *Pruning-Algorithmus* zu guten Laufzeiten. Für höherdimensionale Räume wird ein selbst entwickelter modifizierter Pruning-Algorithmus vorgestellt.

In Kapitel 5 wird der modifizierte AML-Algorithmus (FastAML) gemäß Zielsetzung 3 empirisch mit anderen ausgewählten Clusteringverfahren (AML, *k-Mittelwert*, *EM*, *CLASSIT*, neu-

ronale Netze) verglichen. Dabei kommt das von M. Zerbst und L. Tschiersch entwickelte Gütekriterium der *gewichteten mittleren Minimaldistanz* (*GMMD*) erstmals in größerem Umfang zum Einsatz. Die Algorithmen und die Testumgebung wurden dazu in der Programmiersprache JAVA implementiert.

Da der Schwerpunkt auf der Verbesserung der Laufzeit des AML-Algorithmus liegt, konzentriert sich Kapitel 5 auf die Demonstration herausragender Eigenschaften der verglichenen Algorithmen anhand weniger Beispiele. Zur weiteren Vereinfachung orientiert sich die Arbeit zudem bis auf Ausnahmen weiterhin am Anwendungsgebiet der Erosionserkennung auf Luftaufnahmen.

Das Fazit und der Ausblick in Kapitel 6 fassen die wesentlichen Ergebnisse der Arbeit zusammen und weisen auf weitere Verbesserungen und Anwendungsmöglichkeiten für den FastAML-Algorithmus hin.

Kapitel 1

Clusteranalyse

Der erste Teil dieses Kapitels führt in die Clusteranalyse ein. Der Rest des Kapitels stellt die Verfahren vor, die im Kapitel 5 anhand ihrer Laufzeit und verschiedenen Gütekriterien miteinander verglichen werden.

1.1 Einführung und Überblick

1.1.1 Maschinelles Lernen

Clusteringverfahren lassen sich in das Gebiet der *multivariaten Datenanalyse* und des *maschinellen Lernens* einordnen. Nach Wrobel et al. [WMJ00] beschäftigt sich das maschinelle Lernen mit der „computergestützten Modellierung und Realisierung von Lernphänomenen".

Da sich eine präzise inhaltliche Definition des Begriffs „Lernen" gemäß den zuvor genannten Autoren als schwierig erweist, sei im Folgenden ein Textauszug aus [Fis99] genannt, welcher sich auf unsere intuitive Vorstellung bezieht:

> *„Lernen ist der Prozess, durch Unterweisung, Experimente, Beobachtung oder Erfahrung Wissen oder Fähigkeiten zu erwerben. Das Maschinelle Lernen befasst sich mit der Umsetzung solcher Prozesse auf Maschinen, beziehungsweise in algorithmische Verfahren."*

Die Aufzählung der einzelnen Typen von Lernaufgaben, die bisher Gegenstand des Forschungsgebietes gewesen sind, ist ebenfalls eine Möglichkeit, den Begriff (extensional) zu defi-

nieren [WMJ00]. Eine solche Aufzählung soll und kann im Rahmen dieser Arbeit nicht geleistet werden. Stattdessen werden Unterscheidungen zwischen verschiedenen Typen von Lernverfahren getroffen, die für die Einordnung der in diesem Kapitel vorgestellten Clusteringverfahren hilfreich sind.

Datenorientierte Algorithmen, die basierend auf Mustern in Daten Generalisierungen bilden (Induktion), werden als *ähnlichkeitsbasiert* bezeichnet. Sie lassen sich abgrenzen von Lernverfahren, die Vorwissen, Erklärungen, Analogien oder andere Techniken nutzen [Lug01]. In dieser Arbeit werden ausschließlich ähnlichkeitsbasierte Algorithmen behandelt, da kaum Vorwissen über die Eigenschaften der zu lernenden Daten vorliegt.

Eine andere wichtige Unterscheidung ist die zwischen *überwachtem* und *nicht überwachtem* Lernen.

Überwachtes Lernen (supervised learning). Die Lernaufgabe besteht darin, ein sog. *Zielkonzept* aus einer Menge von klassifizierten Beispielen, der sog. *Trainingsmenge*, zu lernen. Im Gegensatz zum reinen Auswendiglernen der Zuordnung von Objekten zu Klassen wünscht man sich, dass das Lernverfahren von den vorgelegten Beispielen abstrahiert. Unklassifizierte, noch nicht vorgelegte Objekte sollen nach dem Training ebenfalls der richtigen Klasse zugeordnet werden.

Wie gut ein Verfahren gelernt hat, lässt sich mit Hilfe einer *Testmenge* von Objekten überprüfen, deren Klassenzuordnung nur dem „Lehrer“ bekannt ist. Ein Maß für die Güte könnte dabei z. B. die Anzahl der korrekt klassifizierten Objekte aus der Testmenge sein.

Beschreibungen verschiedener überwachter Lernverfahren wie *statistische Modellierung, Entscheidungsbäume, Klassifikationsregeln, lineare Modelle, instanzbasiertes Lernen* oder *konnektionistische Modelle* finden sich u.a. in [WF01], [WMJ00] und [Lug01].

Nicht überwachtes Lernen (unsupervised learning). Die Klassenzugehörigkeit der Objekte ist nicht bekannt und es gibt keinen „Lehrer“. Die Aufgabe des Lernverfahrens ist es, Beziehungen und Gemeinsamkeiten zwischen den Objekten herauszufinden, selbstständig Konzepte zu bilden und diese zu bewerten.

Für das hier beispielhaft ausgewählte Anwendungsgebiet der Erosionserkennung auf Luftaufnahmen eignen sich überwachte Lernverfahren, sofern Bilder vorliegen, bei denen die Klassenzugehörigkeit der darauf abgebildeten Regionen bereits bekannt ist. Dies ist nicht der Fall.

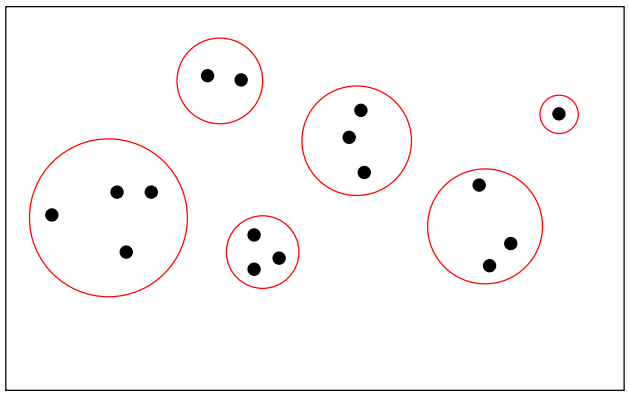


Abb. 1.1: Aufteilung einer Beispielmenge von Instanzen in 6 Cluster

Daher bietet sich der Einsatz von nicht überwachten Lernverfahren an, um dem Substanzwissenschaftler einen besseren Einblick in die Datenlage zu gewähren.

Im Unterschied zur Entdeckung von neuem Wissen über eine Domäne durch sog. *Entdeckungssysteme* (z. B. der Entdeckung naturwissenschaftlicher Gesetze) sollen die Luftaufnahmen in verschiedene Kategorien eingeteilt werden.

Clusteringverfahren sind nicht überwachte Lernverfahren, welche die Ähnlichkeit von Objekten messen und diese darauf basierend in Klassen einteilen. Sie besitzen zudem die Fähigkeit, die Komplexität (Anzahl, Dimension) einer Menge von Daten zu reduzieren. Die den Daten zu Grunde liegende Struktur bleibt dabei weitestgehend erhalten. Dadurch wird eine große Anzahl von Datenpunkten für die weitere Verarbeitung erst handhabbar.

1.1.2 Aufgabenstellung der Clusteranalyse

Gegeben sei eine Menge von Instanzen (Objekten) S aus einem Instanzenraum X. Gesucht ist eine Einteilung der Menge S in Teilmengen („Cluster“) dergestalt, dass die in einem Cluster zusammengefassten Instanzen möglichst homogen sind. Objekte aus verschiedenen Clustern sollten möglichst heterogen sein. Je nach Lernaufgabe können sich die gefundenen Cluster überlappen (*Clumping*) oder eine Partitionierung der Menge von Instanzen in disjunkte Teilmengen bilden (siehe Abb. 1.1).

Formal lässt sich das Problem der Clusterung wie folgt definieren [WMJ00]:

Definition 1.1 (Clusteranalyse) *Sei X ein Instanzenraum und $S \subseteq X$ eine Menge von Instanzen. Sei weiterhin*

$$dist : X \times X \to \mathbb{R}^+$$

eine Abstandsfunktion, und

$$q : 2^{2^X} \to \mathbb{R}$$

eine Qualitätsfunktion. Gegeben S, $dist$ und q besteht die Aufgabe der Clusteranalyse darin, eine Clusterung

$$\mathcal{C} = \{C_1, \ldots, C_k\}, \text{ wobei } C_i \subseteq S \ \ \forall\, i = 1, \ldots, k,$$

zu finden, so dass $q(\mathcal{C})$ maximiert wird, und (optional)

$$C_i \cap C_j = \emptyset \quad \forall \quad i \neq j \in \{1, \ldots, k\} \quad \textit{und}$$
$$\bigcup_{i=1,\ldots,k} C_i = \mathcal{S} \quad \textit{(Partitionierung).}$$

Ein Verfahren zur Bestimmung der besten Partitionierung (im Sinne der gewählten Qualitätsfunktion) lässt sich umittelbar angeben: zähle alle Möglichkeiten auf, die Instanzen aus S in nicht leere Teilmengen einzuteilen. Alle Partitionierungen, welche die Qualitätsfunktion maximieren, sind eine mögliche Lösung. Dieses Verfahren ist nicht praktikabel, da es bereits für eine vorgegebene Anzahl von k Clustern

$$\frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^{|S|}$$

mögliche Partitionierungen gibt [Tsc02, JW92, DH73]. Eine Abschätzung mittels der Formel $k^{|S|}/k!$ ergibt z. B. für die Einteilung von 100 Instanzen in 5 Cluster ungefähr 10^{67} mögliche Partitionierungen [DH73]. Alle weiteren in diesem Kapitel vorgestellten Clusteringverfahren können somit allenfalls Annäherungen an gute Clusterungen finden, aber nicht notwendig die beste Clusterung.

Bei *Clumpingverfahren* können sich die Cluster überlappen. Da gemäß der Zielsetzung dieser Arbeit einige bekannte Clusteringverfahren mit Maximum-Linkage und Advanced-Maximum-Linkage verglichen werden, seien o.B.d.A. Verfahren zur Erzeugung disjunkter Clusterungen betrachtet. In diesem Zusammenhang beziehen sich alle in diesem Kapitel vorgestellten Abstands- und Qualitätsfunktionen auf partitionierende Clusteringverfahren.

1.1.3 Repräsentation der Instanzen

Im Bereich der künstlichen Intelligenz und des maschinellen Lernens gibt es verschiedene Ansätze, Wissen zu repräsentieren. So unterscheidet Luger [Lug01] zwischen *symbolbasiertem*, *konnektionistischem* und *emergentem* Lernen. Im Rahmen der Clusteranalyse stellt sich das Repräsentationsproblem für die Instanzen und die Bemessung ihrer Ähnlichkeit zueinander (siehe Abschnitt 1.1.4).

Symbolbasierte Lernverfahren operieren auf Symbolen, die auf Entitäten und Beziehungen einer Problemdomäne verweisen und durch den Menschen interpretierbar sind. Gesucht ist eine Generalisierung, heuristische Regel oder ein Plan in der Symbolsprache (z. B. Prädikatenlogik, semantische Netze oder Frames), so dass Trainingsbeispiele korrekt klassifiziert werden. Die Arbeitsweise der Verfahren lässt sich in etwa vergleichen mit der eines Mathematikers oder Logikers, der Symbole auf einem Blatt Papier manipuliert und logische Regeln ableitet.

Konnektionistische Modelle hingegen orientieren sich am biologischen Vorbild des (tierischen oder menschlichen) Gehirns. Wissen wird als Muster von Aktivitäten in einem Netz dargestellt, das aus kleinen voneinander unabhängigen Arbeitseinheiten besteht (siehe Abschnitt 1.6). Muster einer Domäne werden nicht symbolisch, sondern als numerische Vektoren kodiert. Die Verbindungen zwischen den Arbeitseinheiten werden numerisch repräsentiert und die Transformation der Muster erfolgt mit Hilfe numerischer Verfahren.

Emergente Modelle des Lernens orientieren sich ebenfalls an einem biologischen Vorbild, nämlich der Genetik und dem Prozess der Evolution. Genetische Algorithmen erzeugen eine Population von Lösungsmöglichkeiten und bewerten die Fähigkeit jedes Individuums, Probleminstanzen zu lösen. Nur die Fähigsten überleben und gehen Verbindungen miteinander ein. So entstehen Generationen von Individuen, die immer leistungsfähiger werden.

Für das uns vorliegende Problem der Erosionserkennung auf Luftaufnahmen ist eine geeignete Repräsentation bereits in den Arbeiten zu diesem Thema [Zer01, Tsc02] erarbeitet worden und ausführlich in Abschnitt 2.5 beschrieben.

Die Art der gewählten Repräsentation für die Objekte (Instanzen) aus S entspricht dabei einer im Bereich der Clusteranalyse und des Data Minings [WF01] häufig verwendeten Darstellung durch d-dimensionale *Merkmalsvektoren*. Die Komponenten der Vektoren entsprechen den möglichen Eigenschaften (Attributen, Features) der repräsentierten Objekte. Jedes der d Attribute kann unterschiedliche Werte aus einem vorgegebenen — meist nominalen oder nume-

		Taube	Henne	Ente	Gans	Eule	Falke	Adler	Fuchs	Hund	Wolf	Katze	Tiger	Löwe	Pferd	Zebra	Kuh
	klein	1	1	1	1	1	1	0	0	0	0	1	0	0	0	0	0
ist	mittel	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0
	groß	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1
	2 Beine	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	4 Beine	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
hat	Haare	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1
	Hufe	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	Mähne	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0
	Federn	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	jagen	0	0	0	0	1	1	1	1	0	1	1	1	1	0	0	0
mag es	rennen	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1	0
zu	fliegen	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0
	schwimmen	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0

Abb. 1.2: Tiere und ihre Eigenschaften (aus [Koh01])

rischen — Wertebereich annehmen. Betrachte dazu das Beispiel in Abb. 1.2. Die verschiedenen Instanzen, in diesem Fall Tiere, können durch Eigenschaften wie Größe, Anzahl der Gliedmaßen, Art der Fortbewegung usw. charakterisiert werden. Die Werte der Attribute sind binär: die Eigenschaft trifft zu oder nicht. Die Auswahl der in diesem Kapitel vorgestellten Distanzfunktionen und Clusteringverfahren ergibt sich aus der genannten Art der Repräsentation.

Die Auswahl der Attribute und ihrer Wertebereiche kann sich als schwierig erweisen. Welche Attribute sind wichtig, um ein Objekt im Sinne der Lernaufgabe ausreichend zu beschreiben? In manchen Anwendungen kann es z. B. sinnvoll sein, die Attribute a priori je nach gewünschtem Einfluss unterschiedlich zu gewichten [Eve86]. Wie viele Attribute sind nötig? Wie ist mit der Korrelation von Werten umzugehen?

Das angesprochene Problem bezieht sich auf die *induktive Voreinstellung* beim Lernen [Lug01], d.h. auf die Vorannahmen, auf die der Programmentwickler oder Substanzwissenschaftler bei der Strukturierung des Lernprozesses zurückgreift. Einerseits ermöglichen diese Annahmen erst den Lernprozess, auf der anderen Seite beschränken sie unter Umständen den Umfang dessen, was gelernt werden kann. So kann etwa eine ungeeignete Repräsentation den Suchraum eines Lernverfahrens so weit beschneiden, dass gute Lösungen nicht gefunden werden. In der Praxis werden geeignete Attribute und deren Wertebereiche bis heute oft rein empirisch bestimmt. Für eine automatische Bestimmung wichtiger Attribute eignet sich die aus der Statistik bekannte Hauptkomponentenanalyse [Eve86, JW92].

1.1.4 Ähnlichkeits- und Abstandsfunktionen

Das in Abschnitt 1.1.3 beschriebene Problem der Auswahl einer geeigneten Repräsentation für Instanzen steht in engem Zusammenhang damit, die Ähnlichkeit oder den Abstand zwischen zwei Objekten bestimmen zu wollen.

Abstandsfunktion Für eine Abstandsfunktion $dist$ müssen folgende Eigenschaften gelten [WMJ00]:

1. $dist : X \times X \rightarrow \mathbb{R}^+, \;\; dist(x,x) = 0 \;\; \forall\, x \in X$
2. $dist(x,y) = dist(y,x) \;\; \forall\, x,y \in X$
3. $dist(x,z) \leq dist(x,y) + dist(y,z) \;\; \forall\, x,y,z \in X$

Um eine Ähnlichkeitsfunktion handelt es sich, wenn nur die ersten beiden Eigenschaften erfüllt sind und der Wertebereich im Intervall $[0,1]$ liegt. Ähnlichkeitsfunktionen werden nicht weiter besprochen, da sie in dieser Arbeit nicht verwendet werden. Definitionen und Beispiele für häufig verwendete Ähnlichkeitsfunktionen finden sich u.a. in [Eve86, WMJ00].

Abstandsfunktionen haben einen beliebigen positiven Wertebereich und sind eine Metrik, da sie zusätzlich die Dreiecks-Ungleichung (Eigenschaft 3) erfüllen. Es gibt viele verschiedene Möglichkeiten, Abstandsfunktionen zu definieren, solange diese den oben genannten Forderungen entsprechen. Beispiele sind etwa die *Hamming-Distanz*, die *City-Block-Metrik* oder die *Mahalanobis-Distanz* [Eve86]. Die Arbeit beschränkt sich auf die folgenden Abstandsmaße, da diese durch die Arbeiten von Tschiersch [Tsc02] und Zerbst [Zer01] für die Anwendung der Erosionserkennung auf Luftaufnahmen bereits als geeignet ermittelt wurden.

Euklidischer Abstand Eine für diese Arbeit und viele Anwendungen relevante Abstandsfunktion ist der *euklidische Abstand*. Für die Punkte $x = (x_1, x_2, \ldots, x_d)$ und $y = (y_1, y_2, \ldots, y_d)$ aus $\mathbb{R}^d$ ist er definiert als:

$$dist(x,y) = \sqrt{\sum_{c=1}^{d} (x_c - y_c)^2} \qquad . \tag{1.1}$$

Der euklidische Abstand wird beeinflusst von einer unterschiedlichen Skalierung der Attribute [DH73, Eve86]. Aus diesem Grund werden die Daten vor einer Verarbeitung durch Clusteringverfahren normalisiert. Dadurch weisen alle Attribute einen Erwartungswert von 0 und eine Standardabweichung von 1 auf.

Gewichteter euklidischer Abstand Neben dem reinen euklidischen Abstand spielt bei den in Abschnitt 1.4 und 1.2.3 beschriebenen Clusteringverfahren Maximum-Linkage und Advanced-Maximum-Linkage der *gewichtete euklidische Abstand* eine Rolle. Ziel ist es, mit Hilfe einer Gewichtungsfunktion f, in die je nach Anwendung unterschiedliche Parameter eingehen können, den Einfluss des euklidischen Abstands zwischen zwei Instanzen abhängig von anderen Größen (z. B. der Häufigkeitsverteilung der Instanzen) unterschiedlich zu bewerten.

Der gewichtete euklidische Abstand $wdist$ ergibt sich dementsprechend durch Multiplikation des euklidischen Abstandes $dist$ mit der je nach Anwendung zu definierenden Gewichtungsfunktion f:

$$wdist(x, y) = f \cdot dist(x, y) \qquad . \tag{1.2}$$

Abstände zwischen Clustern Um die Qualität einer Clusterung bestimmen zu können, müssen nicht nur Abstände zwischen Punkten berechnet werden, sondern auch zwischen Clustern oder Punkten und Clustern.

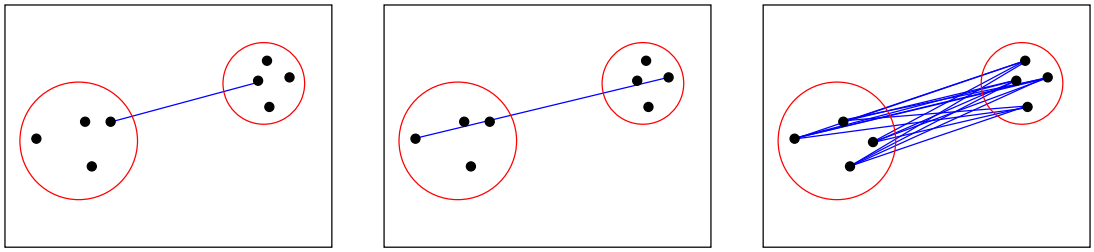
Sofern die Instanzen aus dem Instanzenraum X numerische Attribute haben, lässt sich das Zentrum $z(C)$ einer Menge von Instanzen $C \subseteq X$ durch Mittelung wie folgt berechnen:

$$z(C) = \frac{1}{|C|} \sum_{x \in C} x \qquad . \tag{1.3}$$

Der Abstand zwischen zwei Clustern C_i und C_j lässt sich dann über den Abstand zwischen den *Clusterzentren* $z(C_i)$ und $z(C_j)$ definieren:

$$dist(C_i, C_j) = dist(z(C_i), z(C_j)) \qquad . \tag{1.4}$$

Andere Methoden definieren den Abstand zwischen zwei unterschiedlichen Clustern (siehe Abb. 1.3) als den Abstand zwischen den Punkten der Cluster, die sich am nächsten sind



(a) Nearest neighbour (b) Furthest neighbour (c) Inter-individuell

Abb. 1.3: Distanzmaße zwischen Clustern

(*nearest neighbour distance*, siehe Abb. 1.3(a)) oder am weitesten voneinander entfernt liegen (*furthest neighbour distance*, siehe Abb. 1.3(b)). Auch der Durchschnitt aller inter-individuellen Abstände (siehe Abb. 1.3(c)) zwischen den Punkten der Cluster kann als Abstand definiert werden [Eve86].

1.1.5 Qualitätsfunktionen

Es gibt viele verschiedene Qualitätsfunktionen, die als Maß zur Beurteilung der Güte einer Clusterung eingesetzt werden können. Die Auswahl der Qualitätsfunktion hängt dabei von der jeweiligen Anwendung ab. Im Folgenden sollen einige Qualitätsfunktionen vorgestellt werden, die im Kapitel 5 zur Beurteilung der Güte von Clusterungen herangezogen werden.

Summe der quadrierten Fehler Ein häufig verwendetes Kriterium zur Bestimmung der Qualität von Clusterungen ist die sog. *Intra-Class-Varianz*. Diese basiert auf der Summe der durchschnittlichen quadrierten Abstände der Elemente in einem Cluster i von dessen Zentrum $z(C_i)$ über alle Cluster k (SSW, Sum of Squares Within) [DH73, Zer01]:

$$SSW(\mathcal{C}) = \sum_{i=1}^{k} \sum_{x \in C_i} (x - z(C_i))^2 \quad . \tag{1.5}$$

Das Zentrum eines Clusters repräsentiert diesen gut in dem Sinne, dass es die Summe der Abstände zu allen anderen Instanzen in dem Cluster minimiert. Das Kriterium trägt somit der in Abschnitt 1.1.2 aufgestellten Forderung für eine Clusterung Rechnung, möglichst homogene Cluster zu bilden. D.h. je kleiner die SSW, desto homogener die jeweiligen Cluster. Die SSW eignet sich besonders gut für Clusterprobleme, bei welchen die Instanzen bereits kompakte Gruppen bilden, die deutlich voneinander getrennt sind [DH73].

Summe der quadrierten Fehler zwischen Clustern Die Intra-Class-Varianz kann als Maß für die Homogenität der Elemente in den gebildeten Clustern angesehen werden. Die *Inter-Class-Varianz* hingegen beschreibt die Heterogenität zwischen den gebildeten Clustern. Sie lässt sich definieren als die quadratische Abweichung der Clusterzentren vom Zentrum $z(S) = \frac{1}{|S|}\sum_{x \in S} x$ der Instanzenmenge (SSB, Sum of Squares Between), gewichtet mit der Anzahl der Elemente des jeweiligen Clusters [DH73, Zer01]:

$$SSB(\mathcal{C}) = \sum_{i=1}^{k} |C_i| \cdot (z(C_i) - z(S))^2 \qquad . \tag{1.6}$$

Da es gilt, die Heterogenität der Cluster zu maximieren, lassen sich größere Werte der SSB als bessere Clusterung interpretieren.

Summe der quadrierten Abweichungen aller Instanzen Die Summe der quadrierten Abweichungen aller Instanzen (SST, Sum of Squares Total) ist nichts anderes als die *Gesamtvarianz*, die sich wie folgt berechnen lässt [DH73, Zer01]:

$$SST(S) = \sum_{i=1}^{k} \sum_{x \in C_i} (x - z(X))^2 \qquad . \tag{1.7}$$

Somit gilt $SST(S) = SSW(\mathcal{C}) + SSB(\mathcal{C})$ [DH73, Zer01, Tsc02]. Die Minimierung der Intra-Class-Varianz SSW und die Maximierung der Inter-Class-Varianz SSB bedingen sich gegenseitig. D.h. eine Clusterung mit hoher Homogenität der Elemente innerhalb der Cluster wird auch eine hohe Heterogenität der Cluster aufweisen — und umgekehrt.

Dabei ist allerdings zu beachten, dass die beiden Größen SSW und SSB auch von der Anzahl der Cluster k abhängen. Mit höherer Clusteranzahl steigt die Trennschärfe der Clusterung. Die beiden Ziele, eine hohe Homogenität und eine kleine Anzahl von Klassen zur Beschreibung der Datenmenge zu erreichen, stehen sich also diametral gegenüber.

Nach Tschiersch [Tsc02] handelt es sich bei der Minimierung der Intra-Class-Varianz nur um ein notwendiges Kriterium für die Güte einer Clusterung. So berücksichtigt die Intra-Class-Varianz z. B. keine seltenen Elemente, die im Rahmen der Erosionserkennung auf Luftaufnahmen erhalten bleiben sollen. Laut [Tsc02] bedarf es daher eines weiteren, hinreichenden Kriteriums, um die Güte einer Clusterung im Hinblick auf die vorliegende Anwendung der Erosionserkennung bewerten zu können.

Mittlere Minimaldistanz Ein hinreichendes Kriterium zur Bestimmung der Güte einer Clusterung im Rahmen der Erosionsproblematik ist die Berechnung der von Zerbst [Zer01] und Tschiersch [Tsc02] verwendeten mittleren Minimaldistanz (MMD) zwischen den Clusterzentren:

$$MMD(\mathcal{C}) = \frac{1}{k} \sum_{i=1}^{k} \min_{j \in \{1,\ldots,k\} \setminus i} dist(z(C_i), z(C_j)) \qquad . \tag{1.8}$$

Je größer der Wert der MMD, desto höher ist die Trennung, d.h. der Abstand, zwischen den Clustern und die Chance, seltene Elemente ebenfalls zu berücksichtigen [Tsc02].

Obwohl die MMD als Kriterium zur Beurteilung der Güte von Clusterungen hinreicht, ist sie aufgrund des arithmetischen Mittels äußerst empfindlich gegenüber Ausreißern. Tschiersch [Tsc02] zeigt, dass die MMD die für die Anwendung bessere Clusterung unter Umständen nicht korrekt identifiziert. Dieses Problem hat zu einer Weiterentwicklung der MMD zur *gewichteten mittleren Minimaldistanz* ($GMMD$) geführt, welche auch die Konzentration der Distanzen berücksichtigt. Dieses Kriterium reicht laut [Hei02] sehr nahe an unsere bezüglich der vorliegenden Anwendung intuitive Bewertung von Clusterungen heran.

Gewichtete mittlere Minimaldistanz Das Kriterium der gewichteten mittleren Minimaldistanz ($GMMD$) beruht auf zwei grundsätzlichen Ideen. Zum einen gilt es, das arithmetische Mittel der MMD durch ein gewichtetes, symmetrisches Mittel zu ersetzen, um den Einfluss von Ausreißern zu begrenzen. Zum anderen soll die Konzentration der Distanzen zwischen den Clustern berücksichtigt werden.

Die Berechnung eines gewichteten, symmetrischen Mittels $GMMD_{d_i}$ über die geordneten Distanzen d_i mit $i = 1, \ldots, k$ zwischen den Clusterzentren erfolgt durch Einführung einer Gewichtungsfunktion $w(i)$:

$$GMMD_{d_i}(\mathcal{C}) = \sum_{i=1}^{k} w(i) \cdot d_i, \quad \text{mit} \tag{1.9}$$

$$d_i = \min_{j \in \{1,\ldots,k\} \setminus i} dist(z(C_i), z(C_j)), \quad i = 1, \ldots, k, \quad d_1 \leq d_2 \leq \ldots \leq d_k$$

$$\text{und} \qquad w(i) = \begin{cases} \frac{1}{W}\left(\frac{1-a}{\frac{k+1}{2}-1}\cdot(i-1)+a\right) & , \quad i \leq \frac{k}{2} \\ \frac{1}{W}\left(\frac{a-1}{\frac{k+1}{2}-1}\cdot\left(i-\frac{k+1}{2}\right)+1\right), & i > \frac{k}{2} \end{cases} ,$$

$$\text{mit} \qquad W = \sum_{i \leq \frac{k}{2}} \left(\frac{1-a}{\frac{k+1}{2}-1}\cdot(i-1)+a\right) + \sum_{i > \frac{k}{2}} \left(\frac{a-1}{\frac{k+1}{2}-1}\cdot\left(i-\frac{k+1}{2}\right)+1\right) \quad .$$

Die Gewichtung führt dazu, dass Werte, die am Rand der geordneten Distanzen liegen, weniger stark berücksichtigt werden als Werte in der Mitte. Dadurch schwächt sich der Ausreißereffekt in jeder Richtung ab.

Über den Parameter a, $0 \leq a < 1$ kann der Grad der Berücksichtigung der äußeren Randwerte gesteuert werden. Die bisher entwickelte Qualitätsfunktion kontrolliert, ob die Clusterzentren annähernd äquidistanten Abstand zueinander haben und die Menge der Instanzen S in regelmäßigen Abständen abdecken [Hei02].

Für die Bestimmung der Konzentration der Minimaldistanzen eignet sich der mit der Lorenzkurve verwandte *Gini-Koeffizient* G, $0 \leq G < 1$, wobei für $G = 0$ keine Konzentration und für $G = \frac{k-1}{k}$ totale Konzentration vorliegt [Har85]. Gemäß [Tsc02] ist der Gini-Koeffizient G für das vorliegende Problem gegeben durch:

$$G(\mathcal{C}) = \sum_{i=1}^{k} \left(\frac{i-1}{k} + \frac{i}{k}\right) \cdot \frac{d_i}{\sum_{j=1}^{k} d_j} \quad - 1 \qquad . \tag{1.10}$$

Ein kleiner Wert von G kann unter der Annahme, dass sich die Minimaldistanzen möglichst wenig unterscheiden (bewertet durch $GMMD_{d_i}$), als bessere Clusterung interpretiert werden [Hei02]. Die eigentliche Qualitätsfunktion $GMMD$ ergibt sich daher aus dem Verhältnis von $GMMD_{d_i}$ zum Gini-Koeffizienten G gemäß:

$$GMMD(\mathcal{C}) = \frac{GMMD_{d_i}(\mathcal{C})}{\sqrt{G(\mathcal{C})}} \qquad . \tag{1.11}$$

Mit der $GMMD$ liegt ein Kriterium vor, welches — bezüglich des Anwendungsgebietes der Erosionserkennung — im Wesentlichen der intuitiven Bewertung einer berechneten Auswahl von Clusterzentren durch den Substanzwissenschaftler entspricht [Hei02].

1.2 Iteratives distanzbasiertes Clustering

Iteratives distanzbasiertes Clustering wird auch als *iterative Optimierung* bezeichnet. Der Begriff bezieht sich dabei auf die schrittweise Maximierung einer gegebenen Qualitätsfunktion.

In der Praxis ist es möglich, sich einer optimalen Partitionierung iterativ anzunähern, indem zu Beginn eine zufällige Partitionierung erzeugt wird und danach Instanzen zwischen den Clustern verschoben werden, solange sich der Wert der Qualitätsfunktion verbessert.

Verfahren dieser Art sind auch unter dem Namen *Hill-Climbing* oder *Gradientenabstieg* bekannt [DH73, RN95]. Wie bei allen Hill-Climbing-Verfahren kann es passieren, dass iteratives distanzbasiertes Clustering die Suche in einem lokalen Maximum abbricht. Ein globales Maximum wird dadurch evtl. nicht gefunden. In diesem Zusammenhang hat es sich bewährt, mehrere Durchläufe mit unterschiedlichen zufälligen Partitionierungen zu starten.

1.2.1 k-Mittelwert

Eines der bekanntesten iterativen Optimierungsverfahren der Clusteranalyse ist das sog. *k-Mittelwert-Verfahren*. Die Anzahl der zu berechnenden Cluster ist durch den Parameter k fest vorgegeben. Das Verfahren minimiert (lokal) die Summe der quadrierten Fehler (Formel 1.5). Dies setzt voraus, dass es im Instanzenraum X möglich ist, das Zentrum einer Menge von Instanzen zu berechnen.

Das Verfahren beginnt mit einer zufälligen Auswahl von k initialen Clusterzentren aus dem Instanzenraum X (Initialschritt). Danach weist das Verfahren die Instanzen aus S jeweils einem der k Cluster zu, für welchen der euklidische Abstand (Formel 1.1) zwischen Instanz und Clusterzentrum unter allen aktuellen Clusterzentren minimal ist. Nach erfolgter Zuordnung aller Instanzen werden die Clusterzentren gemäß (Formel 1.3) neu berechnet und bewegen sich damit in Richtung des Datenschwerpunktes [Tsc02]. Dadurch verringert sich die Summe der quadrierten Fehler (Formel 1.5). Der gesamte Vorgang wird solange wiederholt, bis sich die Clusterzentren stabilisiert haben und keiner Veränderung mehr unterliegen.

Bei ungünstiger Auswahl der initialen Clusterzentren konvergiert das Verfahren sehr langsam [WF01, Tsc02]. Zudem können Ausreißer in den Daten dazu führen, dass zusammenhängende Cluster zu Gunsten einer kleineren Summe der quadrierten Fehler (Formel 1.5) aufgetrennt werden [DH73]. Das liegt daran, dass die gewählte Qualitätsfunktion den Abstand zwi-

schen den Clustern (und damit die Heterogenität der Cluster) nicht berücksichtigt. Für die Erkennung von Erosion eignet sich das Verfahren in der klassischen Variante nicht, da die Wahl der Qualitätsfunktion und die wiederholte Mittelwertbildung seltene Elemente verschwinden lässt [Zer01, Tsc02].

Es gibt viele Adaptionen des Verfahrens. Einige versuchen, die Konvergenz durch eine verbesserte Auswahl der initialen Clusterzentren zu beschleunigen [Eve86]. Andere Ansätze, wie der in [Tsc02] erwähnte ISODATA-Ansatz, beschränken die Iterationstiefe durch Anwendung einer veränderten Abbruchbedingung, die zusätzliche Parameter überprüft.

Das Verfahren kann als Instanz des in Abschnitt 1.3.1 vorgestellten EM-Algorithmus angesehen werden [DH73, WMJ00].

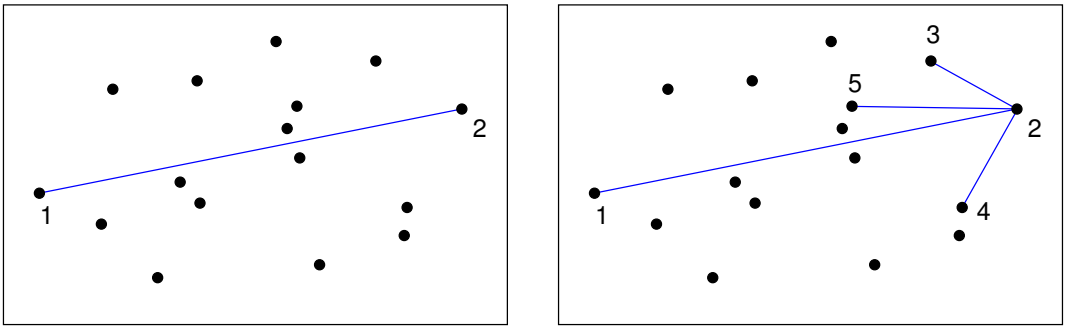
1.2.2 Maximum-Linkage (ML)

Der von Zerbst [Zer01] und Tschiersch [Tsc02] im Zusammenhang mit der Erosionsproblematik vorgestellte Maximum-Linkage-Algorithmus (ML) soll die in Abschnitt 1.2.1 genannten Schwächen des k-Mittelwert-Verfahrens durch eine geeignetere Auswahl der initialen Clusterzentren ausgleichen.

Die Bezeichnung „Linkage“ ist irritierend, da es sich bei ML um keines der in Abschnitt 1.5 vorgestellten klassischen hierarchischen Clusteringverfahren handelt. Heim [Hei02] macht deutlich, dass Maximum-Linkage und Advanced-Maximum-Linkage (siehe Abschnitt 1.2.3) Startwertzerstreuungsalgorithmen für das k-Mittelwert-Verfahren sind, was auch aus der folgenden Darstellung hervorgeht.

Die Grundidee des Verfahrens besteht darin, nicht nur die Summe der quadrierten Abstände (Formel 1.5) zu minimieren, sondern auch den Abstand zwischen den Clustern zu maximieren. Dabei arbeitet das Verfahren auf einer Teilmenge der Instanzenmenge S, nämlich der Menge unterschiedlicher Instanzen U. Die k initialen Clusterzentren werden aus U nacheinander so ausgewählt, dass ihr Abstand zueinander hinsichtlich einer gegebenen Abstandsfunktion maximal ist. Auf diese Weise erreicht das Verfahren eine hohe Trennung der Cluster und eine hohe Abdeckung des gesamten Merkmalsraums. Auch seltene Elemente können nun eigene Cluster bilden.

Sei $Z_k = \{z_1, z_2, \ldots, z_k\}$ die Menge der k zu bestimmenden initialen Clusterzentren und bezeichne $Z_s = \{z_1, z_2, \ldots, z_s\}$ mit $s \in \{1, \ldots, k\}$ die Menge der vom Verfahren bereits



(a) Initialschritt (b) Iterationsschritt ($k = 10, s = 5$)

Abb. 1.4: Die beiden Schritte des ML-Algorithmus

bestimmten s Clusterzentren. Das Verfahren arbeitet wie folgt:

1. *Initialschritt*: Bestimme zwei Instanzen mit maximalem Abstand aus U als Clusterzentren z_1, z_2 (siehe Abb. 1.4(a)), d.h. es gilt:

$$dist(z_1, z_2) = \max_{i,j \in \{1,\ldots,|U|\}} dist(x_i, x_j) \quad \text{mit } x_i, x_j \in U \qquad . \tag{1.12}$$

2. *Iterationsschritt*: Bestimme für jedes bisher gefundene Clusterzentrum aus Z_s den minimalen Abstand Δ_{ij} zu allen Instanzen aus $S \setminus Z_s$:

$$\Delta_{ij} = \min_{i \in \{1,\ldots,|U \setminus Z_s|\}} dist\,(x_i, z_j) \quad \text{mit } x_i \in U \setminus Z_s, \;\; j \in \{1, \ldots, s\} \qquad . \tag{1.13}$$

   Die mit einem Maximum aller minimalen Abstände korrespondierende Instanz wird das neue Clusterzentrum z_{s+1} (siehe Abb. 1.4(b)):

$$Z_{s+1} = Z_s \cup \{z_{s+1}\}, \quad z_{s+1} \in \left\{ x_i \,\middle|\, \max_{j \in \{1,\ldots,s\}} \Delta_{ij} \right\} \subseteq U \setminus Z_s \qquad . \tag{1.14}$$

   Wiederhole den Iterationsschritt, bis alle k initialen Clusterzentren gefunden wurden ($s = k$).

Zur Clusterung der Elemente wird zum Schluss ein einziger Schritt des k-Mittelwert-Verfahrens durchgeführt. Jede Instanz aus $U \setminus Z_k$ wird dem Clusterzentrum aus Z_k mit dem geringsten euklidischen Abstand zugeordnet. Danach werden die neuen Clusterzentren gemäß (Formel 1.3) berechnet.

Im Gegensatz zum klassischen k-Mittelwert-Verfahren bietet die verbesserte Auswahl der initialen Clusterzentren durch den ML-Algorithmus folgende Vorteile:

- Aufgrund weniger Mittelwertbildungen können auch seltene Elemente eigene Cluster bilden (wichtig für Erosionserkennung).
- Der Abstand zwischen den Clustern wird berücksichtigt, was zu einer größeren Heterogenität der Cluster führt.
- Auf den geclusterten Bildern ergeben sich zusammenhängende Farbflächen (wichtig für Erosionserkennung).
- Durch Beschränkung auf einen einzigen Schritt des k-Mittelwert-Verfahrens nach Bestimmung der initialen Clusterzentren verbessert sich die Laufzeit.
- Eine automatische Bestimmung der Clusteranzahl k durch entsprechende Gütekriterien wird möglich. Neue Clusterzentren mit maximalem Abstand zu den bereits bestehenden können jederzeit in einem weiteren Iterationsschritt ermittelt werden.

An dieser Stelle sei noch angemerkt, dass einige der initialen Clusterzentren abhängig von der empirischen Varianz der Instanzen aus S mit Hilfe des durch die relativen Häufigkeiten der Instanzen gewichteten euklidischen Abstandes (Formel 1.2) bestimmt werden sollten. Da die im nächsten Abschnitt 1.2.3 vorgestellte Erweiterung des ML-Algorithmus die Häufigkeiten der Instanzen ebenfalls berücksichtigt und der ursprüngliche ML-Algorithmus für diese Arbeit nicht implementiert wurde, sei der interessierte Leser an dieser Stelle auf die Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02] verwiesen.

1.2.3 Advanced-Maximum-Linkage (AML)

Advanced-Maximum-Linkage (AML) [Tsc02], die Erweiterung des ML-Algorithmus, führt neben der Anzahl k zu bestimmender Cluster einen weiteren Parameter ρ ein. Durch diesen lässt sich die Stärke der Berücksichtigung der den Daten zu Grunde liegenden Häufigkeitsverteilung bei Abstandsberechnungen prozentual steuern.

Sei $\rho \in [0, 1]$, wobei die Häufigkeitsstruktur für $\rho = 0$ gar nicht und für $\rho = 1$ maximal berücksichtigt wird. Dann lässt sich die Funktion f zur Gewichtung des euklidischen Abstandes (Formel 1.2) wie folgt definieren:

$$f(\rho,\lambda) = \rho\lambda + (1-\rho), \text{ mit } \lambda = \begin{cases} \frac{h_i \cdot h_j}{|S|^2}, & \text{im Initialschritt} \\ \frac{h_i}{|S|}, & \text{im Iterationsschritt} \end{cases}, \tag{1.15}$$

wobei $i, j \in \{1, \ldots, |S|\}, i \neq j$ und h_i die absolute Häufigkeit der Instanz $i \in S$ bezeichnet.

In der Praxis hat sich gezeigt, dass der Zusammenhang zwischen der Wahl von ρ und der sichtbaren Auswirkung auf die resultierende Clusterung nicht immer proportional ist. Dieses Problem analysiert [Hei02] und diskutiert mehrere Verbesserungsvorschläge für die Gewichtungsfunktion $f(\rho, \lambda)$.

1.3 Wahrscheinlichkeitsbasiertes Clustering

1.3.1 Erwartungs-Maximierung

Der sog. *EM-Algorithmus* (EM = Erwartungs-Maximierung) basiert auf dem statistischen Modell der *endlichen Mischung* (engl. *finite mixture*) von Wahrscheinlichkeitsverteilungen. Jedem der k zu bestimmenden Cluster ist eine Verteilung zugeordnet, welche die Attributwert-Verteilungen für Instanzen dieses Clusters beschreibt. Zusätzlich gibt es eine Wahrscheinlichkeitsverteilung, welche die relative Population der Cluster repräsentiert, d.h. die Cluster sind nicht gleich wahrscheinlich. Am Ende des Clusterings ist für jede Instanz eine Wahrscheinlichkeit bestimmt, mit welcher die Instanz zu einem bestimmten Cluster gehört.

Für eine Verdeutlichung des Algorithmus wird der einfachste Fall einer endlichen Mischung betrachtet: es gibt ein einziges numerisches Attribut, das für jeden Cluster eine Gauss'sche Normalverteilung aufweist. Die Normalverteilungen der Cluster haben unterschiedliche Parameter, d.h. Mittelwerte und Varianzen. Betrachte dazu Abb. 1.5. Gegeben sind zwei Gauss'sche Normalverteilungen mit unterschiedlichen Mittelwerten (μ_{C_1}, μ_{C_2}) und Standardabweichungen ($\sigma_{C_1}, \sigma_{C_2}$) für die beiden Cluster C_1 und C_2. Man stelle sich vor, dass aus diesen Verteilungen Stichproben entnommen werden. Dabei wird C_1 mit Wahrscheinlichkeit p_{C_1} ausgewählt und C_2 mit Wahrscheinlichkeit p_{C_2}. Es entsteht eine Datenmenge. Das Problem der endlichen Mischung besteht nun umgekehrt darin, für eine gegebene Datenmenge die Parameter $\mu_{C_1}, \sigma_{C_1}, p_{C_1}$ und $\mu_{C_2}, \sigma_{C_2}, p_{C_2}$ zu bestimmen.

Der EM-Algorithmus basiert auf zwei Beobachtungen. Ist bekannt, aus welcher der Verteilungen ein Datenpunkt stammt, so können der Mittelwert und die Standardabweichung gemäß

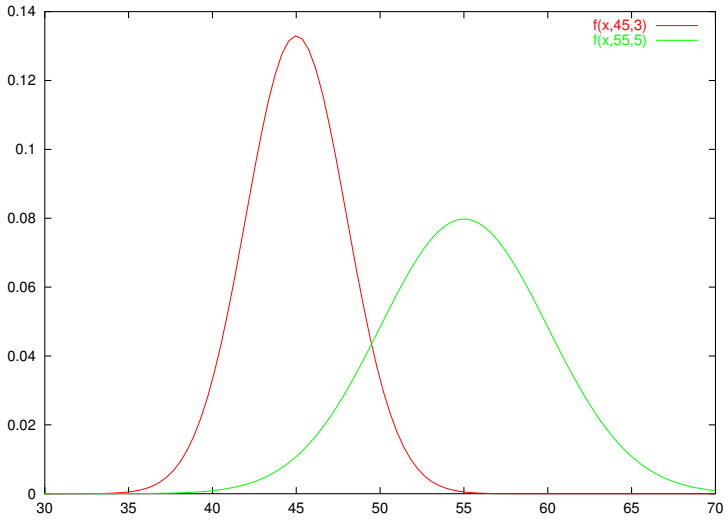


Abb. 1.5: Mischungsmodell aus zwei Normalverteilungen mit $\mu_{C_1} = 0.45, \sigma_{C_1} = 3, p_{C_1} = 0.7$ und $\mu_{C_2} = 0.55, \sigma_{C_2} = 5, p_{C_2} = 0.3$

den aus der Statistik bekannten Formeln aus den Datenpunkten berechnet werden. Sind umgekehrt die Parameter bekannt, so kann die Wahrscheinlichkeit w_{ij} berechnet werden, mit der eine Instanz $x_j \in S$ zu einem Cluster C_i gehört:

$$w_{ij} = P(C_i|x_j) = \frac{P(x_j|C_i) \cdot P(C_i)}{P(x_j)} = \frac{f(x_j; \mu_{C_i}, \sigma_{C_i}) \cdot p_{C_i}}{P(x_j)} \quad . \tag{1.16}$$

Dabei ist $f(x_j; \mu_{C_i}, \sigma_{C_i})$ die Normalverteilungsfunktion für den Cluster C_i:

$$f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\,\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \quad . \tag{1.17}$$

Werden die Zähler $P(C_i|x_j)$ mit $i = 1, \ldots, k$ normiert, indem durch ihre Summe geteilt wird, so verschwindet der Nenner $P(x_j)$ [WF01].

Weder die Herkunft der Datenpunkte noch die Parameter der Verteilungen sind bekannt. Der EM-Algorithmus verfolgt ähnlich wie der in Abschnitt 1.2.1 beschriebene k-Mittelwert-Algorithmus die Strategie, sich iterativ an die Parameter anzunähern:

1. Initialisiere die Parameter der Verteilungen mit zufälligen Werten.
2. Berechne für jede Instanz $x_j \in S$ gemäß Formel 1.16 die Wahrscheinlichkeit w_{ij}, dass die Instanz zu Cluster i gehört — für alle Cluster (Erwartung).

3. Schätze die Parameter basierend auf den in Schritt 2 berechneten Wahrscheinlichkeiten neu ab (Maximierung).

4. Wiederhole die Schritte 1 und 2, bis eine vorgegebene Güte erreicht ist.

Der Mittelwert μ_{C_i} und die Standardabweichung $\sigma^2_{C_i}$ lassen sich für den Cluster C_i wie folgt abschätzen [WF01]:

$$\mu_{C_i} = \frac{w_{i1}x_1 + w_{i2}x_2 + \ldots + w_{i|S|}x_{|S|}}{w_{i1} + w_{i2} + \ldots + w_{i|S|}} \qquad \text{und} \tag{1.18}$$

$$\sigma^2_{C_i} = \frac{w_{i1}(x_1 - \mu_{C_i}) + w_{i2}(x_2 - \mu_{C_i}) + \ldots + w_{i|S|}(x_{|S|} - \mu_{C_i})}{w_{i1} + w_{i2} + \ldots + w_{i|S|}} \quad . \tag{1.19}$$

Als Gütekriterium für den Abbruch des Algorithmus wird die Gesamtwahrscheinlichkeit verwendet, mit welcher die Instanzen aus dieser Datenmenge stammen (unter Berücksichtigung der Schätzungen für die Parameter). Diese Gesamtwahrscheinlichkeit erhöht sich mit jeder Iteration des EM-Algorithmus. Sie ergibt sich aus der Multiplikation der Wahrscheinlichkeiten der einzelnen Instanzen [WF01]:

$$\prod_{j=1}^{|S|} \left(\sum_{i=1}^{k} p_{C_i} P(x_j|C_i) \right) \quad . \tag{1.20}$$

Wird in Formel 1.16 die Wahrscheinlichkeit $P(x_j|C_i)$ einer Instanz durch die Normalverteilungsfunktion ersetzt, so ergibt sich das (rein technische) Problem, dass es sich streng genommen um keine Wahrscheinlichkeit mehr handelt (der Wert liegt nicht zwischen 0 und 1). Es handelt sich in Wirklichkeit um eine sog. *Likelihood-Funktion*. Für die Praxis spielt diese Unterscheidung jedoch keine Rolle [WF01].

Der Algorithmus iteriert so lange, bis der Anstieg der „Gesamtwahrscheinlichkeit“ vernachlässigbar ist, also z. B. bei zehn Iterationen hintereinander unter 10^{-10} liegt.

Wie beim iterativen distanzbasierten Clustering (siehe Abschnitt 1.2) wird eine Hill-Climbing-Suche durchgeführt. Der Algorithmus findet unter Umständen nur ein lokales Maximum. Das Problem kann wie diskutiert behoben werden, indem man den Algorithmus mit mehreren zufälligen Initialisierungen für die Parameter startet und die Clusterung mit dem größten lokalen Maximum für die „Gesamtwahrscheinlichkeit“ als Ergebnis wählt.

Solange die Attribute unabhängig voneinander sind, lässt sich der Algorithmus auf mehrere numerische Attribute erweitern. Die Wahrscheinlichkeit für eine Instanz ergibt sich in diesem Fall durch Multiplikation der Wahrscheinlichkeiten der einzelnen Attribute [WF01]. Die v möglichen Werte nominaler Attribute werden nicht durch eine Normalverteilung, sondern durch v Zahlen charakterisiert, welche die Wahrscheinlichkeiten der Werte darstellen.

Korrelierte Attribute lassen sich ebenfalls verarbeiten. Die Verfahren führen allerdings zu einer sehr großen Anzahl zu bestimmender Parameter, was zum Problem der sog. *Überanpassung* führt. In diesem Zusammenhang wurden Verfahren wie das *Baysssche Clustering* und das AUTOCLASS-Programm entwickelt. Diese Verfahren werden hier nicht vorgestellt, da die Attribute der Instanzen im Rahmen der Erosionserkennung auf Luftaufnahmen unabhängig voneinander sind (siehe Abschnitt 2.5). Für nähere Informationen zu den Verfahren siehe [WF01].

1.4 Inkrementelles Clustering

1.4.1 COBWEB

Das COBWEB-Clusteringverfahren [Fis87] arbeitet im Unterschied zu k-Mittelwert (siehe Abschnitt 1.2.1) nicht iterativ mit der gesamten Datenmenge, sondern inkrementell, d.h. Instanz für Instanz. Die Instanzen werden dabei anhand eines Gütekriteriums, der sog. *Kategoriennützlichkeit* (engl. *Category utility*), in einer baumartigen Hierarchie von Kategorien angeordnet. Dabei bilden die Instanzen die Blätter des Baumes und die Unterkategorien die Knoten bis zum Wurzelknoten. Der Wurzelknoten repräsentiert die gesamte Instanzenmenge S.

Die Kategoriennützlichkeit $CU(\mathcal{C})$ misst die Gesamtqualität einer Aufteilung von Instanzen in Cluster. Sie ist wie folgt definiert:

$$CU(\mathcal{C}) = \sum_h \sum_i \sum_j p(f_i = v_{ij}) \cdot p(f_i = v_{ij}|C_h) \cdot p(C_h|f_i = v_{ij}) \quad . \tag{1.21}$$

Hierzu ist anzumerken, dass COBWEB die Anzahl der zu bildenden Kategorien (= Cluster) selbst ermittelt. Die Summe 1.21 wird über alle Kategorien C_h, alle Eigenschaften f_i der Instanzen und alle Eigenschaftswerte v_{ij} gebildet.

Häufig vorkommende Eigenschaftswerte sollen größeren Einfluss haben. Dazu dient die Gewichtung mit $p(f_i = v_{ij})$. Die sog. *Vorhersagbarkeit* (engl. *predictability*) $p(f_i = v_{ij}|C_h)$

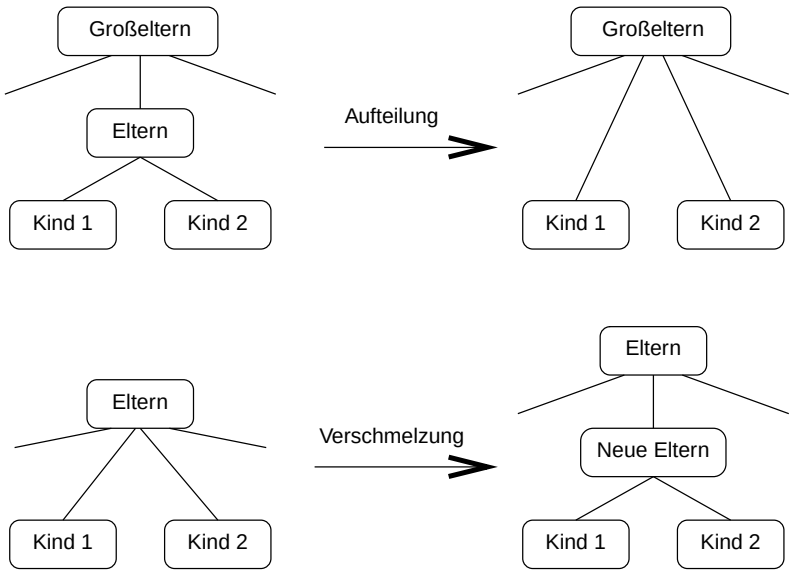


Abb. 1.6: Aufteilung und Verschmelzung von Knoten beim COBWEB-Clusteringverfahren (aus [Lug01])

ist die bedingte Wahrscheinlichkeit, mit der eine Instanz aus Kategorie C_h die Eigenschaft f_i mit dem Wert v_{ij} aufweist. Je höher diese Wahrscheinlichkeit, desto eher weisen zwei Instanzen einer Kategorie dieselben Eigenschaftswerte auf. Die sog. *Prädiktion* (engl. *predictiveness*) beschreibt die bedingte Wahrscheinlichkeit, mit der eine Instanz mit Eigenschaft f_i und dem Wert v_{ij} der Kategorie C_h angehört. Je höher diese Wahrscheinlichkeit, desto weniger wahrscheinlich ist es, dass eine Instanz aus einer anderen Kategorie dieselben Eigenschaftswerte besitzt. Hohe Werte von $CU(\mathcal{C})$ weisen somit darauf hin, dass Instanzen aus derselben Kategorie mit hoher Wahrscheinlichkeit gleiche Eigenschaftswerte haben. Zugleich ist die Wahrscheinlichkeit gering, dass Instanzen unterschiedlicher Kategorien gleiche Eigenschaftswerte haben. Diese Bewertung entspricht der in Abschnitt 1.1.2 aufgestellten Forderung bezüglich der Homogenität und Heterogenität von Elementen in Clustern. Ähnlich wie iterativ distanzbasierte Clusteringverfahren (siehe Abschnitt 1.2) führt COBWEB eine Hill-Climbing-Suche in dem Raum möglicher Einteilungen in Cluster durch.

Der Algorithmus startet mit der ersten Instanz als Wurzel. Danach wird der bis dahin generierte Baum von der Wurzel aus für jede neue Instanz durchlaufen. Auf jeder Ebene wird geprüft, an welcher Stelle die neue Instanz gemäß Formel 1.21 am besten eingefügt werden sollte. Im einfachsten Fall bildet die neue Instanz eine eigene neue Kategorie oder wird einer bereits bestehenden hinzugefügt (d.h. als Blatt angehängt). Bei einer reinen Beschränkung auf diese beiden Operationen wäre die Generierung des Baumes zu sehr von der Reihenfol-

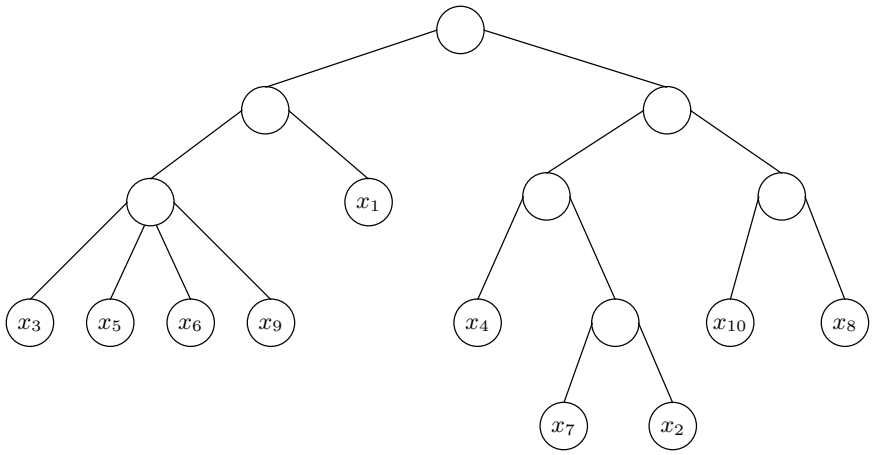


Abb. 1.7: COBWEB-Kategorienhierarchie für 10 Instanzen aus S

ge der gezeigten Trainingsbeispiele abhängig [WF01]. Aus diesem Grund können Teilbäume auch restrukturiert werden. Dazu können zwei bestehende Kategorien *verschmolzen* oder eine bestehende Kategorie in zwei Kategorien *aufgeteilt* werden (siehe Abb. 1.6). In letzterem Fall wird die neue Instanz in die am besten geeignete Kategorie eingefügt. Ein Beispiel für eine von COBWEB erzeugte Hierarchie von Kategorien ist in Abb. 1.7 dargestellt (für ein reales und ausführliches Beispiel siehe [WF01]).

Die Bäume können bei einer großen Anzahl von Instanzen sehr unübersichtlich werden. Schließlich gibt es genauso viele Blätter wie Instanzen. Über den Parameter *Cutoff* kann das Wachstum des Baumes beschränkt werden. Der Parameter legt den Schwellenwert fest, wann sich zwei Instanzen so ähnlich sind, dass keine eigenen Kindknoten für sie gebildet werden müssen. In diesem Fall beinhalten Blattknoten mehrere zueinander ähnliche Instanzen.

Ein Nachteil inkrementeller Methoden ist, dass die Ergebnisse trotz der möglichen Restrukturierung von Teilbäumen noch immer stark von der Reihenfolge der Instanzen abhängen.

1.4.2 CLASSIT

Die numerische Variante des COBWEB-Clusteringverfahrens wird CLASSIT [GLF90] genannt. Für numerische Attribute wird die Kategoriennützlichkeit über eine Schätzung des Mittelwertes und der Standardabweichung für den Attributwert definiert (für die genaue Def. siehe [WF01]). Die Abschätzung der Standardabweichung für Knoten, die nur eine Instanz enthalten, kann Null sein und führt zu unendlich großen Werten der Kategoriennützlichkeit. Das Problem wird durch

einen *Schärfe* genannten Parameter gelöst, durch den jedes Attribut mit einer Mindestvarianz belegt werden kann.

1.5 Hierarchisches Clustering

Statt einer einzigen Clusterung erzeugen *hierarchische Clusteringverfahren* eine Serie ineinander geschachtelter Clusterungen. *Agglomerative Verfahren* starten mit einzelnen Instanzen aus der Instanzenmenge S und vereinigen diese sukzessive zu Gruppen. Am Ende bleibt ein einziger Cluster bestehend aus allen Instanzen übrig. *Divisive Verfahren* hingegen teilen umgekehrt die Instanzen sukzessive in kleinere Gruppen ein, so dass die letzte Clusterung nur aus einzelnen Instanzen besteht. Jede Vereinigung oder Auftrennung hat zum Ziel, eine vorgegebene Qualitätsfunktion zu verbessern. Für eine vorgegebene Anzahl von Clustern stoppen die Verfahren, sobald diese erreicht wurde. Das Ergebnis eines hierarchischen Clusteringverfahrens wird üblicherweise in einem sog. *Dendogramm* (siehe Abb. 1.8) dargestellt.

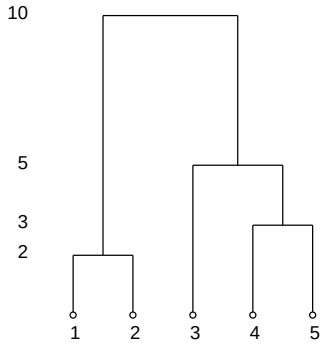


Abb. 1.8: Dendogramm

Hierarchische Clusteringverfahren werden in dieser Arbeit nur der Vollständigkeit halber erwähnt. Aufgrund der hohen Laufzeit der Verfahren sind sie nicht implementiert. Im Folgenden werden einige bekannte agglomerative Verfahren vorgestellt. Für weitere Informationen zu divisiven Verfahren sei auf [Eve86] verwiesen.

1.5.1 Single-Linkage

Agglomerative Methoden berechnen eine Ähnlichkeits- oder Distanzmatrix, in welcher die Ähnlichkeit oder der Abstand zwischen den jeweiligen Instanzen eingetragen wird. Danach werden in jedem Schritt des Verfahrens die Instanzen oder Gruppen zusammengefasst, welche die größte Ähnlichkeit haben. Der einzige Unterschied zwischen den agglomerativen Methoden liegt in der Wahl der Ähnlichkeits- oder Abstandsfunktion.

Das Single-Linkage Clusteringverfahren ist auch unter dem Namen *Nearest Neighbour Clustering* bekannt, da es in jedem Schritt Instanzen oder Gruppen von Instanzen mit dem

geringsten Abstand zueinander zusammenfasst. In jedem Schritt wird somit die Anzahl der Gruppen um eine verringert. Dabei ist der Abstand zwischen zwei Gruppen definiert als der Abstand zwischen den beiden nächsten Elementen der Gruppen (siehe Abb. 1.3(a)).

1.5.2 Complete-Linkage

Das Complete-Linkage-Verfahren ist das Gegenteil von Single-Linkage. In jedem Schritt werden die Instanzen oder Gruppen zusammenfasst, die am weitesten voneinander entfernt sind. Der Abstand zwischen zwei Gruppen ist dementsprechend definiert als der Abstand zwischen den beiden am weitesten voneinander entfernt liegenden Elementen der Gruppen (siehe Abb. 1.3(b)). Das Verfahren wird auch *Furthest Neighbour Clustering* genannt.

1.5.3 Average-Linkage

Beim Average-Linkage-Verfahren wird der Abstand zwischen zwei Gruppen definiert als der Durchschnitt aller paarweise gebildeten Abstände zwischen den Elementen der Gruppe (siehe Abb. 1.3(c)).

1.5.4 Ward's Methode

Ward schlägt ein Verfahren vor, welches gewisse Ähnlichkeiten mit dem in Abschnitt 1.2 vorgestellten iterativen distanzbasierten Clustering hat [WMJ00]. In jedem Schritt werden die Instanzen oder Gruppen von Instanzen miteinander vereinigt, welche die Summe der quadrierten Fehler (Formel 1.5) der daraus resultierenden Clusterung minimieren.

1.6 Clustering mit künstlichen neuronalen Netzen

Künstliche neuronale Netze folgen dem konnektionistischen Modell der Wissensrepräsentation und arbeiten nicht mit Symbolen, sondern mit numerischen Methoden und Merkmalsvektoren (siehe Abschnitt 1.1.3). Für die vorliegende Problemstellung der Erosionserkennung auf Luftaufnahmen und der damit verbundenen numerischen Repräsentation der Datenpunkte (siehe Abschnitt 2.5) spielt dieser Unterschied jedoch keine Rolle. Alle in diesem Kapitel vorgestellten Clusteringverfahren können mit numerischen Merkmalsvektoren arbeiten. Zudem gibt es

eine enge Verwandtschaft zwischen der in diesem Abschnitt vorgestellten Kohonen-Lernregel und dem k-Mittelwert-Verfahren [Tsy71].

Den meisten konnektionistischen Modellen gemein ist die verteilte Repräsentation von Wissen. Diese führt zu einer gewissen Parallelität der Verarbeitung, da die Einheiten eines Netzes oder Schichten von Einheiten Eingaben — eine hardwaretechnische Realisierung vorausgesetzt — gleichzeitig und unabhängig voneinander verarbeiten können. Die Verteilung führt außerdem zu einer hohen Fehlertoleranz, da Netze selbst bei Ausfall einzelner Einheiten noch akzeptable Ergebnisse liefern können. Dazu müssen die Netze allerdings entsprechend dimensioniert sein. Bei richtigem Training können neuronale Netze ferner mit verrauschten Daten oder Störungen in den Eingabedaten besser umgehen als konventionelle Algorithmen. Sie lassen sich somit am besten in Problembereichen einsetzen, für die symbolbasierte Modelle weniger geeignet sind. Dazu gehören Wahrnehmungsprobleme oder das Fehlen einer klar definierten Syntax.

Ein Problem konnektionistischer Modelle ist ihre schwierige Interpretierbarkeit. Die verteilte Speicherung von Wissen ermöglicht im Unterschied zum symbolbasierten Lernen kaum den gezielten Einsatz von Hintergrundwissen über eine Problemdomäne. Neuronale Netze werden nicht programmiert, sondern eignen sich Wissen allein über den (oft langwierigen) Vorgang des Trainings an. Eine andere Schwierigkeit kann sich bei der Überführung der Instanzen in numerische Merkmalsvektoren und deren Interpretation ergeben (siehe Abschnitt 1.1.3).

1.6.1 Biologische neuronale Netze

Künstliche neuronale Netze orientieren sich am biologischen Vorbild des tierischen und menschlichen Gehirns, obwohl sie von der Komplexität biologischer Nervensysteme sehr weit abstrahieren und starke Vereinfachungen vornehmen.

Die fundamentale funktionale Einheit (siehe Abb. 1.9) eines jeden Nervensystems, so auch des menschlichen Gehirns, ist die Nervenzelle bzw. das *Neuron*. Der Zellkörper (*soma*) enthält den Zellkern. Vom Zellkörper gehen mehrere *Dendriten* genannte Fasern aus sowie eine einzige lange als *Axon* bezeichnete Faser. Das Axon kann sich am Ende wiederum in mehrere Fasern aufspalten, die über sog. *Synapsen* mit den Körpern anderer Zellen verbunden sind.

Durch elektrochemische Reaktionen werden Signale von einem Neuron zu einem anderen übertragen, wobei chemische Übertragungssubstanzen ausgehend von den Synapsen in die Dendriten eindringen und dabei das Spannungspotential des Zellkörpers entweder erhöhen (excitatorische Synapsen) oder verringern (inhibitorische Synapsen). Erreicht das Spannungspotential

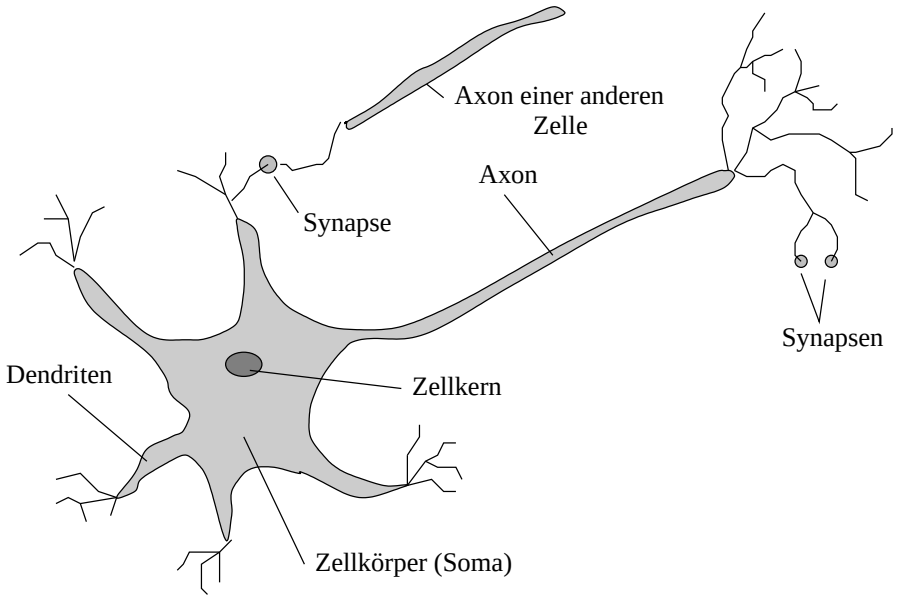


Abb. 1.9: Schematische Darstellung eines Neurons

des Zellkörpers einen bestimmten Schwellwert, so wird ein elektrischer Impuls, das sog. *Aktionspotential*, über das Axon zu anderen Zellen gesendet. Man sagt auch, das Neuron "feuert".

Über die Synapsen kann sich die Stärke der Verbindungen zwischen Neuronen über längere Zeit als Antwort auf bestimmte Stimulationsmuster verändern. Zudem können Neuronen neue Verbindungen mit anderen Neuronen bilden und manchmal bewegen sich sogar ganze Neuronengruppen von einem Ort zum anderen. Man geht davon aus, dass diese Mechanismen die Grundlagen des Lernens in biologischen Nervensystemen darstellen.

Für eine ausführliche Darstellung neurowissenschaftlicher Grundlagen sei auf [DMS01] verwiesen, für eine funktionale und anatomische Beschreibung des menschlichen Gehirns und Nervensystems auf [Tre99].

1.6.2 Künstliche neuronale Netze

Ein künstliches neuronales Netz (KNN) ist nichts anderes als ein gewichteter Graph (für eine Definition siehe [CLRS01]), dessen Knoten (Einheiten) über gewichtete Kanten (Verbindungen) miteinander verbunden sind. Die Grundlage bildet die in Abb. 1.10 gezeigte Einheit eines künstlichen Neurons.

Sei n die Anzahl aller Einheiten und n_i die Anzahl der Eingabesignale für eine Einheit

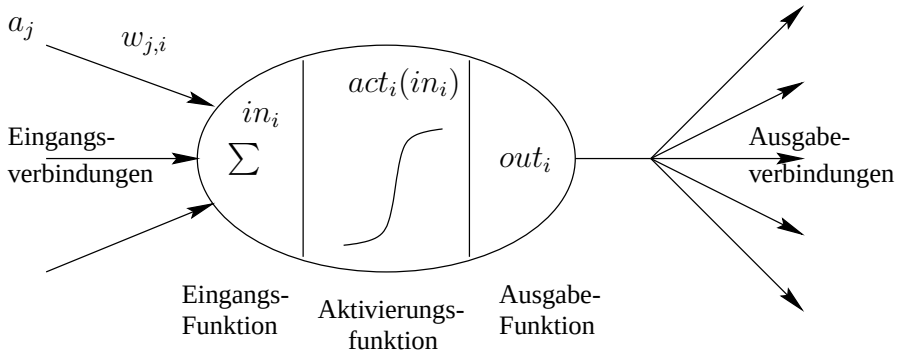


Abb. 1.10: Künstliches Neuron

$i \in \{1, \ldots, n\}$. Die lineare Eingangsfunktion in_i einer Einheit i berechnet die totale gewichtete Summe (die sog. *Aktivierungsebene*) aller n_i Eingangssignale a_j (mit $j \in \{1, \ldots, n_i\}$):

$$in_i = \sum_{j=1}^{n_i} w_{ji} a_j \qquad .$$

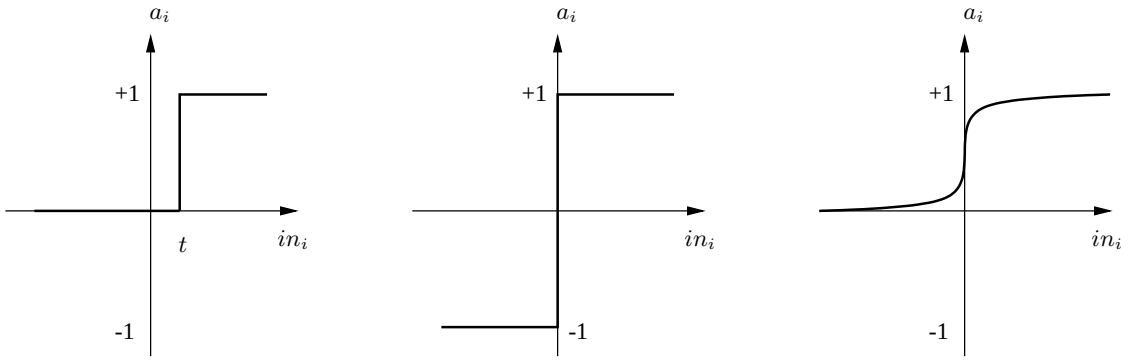
Dabei stellt das reellwertige Gewicht w_{ji} die Verbindungsstärke zwischen einer Einheit j und der Einheit i dar. Die Eingabesignale können aus der Umgebung oder durch Aktivierung anderer Einheiten erzeugt werden. In der Regel sind die Eingaben diskret, aus der Menge $\{0, 1\}$ oder $\{-1, 1\}$ oder reelle Zahlen.

Die *Schwellenwertfunktion* $act_i(in_i)$ bestimmt in Abhängigkeit von der Aktivierungsebene und einem Schwellenwert t, wann eine Einheit entsprechend dem biologischen Vorbild "feuert". Gängige Schwellenwertfunktionen sind die sog. *Step-Funktion* und die *Sign-Funktion*, die beide diskrete Werte $\{0, 1\}$ bzw. $\{-1, 1\}$ liefern oder die *Sigmoid-Funktion*, welche kontinuierliche Werte zwischen 0 und 1 ausgibt (siehe Abb. 1.11).

Die Ausgabe entspricht in der Regel dem Wert der Schwellenwertfunktion (d.h. $out_i = act_i(in_i)$).

Im Laufe der Zeit haben sich viele verschiedene Netzstrukturen (Topologien) und Lernverfahren für die Adaption der Gewichte an Trainingsdaten entwickelt.

McCulloch und Pitts haben 1943 gezeigt, dass ihre Neuronen jede logische Funktion berechnen können [MP43]. Aus diesen Neuronen bestehende Systeme bilden somit ein vollständiges Rechenmodell. Das von Rosenblatt 1958 entwickelte *Perceptron* besteht aus einer einzigen Schicht von Neuronen, die mit einem überwachten Lernverfahren trainiert wird [Ros58]. Nilsson et al. zeigten 1965, dass das Perceptron Probleme linear nicht trennbarer Daten nicht lösen



$$\text{step}_t(x) = \begin{cases} 1, & x \geq t \\ 0, & x < t \end{cases} \qquad \text{sign}(x) = \begin{cases} +1, & x \geq 0 \\ -1, & x < 0 \end{cases} \qquad \text{sigmoid}(x) = \frac{1}{1+e^{-x}}$$

Abb. 1.11: Schwellenwertfunktionen

kann [Nil65]. Mit Hilfe der von Rumelhart et al. entwickelten *Delta-Regel* [RHW86] ist es gelungen, mehrschichtige neuronale Netze (sog. *Feed-Forward-Netze*) durch ein Gradientenabstiegsverfahren (dem *Backpropagation-Algorithmus*) zu trainieren. Mehrschichtige neuronale Netze sind rechnerisch vollständig, d.h. der Klasse der Turing-Maschinen äquivalent [Lug01]. Feed-Forward-Netze werden für die Klassifikation von Daten verwendet.

Es gibt noch viele andere Arten neuronaler Netze und Lernverfahren, die so unterschiedliche Aufgaben wie Mustererkennung, Speicherabruf, Vorhersage, Optimierung oder die Filterung von Rauschen erfüllen. Genannt seien an dieser Stelle das Hebbsche Koinzidenzlernen [Heb49], Assoziativspeicher und lineare Assoziatoren [Koh72, ASRJ77] oder Hopfield-Netze [Hop82].

Für die Clusterung von Daten relevante Modelle nicht überwachten Lernens entsprechen dem sog. *Wettbewerbslernen* (engl. *competitive learning*) und werden im Folgenden vorgestellt.

1.6.3 Alles-dem-Gewinner-Netze

Alles-dem-Gewinner-Netze (engl. *winner-take-all-networks*) [Koh84, HN87] bestehen aus einer Eingabeschicht mit d Einheiten, die den Merkmalen der Instanzen aus S entsprechen, und einer Ausgabeschicht. Die Anzahl der Einheiten der Ausgabeschicht wird im Rahmen der Clusteranalyse als die Anzahl zu bestimmender Cluster k gewählt. Die Einheiten $x_1, x_2, \ldots, x_d$ der Eingabeschicht sind jeweils — wie bei einem Feed-Forward-Netz — über gewichtete Kanten mit allen k Ausgabeeinheiten $O_1, O_2, \ldots, O_k$ verbunden (siehe Abb. 1.12).

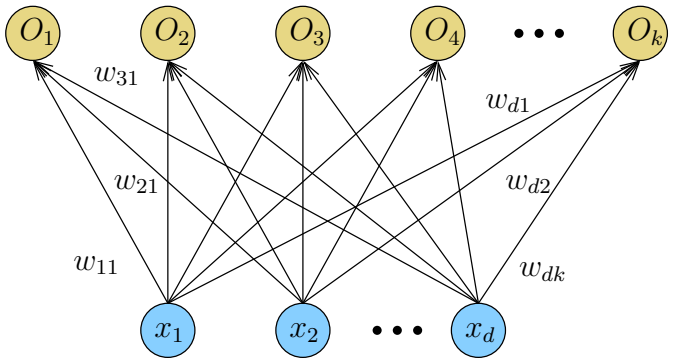


Abb. 1.12: Alles-dem-Gewinner-Netz (aus Gründen der Übersichtlichkeit wurden nicht alle Gewichte w_{ij} eingezeichnet)

Im Unterschied zu Feed-Forward-Netzen stehen die Einheiten der Ausgabeschicht in einem „Wettbewerb“ um die Eingaben. Mit Hilfe eines *Maximalaktivierungstests* [Lug01] wird die Einheit der Ausgabeschicht bestimmt, die am stärksten auf die gerade aktuelle Eingabe reagiert. Der Maximalaktivierungstest wählt die Einheit als Gewinner aus, deren Gewichtsvektor dem Eingabevektor gemäß einem zuvor gewählten Abstandsmaß, z. B. der euklidischen Distanz (Formel 1.1), am nächsten ist. Die Ausgabe der Einheit, die den Wettbewerb gewinnt, ist 1, die der anderen Einheiten 0.

Der Gewinner wird belohnt, indem sein Gewichtsvektor w_i gemäß der sog. *Kohonen-Lernregel* (engl. *Kohonen learning law*) näher an den Eingabevektor rückt:

$$w_i^{neu} = w_i^{alt} + \alpha(x - w_i^{alt}) \qquad . \tag{1.22}$$

Dabei ist α eine kleine positive reellwertige Lernkonstante, die über die Zeit kleiner wird. Sie sorgt dafür, dass sich die Gewichtsvektoren nur langsam und mit zunehmendem Lernfortschritt immer feiner an die Eingaben anpassen. Am Ende des Lernvorgangs repräsentiert jeder der k Gewichtsvektoren einen Teil der Eingabedaten, die auf diese Weise in k unterschiedliche Klassen eingeteilt (geclustert) werden.

Die Grundidee Kohonens besteht darin, die Gewichtsvektoren im $\mathbb{R}^d$ annäherend proportional zur Wahrscheinlichkeitsdichte-Funktion ρ anzuordnen, gemäß der die Eingabevektoren ausgewählt werden [HN89]. Mit anderen Worten soll die Wahrscheinlichkeit dafür, dass ein gemäß ρ ausgewählter Eingabevektor aus $\mathbb{R}^d$ am nächsten an w_i liegt, für alle $i = 1, 2, \ldots, k$ annäherend $1/k$ betragen. Eine möglichst exakte Annäherung an ρ ist interessant, da eine Wahr-

scheinlichkeitsdichte-Funktion sehr viele Informationen über die Daten enthält [HN89].

Es hat sich gezeigt, dass die geforderte Eigenschaft bei zufälliger Initialisierung der Gewichtsvektoren nur unter günstigen Umständen eintritt. Liegen etwa die Gewichtsvektoren alle in einer bestimmten Region von $\mathbb{R}^d$ und stammen alle Trainingsbeispiele aus einer anderen, so kann es passieren, dass nur ein einziger Gewichtsvektor für die Repräsentation der Trainingsbeispiele verwendet wird. Manchmal werden Gewichtsvektoren auch übereinander geschoben und repräsentieren damit dieselben Eingaben. Beide Probleme können durch die Einführung eines sog. *Bewusstseins-Parameters* behoben werden, der in jeder Iteration aktualisiert wird und Einheiten für eine Weile abschaltet, falls sie zu häufig gewinnen [HN89, Lug01]. Laut Hecht-Nielsen [HN89] ordnen sich die Gewichtsvektoren dadurch in nahezu allen Fällen in einer annäherend gleich wahrscheinlichen Konfiguration an.

Im Vergleich mit anderen KNNs weisen Alles-dem-Gewinner-Netze den Nachteil auf, weniger robust gegenüber Ausfällen einzelner Einheiten zu sein: fällt eine Einheit aus, so kann die durch sie repräsentierte Klasse nicht mehr erkannt werden. Die Netze können zudem kein hierarchisches Wissen speichern [HPK91].

1.6.4 Selbstorganisierende Merkmalskarten

Selbstorganisierende Merkmalskarten (engl. *self-organizing feature maps*, SOMs) sind im Wesentlichen eine Variante der Alles-dem-Gewinner-Netze. Im Unterschied dazu sind die Ausgabeeinheiten auf einer Linie oder einem Gitter angeordnet (unterschiedliche Dimensionen und diverse Topologien des Gitters sind möglich). Ziel ist es, die Gewichtsvektoren der Ausgabeeinheiten so anzupassen, dass auf dem Gitter nebeneinander liegende Ausgabeeinheiten zueinander ähnliche Eingabevektoren repräsentieren. In diesem Sinn sollen nicht lineare statistische Beziehungen zwischen höherdimensionalen Eingabedaten in einfache geometrische Beziehungen zwischen den zugehörigen Punkten auf einer niedrig dimensionalen Anzeige umgewandelt werden [Koh01]. Ein Netzwerk, das eine solche Topologie erhaltende Abbildung durchführt, wird *Merkmalskarte* genannt.

Ein Verfahren zum Trainieren von SOMs wurde 1982 von Kohonen vorgestellt [Koh82]. Die Struktur des Netzes entspricht der in Abschnitt 1.12 beschriebenen, bis auf die Anordnung der Ausgabeneuronen auf einem in der Regel zweidimensionalen Gitter (siehe Abb. 1.13). Für einen Eingabevektor x wird wie bei den Alles-dem-Gewinner-Netzen als Gewinner i^* der Ausgabevektor bestimmt, dessen Gewichtsvektor dem Eingabevektor gemäß einer gegebenen

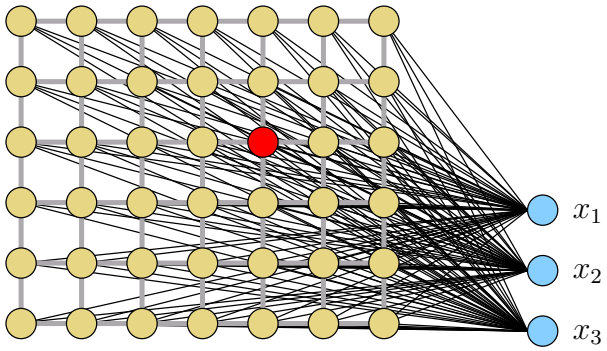


Abb. 1.13: SOM mit drei Eingabeeinheiten, 7×6 Ausgabeeinheiten und einem Gewinner

Abstandsfunktion am nächsten ist.

Der Unterschied liegt in der Lernregel für die Adaption der Gewichtsvektoren. Anstatt nur den Gewichtsvektor des Gewinners zu verändern, werden auch die Gewichtsvektoren von Einheiten in der Nachbarschaft des Gewinners i^* gemäß der folgenden Lernregel angepasst:

$$w_i^{neu} = w_i^{alt} + \alpha\Lambda(i, i^*)(x - w_i^{alt}) \qquad \forall \;\; i \in \{1, \ldots, n\} \quad . \tag{1.23}$$

Die sog. *Nachbarschaftsfunktion* $\Lambda(i, i^*)$ bestimmt dabei die Stärke der Anpassung in Abhängigkeit vom Abstand der jeweiligen Einheit i zum Gewinner i^* auf dem zweidimensionalen Gitter. Die Stärke der Anpassung ist um so größer, je näher die jeweilige Einheit am Gewinner liegt. An dieser Stelle geht die topologische Information ein, so dass benachbarte Einheiten auf ähnliche Weise aktualisiert werden und am Ende des Lernvorgangs auf ähnliche Eingaben reagieren.

Eine typische Wahl für $\Lambda(i, i^*)$ ist:

$$\Lambda(i, i^*) = \exp\left(\frac{-dist(O_i, O_{i^*})^2}{2\sigma^2}\right) \qquad , \tag{1.24}$$

wobei der Parameter σ die Breite des Adaptionsbereiches bestimmt und während des Trainings zunehmend kleiner wird. Der Graph der Funktion ist für eine Merkmalskarte bestehend aus 10×10 Einheiten mit $\sigma = 1$ in Abb. 1.14 dargestellt. Zur besseren Übersicht ist die an sich diskrete Funktion als durchgehender Graph gezeichnet. Die Lernrate α kann in Abhängigkeit vom Lernschritt t als $\alpha(t+1) = 1/\alpha(t)$ gewählt werden, für andere Wahlen siehe [Koh01] oder

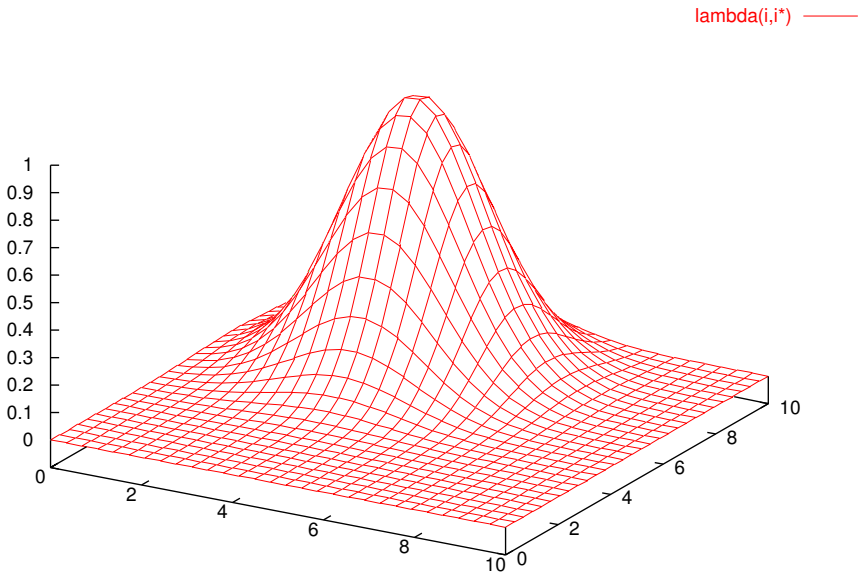


Abb. 1.14: Nachbarschaftsfunktion $\Lambda(i, i^*)$ über dem Gewinner i^* an Position (4,6)

[Tsc02]. Als Distanz zwischen den Einheiten der Ausgabeschicht eignet sich je nach Struktur des Gitters die euklidische Distanz (Formel 1.1). Die Struktur des Gitters ist typischerweise rechteckig oder hexagonal (für eine Diskussion und Abbildungen sei auf [Zer01] verwiesen).

Die Größen $\Lambda(i, i^*)$ und α werden zwecks besserer Konvergenz während des Trainings dynamisch verändert. Zu Beginn startet man in der Regel mit einem weiten Radius für die Nachbarschaftsfunktion und einer hohen Lernrate und verringert die Größen im Laufe des Trainingsvorgangs. Dadurch wird die Karte schnell an die Eingaben angepasst und danach schrittweise verfeinert, bis nur noch Einheiten in unmittelbarer Nachbarschaft des Gewinners und später der Gewinner selbst adaptiert werden.

Kapitel 2

Digitale Bildverarbeitung

Als Beispielanwendung dient in dieser Diplomarbeit die Erkennung von Erosion auf Luftaufnahmen. Die Aufnahmen liegen in digitaler Form zur weiteren Verarbeitung durch Hardware oder Software vor. In diesem Kapitel wird beschrieben, auf welche Weise digitale Bilder auf Rechnern repräsentiert und in welchen Schritten sie verarbeitet werden. Zudem wird diskutiert, auf welche Weise die Bilddaten in Merkmalsvektoren kodiert werden, um sie mit den in Kapitel 1 vorgestellten Clusteringverfahren analysieren zu können.

2.1 Einführung und Übersicht

Mit seinen Augen nimmt der Mensch jeden Tag die Welt um sich herum in Form von Bildern wahr. Von Objekten reflektiertes Licht dringt in das Auge. Die Energie des Lichtstrahls wird von Rezeptoren in elektrische Impulse umgewandelt, die von den Nervenzellen im menschlichen Gehirn verarbeitet werden können.

Die Fotografie und Videotechnik bilden diesen Prozess nach, indem sie die Energie von Lichtstrahlen auf ein Stück Film abbilden. Ein Foto oder Video kann vervielfältigt, bearbeitet und archiviert werden, was bei mehrfacher Wiederholung allerdings zu Qualitätsverlusten führen kann. Digitale Information weist diesen Nachteil nicht auf. Es stellt sich die Frage, welche Möglichkeiten es gibt, digitale Bildinformationen auf einem Rechnersystem zu speichern und zu verarbeiten. Dieser Frage widmet sich die *digitale Bildverarbeitung*.

Ein mögliches Ziel der digitalen Verarbeitung eines Bildes ist die Aufbereitung der in dem Bild enthaltenen Informationen, um sie der Interpretation und Analyse durch Menschen zugäng-

lich zu machen. So könnten etwa Helligkeit und Kontrast verändert, die Intensität der Farben erhöht oder Teile des Bildes eingefärbt werden. Anwendungen ergeben sich etwa in der Medizin, Geographie, Archäologie, Physik, Biologie, Astronomie, in den Rechtswissenschaften, in der Industrie oder beim Militär.

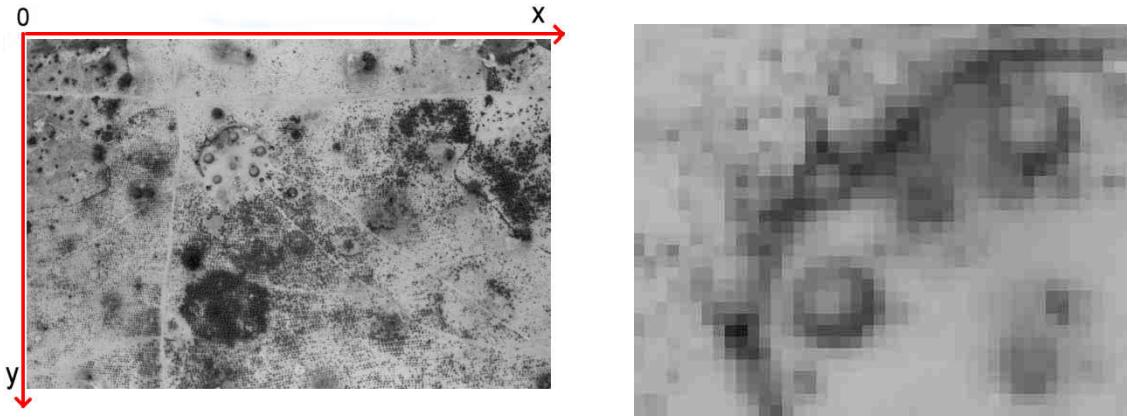
Ein etwas anderes — oft darüber hinausgehendes — Ziel ist die automatische Verarbeitung von Bildern durch Rechner, z. B. die automatische Erkennung von Objekten und die Erstellung einer symbolischen Repräsentation. Beispiele sind etwa die automatische Schrifterkennung, die Erkennung von Fingerabdrücken (Biometrie), die Erstellung von Bilddatenbanken im Sinne des *Information Retrieval* oder die Wettervorhersage anhand von Luftaufnahmen und Satellitenbildern.

Die in Kapitel 1 vorgestellten Clusteringverfahren können als eine Art Vorstufe für die Entwicklung eines Systems angesehen werden, welches Erosion auf Luftaufnahmen automatisch erkennt. Durch die Einteilung der Bildinformation in einige wenige Klassen und der damit verbundenen Informationsreduktion können die Bilder zunächst aber auch durch den Substanzwissenschaftler besser interpretiert werden.

2.2 Digitale Schwarz/Weiß-Bilder

Ein Schwarz/Weiß-Foto, welches mit einer Fotokamera aufgenommen wurde, enthält verschiedene Grauwerte. Das Licht wird von der Oberfläche des Fotos reflektiert und trifft mit unterschiedlicher Intensität auf die Netzhaut des menschlichen Auges. Der Mensch nimmt die verschiedenen Grauwerte als Unterschiede in der Helligkeit wahr. Ein *monochromes Bild* lässt sich entsprechend beschreiben als eine Funktion, die über räumliche Koordinaten x, y jedem Punkt des Bildes einen Wert z proportional zur Helligkeit zuordnet (für die Orientierung der Achsen des Koordinatensystems siehe Abb. 2.1(a)). Da Licht eine Form von Energie ist, ist dieser Wert positiv und endlich.

Für *digitale Bilder* muss der kontinuierliche Definitions- und Wertebereich der Funktion diskretisiert werden. Damit stammen die Koordinaten $x \in \{1, \ldots, m\}$ und $y \in \{1, \ldots, n\}$ aus einer endlichen Menge natürlicher Zahlen und die Helligkeitswerte $z \in \{1, \ldots, b\}$ ebenfalls. Die einzelnen *Bildelemente* (engl. *picture elements*, auch mit *Pixel* abgekürzt), befinden sich dann in einer Matrix, welche für jedes Pixel den dazugehörigen diskretisierten Helligkeitswert enthält.



(a) Orientierung des Koordinatensystems

(b) Über ein Raster werden die analogen Bildinformationen aus Abb. 2.1(a) in einzelne Pixel überführt (hier dargestellt für einen vergrößerten Ausschnitt)

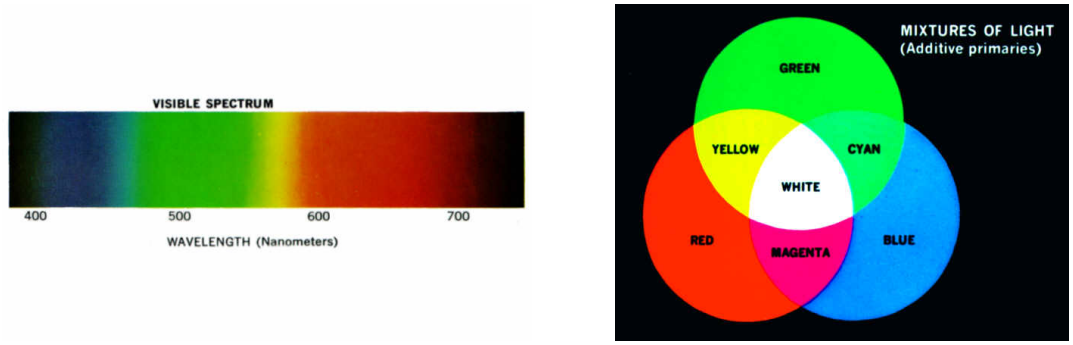
Abb. 2.1: Darstellung digitaler Schwarz/Weiß-Bilder

Wird ein Raster über ein Foto gelegt (siehe Abb. 2.1(b)) und entsprechen die Pixel den Zellen des Rasters, so repräsentiert jedes Pixel eine Menge von Punkten des Fotos. Diese Form der Darstellung wird auch *Rasterbild* genannt. Das Raster ist in der Regel orthogonal und die Kanten haben äquidistanten Abstand.

Es ist nicht ganz klar, wie m, n und b zu wählen sind, um ein Bild möglichst detailgetreu in ein Rasterbild umzuwandeln. Durch die Diskretisierung gehen zwangsläufig Bildinformationen verloren. Ein Maß für die Detailgenauigkeit ist die sog. *Auflösung* (engl. *resolution*). Sie misst die Anzahl der Pixel, die horizontal und vertikal auf einer bestimmten Fläche Platz finden und wird in der Regel in DPI (dots per inch) angegeben. Eine höhere Auflösung entspricht einer feineren Einteilung des Rasters. Die Anzahl der Grauwerte wird aus technischen Gründen meist als eine Zweierpotenz gewählt, d.h. monochrome Rasterbilder enthalten 2, 4, 8, 16, 32, 64, 128 oder 256 verschiedene Grauwerte.

2.3 Digitale Farbbilder

Die Repräsentation digitaler Farbbilder unterscheidet sich grundsätzlich nicht von der monochromer digitaler Bilder. Der einzige Unterschied besteht in der Ersetzung der Grauwerte durch Farben.



(a) Elektromagnetisches Energie-Spektrum (b) Mischung der Primärfarben

Abb. 2.2: Farbtafeln aus [GW92]

Das Licht, das auf die Netzhaut des menschlichen Auges trifft, setzt sich aus elektromagnetischen Wellen zusammen, die einem kontinuierlichen Spektrum von Farben entsprechen. Das sichtbare Spektrum lässt sich in sechs breite Bereiche einteilen: Violett, Blau, Grün, Gelb, Orange und Rot. Die Farbbereiche enden nicht abrupt, sondern gehen ineinander über (siehe Abb. 2.2(a)). Die Farbe wird durch die jeweilige Wellenlänge bestimmt.

Das menschliche Auge besitzt drei verschiedene Typen von Rezeptoren auf der Netzhaut, die jeweils auf die Spektren Rot, Grün und Blau reagieren. Alle anderen Farben, die der Mensch sieht, ergeben sich aus einer Mischung dieser drei Spektren. Die CIE (Commission Internationale de l'Eclairage) hat 1931 aus Gründen der Standardisierung die Werte der Wellenlängen festgelegt, die als „reines" Rot, Grün und Blau angesehen werden sollen (Rot = 700 nm, Grün = 546.1 nm, Blau = 435.8 nm). Diese Farben werden auch *Primärfarben* genannt. Durch Mischung der Primärfarben ergeben sich die sog. *Sekundärfarben* Magenta (Rot und Blau), Cyan (Grün und Blau) und Gelb (Rot und Grün) (siehe Abb. 2.2(b)). Werden alle Primärfarben oder eine Sekundärfarbe mit ihrer entgegengesetzten Primärfarbe gemischt, so ergibt sich bei der Addition von Wellenlängen des Lichtes die Farbe Weiß. Bei der Mischung von Farbpigmenten hingegen sind die Primär- und Sekundärfarben vertauscht und die Mischung aller Farben ergibt Schwarz.

Die Menge an Rot, Grün und Blau, die benötigt wird, um irgendeine andere Farbe durch Mischung zu generieren, lässt sich durch die drei sog. *tristimulus* Werte X, Y und Z charakterisieren. Eine Farbe wird durch ihre *trichromatischen Koeffizienten* spezifiziert, die wie folgt definiert sind:

$$x = \frac{X}{X+Y+Z} \tag{2.1}$$

$$y = \frac{Y}{X+Y+Z} \quad \text{und} \tag{2.2}$$

$$z = \frac{Z}{X+Y+Z} \quad . \tag{2.3}$$

Die tristimulus Werte, die benötigt werden, um eine Farbe zu generieren, können Kurven oder Tabellen entnommen werden (z. B. der Normfarbtafel der CIE, für eine Abb. siehe [Ran89], [GW92] oder [Tsc02]).

Für die standardisierte Spezifikation von Farben haben sich — je nach Anwendung — unterschiedliche Farbmodelle entwickelt. Ein Farbmodell gibt ein dreidimensionales Koordinatensystem an und einen dazugehörigen Unterraum, in dem jede Farbe durch einen einzelnen Punkt repräsentiert wird. Im technischen Bereich haben sich drei Farbmodelle etabliert. Das RGB-Modell (red, green, blue) wird von Farbmonitoren und Farbvideokameras verwendet. Farbdrucker benutzen das CMY-Modell (cyan, magenta, yellow). Farbfernseher greifen auf das YIQ-Modell (luminance, inphase, quadrature) zurück. Für die Bearbeitung von Bildern durch den Menschen eignet sich eher das HSI-Modell (hue, saturation, intensity). Die Modelle können ineinander umgerechnet werden (für nähere Informationen siehe [GW92]). Für die Analyse von Luftaufnahmen und die Interpretation durch den Menschen eignen sich das RGB-Modell und das HSI-Modell, die im Folgenden beschrieben werden.

RGB-Modell Beim RGB-Modell wird jede Farbe durch die Anteile der drei Primärfarben Rot, Grün und Blau bestimmt. Der Teilbereich des von dem Modell verwendeten kartesischen Koordinatensystems ist in Abb. 2.3 dargestellt. Die Farbwerte sind so skaliert, dass sich ein Einheitswürfel ergibt und alle Farbwerte im Bereich $[0, 1]$ liegen. Die Farbe Schwarz liegt im Ursprung, die Farbe Weiß ist am weitesten davon entfernt. Drei Ecken bilden die Primärfarben Rot, Grün und Blau, die anderen Ecken die Sekundärfarben Cyan, Magenta und Gelb. Die Skala der Grauwerte befindet sich auf der Diagonalen von Schwarz nach Weiß.

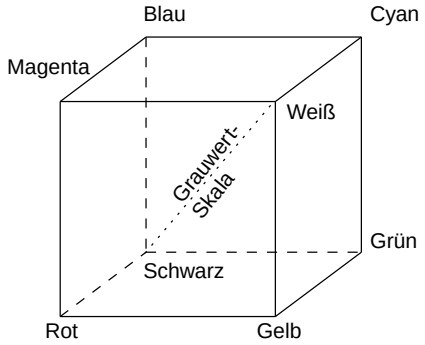


Abb. 2.3: RGB-Farbwürfel

HSI-Modell Das HSI-Modell kommt der Beschreibung von Farben durch den Menschen am nächsten. Anstatt Farben durch die Mischung aus Primärfarben zu beschreiben, setzt sich eine Farbe im HSI-Modell aus den Größen *Farbwert* (engl. *hue*), *Sättigung* (engl. *saturation*) und *Helligkeit* (engl. *intensity*) zusammen. Der Farbwert bestimmt die dominante Wellenlänge in einer Mischung aus elektromagnetischen Wellen. Er entspricht dem, was Menschen meinen, wenn sie ein Objekt „rot“, „gelb“ oder „orange“ nennen. Die Sättigung bezieht sich auf die relative Reinheit der Farbe und bestimmt den Anteil der Mischung von Weiß mit dem Farbwert. So haben die Primärfarben eine volle Sättigung, während die Mischung von Rot und Weiß z. B. ein Pink ergibt. Die Helligkeit ist eine sehr subjektive Größe, die sich einer Beschreibung entzieht. Eine Veränderung der Helligkeit hat keine Auswirkungen auf die Farben in einem Bild.

Die Anzahl der Farben in einem Bild bestimmt dessen Farbtiefe. Die Farbtiefe wird in Bits angegeben und entspricht damit wie bei den Grauwerten der Darstellung durch Zweierpotenzen. Üblich sind Farbtiefen von 8 bit (256 Farben), 16 bit (65536 Farben) und 24 bit (16.777.216 Farben). Bilder mit einer Farbtiefe von 24 bit werden auch True-Color-Bilder genannt.

2.4 Der Prozess der digitalen Bildverarbeitung

Der Prozess der Verarbeitung eines digitalen Bildes lässt sich durch folgende Schritte charakterisieren [GW92]: *Bilderwerb* (engl. *image acquisition*), *Vorverarbeitung* (engl. *preprocessing*), *Segmentierung* (engl. *segmentation*), *Merkmalsdetektion* (engl. *feature selection*) und *Erkennung/Interpretation* (engl. *recognition, interpretation*).

Bilderwerb Der Erwerb eines digitalen Bildes kann durch Geräte wie Kamera oder Scanner erfolgen, die analoge Informationen aus der Umgebung oder von Vorlagen (z. B. Fotos) mit Hilfe eines Analog/Digital-Wandlers in digitale Informationen umwandeln. Dazu bedarf es eines Sensors, der ein elektrisches Signal proportional zur Energie erzeugt, die auf den Sensor trifft. Im Falle eines Scanners wird z. B. durch einen Fotodetektor die Intensität gemessen, mit der ein Lichtstrahl (z. B. ein Laserstrahl) von der Oberfläche des zu scannenden Objekts reflektiert wird.

Vorverarbeitung Durch eine Aufbereitung der Bildinformationen (z. B. Rauschunterdrückung, Farbkorrekturen oder Änderung des Kontrastes) sollen die Chancen erhöht werden, das Bild durch andere Schritte erfolgreich weiterverarbeiten zu können. Die Aufbereitung von Bildern für die Interpretation durch den Menschen ist mit diesem Schritt oft bereits abgeschlossen.

Segmentierung Das Bild wird in Bereiche und/oder Objekte eingeteilt. Es besteht ein Unterschied zwischen einer Einteilung, die auf dem Umriss (und damit der Form) der einzelnen Bildteile beruht und einer, welche die gesamte Region (also die Pixel) repräsentiert. Für die Erkennung von Objekten eignet sich der Umriss besser, für die Erkennung von Texturen die Darstellung durch Pixel. Z. B. entspricht der Schritt der Segmentierung bei der automatischen Schrifterkennung der Einteilung in Regionen mit einzelnen Buchstaben.

Merkmalsdetektion In diesem Schritt werden wesentliche Merkmale der Objekte herausgearbeitet, die für die spätere Unterscheidung und Erkennung der Objekte wichtig sind. Im Falle der Schrifterkennung wären das z. B. Freiräume in den Buchstaben oder Rundungen.

Erkennung/Interpretation Die Erkennung von Objekten entspricht einer Zuweisung von Namen (also z. B. die Erkennung des Buchstabens „c“). Die Interpretation geht noch weiter und versucht die Bedeutung einer Gruppe von erkannten Objekten zu erschließen. Im Falle der Schrifterkennung etwa, ob es sich um eine Postleitzahl handelt.

Was den Schritt des Bilderwerbs angeht, so werden die Erosionsgebiete von Kameras aufgenommen, die an Drachen, Ballons oder Kleinflugzeugen montiert sind [Zer01]. Zum Einsatz kommen sowohl herkömmliche Kameras, deren Negative oder Fotos eingescannt werden, als auch Digitalkameras.

Wird die Clusteranalyse als ein Verfahren für die Analyse digitaler Bilder angesehen, so entspricht sie entweder dem Schritt der Vorverarbeitung oder dem der Segmentierung. Die Unterscheidung hängt mit der Art der Repräsentation von Bildbereichen zusammen. Für einen Cluster können die Instanzen (z. B. die Pixel) bestimmt werden, die in diesen Cluster fallen. In diesem Sinn wird das Bild — basierend auf den Pixeln — in Bereiche eingeteilt (segmentiert). Der Umriss der Bereiche ist jedoch nicht bekannt. Es ist nicht einmal klar, ob sich zusammenhängende Bereiche überhaupt ergeben. Durch eine vorhergehende Clusteranalyse kann der Umriss in nachfolgenden Schritten jedoch unter Umständen leichter bestimmt werden. In diesem Sinn ist die Clusteranalyse ein Vorverarbeitungsschritt.

Für die Erkennung der Objekte (also z. B. einer Benennung der Cluster) eignet sich die Clusteranalyse nicht, da es sich um nicht überwachtes Lernen handelt. Ist das Bild einmal vereinfacht und in Klassen eingeteilt, so lassen sich jedoch überwachte Lernverfahren einsetzen, um den Clustern einen Sinn zu geben.

2.5 Anwendung der Clusteranalyse

Die Luftaufnahmen der Erosionsgebiete liegen in True-Color-Bildformaten vor, die das RGB-Modell zur Darstellung von Farben verwenden. Jedem Pixel des Bildes ist ein RGB-Vektor zugeordnet. Da die Repräsentation von Daten für die erfolgreiche Anwendung von Lernverfahren so wichtig ist (siehe Abschnitt 1.1.3), muss geklärt werden, wie die primären Bildinformationen (Position und Farbe) in Merkmalsvektoren kodiert werden.

Zu unterscheiden ist zwischen zwei möglichen Aufgabenstellungen:

- Es soll ein System entwickelt werden, welches eine Menge von Luftaufnahmen in Klassen einteilt. Die Klassen enthalten Luftaufnahmen von Gebieten mit ähnlichem Erosionsgrad. Die Aufgabe ist in diesem Fall globaler Art: unterscheide Bilder, die erodierte Gebiete zeigen, von solchen, auf denen keine oder kaum Erosion vorliegt. Ein Merkmalsvektor kodiert in diesem Fall die Eigenschaften eines ganzen Bildes.
- Es soll ein System entwickelt werden, welches eine einzelne Luftaufnahme in Gebiete mit unterschiedlichem Erosionsgrad einteilt. Gesucht ist eine Segmentierung des Bildes im Sinne von Abschnitt 2.4. Die Aufgabenstellung schließt die Verwendung mehrerer Luftaufnahmen desselben Gebietes zu unterschiedlichen Zeitpunkten nicht aus. Ein Merkmalsvektor kodiert Eigenschaften einzelner lokaler Bildelemente.

Die Arbeit folgt dem Ziel der Segmentierung eines Bildes in unterschiedliche Regionen. Laut Gonzalez [GW92] stellt die Farbe ein wichtiges Merkmal für die Erkennung von Objekten dar. Die Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02] demonstrieren, dass die Kodierung der Farben eines Bildes in Merkmalsvektoren für die Erkennung von Erosion ausreicht. Die Position der Pixel wird im Folgenden daher nicht betrachtet.

Für numerische Clusteringverfahren bietet sich eine unmittelbare Überführung der Farbrepräsentation des RGB- oder HSI-Modells in Merkmalsvektoren an. Um eine bessere Vergleichbarkeit mit den Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02] zu gewährleisten, wird das RGB-Modell verwendet.

Der Bereich der Primärfarben Rot, Grün und Blau wird auf Rechnern aus technischen Gründen durch Zweierpotenzen bestimmt. True-Color-Bilder umfassen 24 bit, so dass für die Primärfarben je 8 bit (d.h. $2^8 = 256$ verschiedene Werte) zur Verfügung stehen. In der Praxis arbeiten numerische Clusteringverfahren meist auf Vektoren, deren Werte im Intervall $[0, 1]$

liegen. Der Bereich $0, \ldots, 255$ muss daher durch Multiplikation mit $\frac{1}{255}$ entsprechend skaliert werden. So liegt z. B. die Farbe $(34, 16, 248)$ im RGB-Einheitswürfel (siehe Abb. 2.3) an der Position $(0.1328125, 0.0625, 0.96785)$.

Eine Voraussetzung für die Anwendung der in Kapitel 1 beschriebenen Clusteringverfahren ist, dass die einzelnen Attribute der Merkmalsvektoren unabhängig voneinander sind. Da es sich bei den Farben Rot, Grün und Blau um Primärfarben handelt, die selbst nicht aus Mischung entstehen, ist die Unabhängigkeit gewährleistet.

Kapitel 3

Algorithmische Geometrie

Ein zentrales Ziel dieser Arbeit ist die Verbesserung der Laufzeit des in Abschnitt 1.2.3 vorgestellten AML-Algorithmus. Der Initialschritt des Algorithmus geht auf das sog. *Farthest-Pair-Problem* (siehe Abschnitt 4.3) zurück, welches aus dem Bereich der algorithmischen Geometrie stammt. Dieses Kapitel umreißt kurz das Gebiet der algorithmischen Geometrie und widmet sich danach der Berechnung der konvexen Hülle, die zur Lösung des Farthest-Pair-Problems verwendet werden kann (siehe Abschnitt 4.4).

3.1 Einführung

Das Forschungsgebiet der algorithmischen Geometrie ist Ende der 70er Jahre aus dem Bereich der allgemeinen Analyse von Algorithmen hervorgegangen. Inzwischen sind eigene Journale und Konferenzen entstanden und das Gebiet verfügt über eine aktive Gemeinde von Forschern [dBvKOS00].

Zu den zu lösenden Problemen gehört z. B. die Berechnung der Schnittpunkte von Liniensegmenten, die Zerlegung von Polygonen in Dreiecke, die Erforschung spezieller Datenstrukturen für geometrische Probleme wie z. B. Segment-Bäume oder Range-Bäume und die Lokalisierung der eigenen Position anhand von geographischen Karten. Die Berechnung von sog. Voronoi-Diagrammen und der konvexen Hülle einer Menge von Punkten gehören zu den Grundproblemen der algorithmischen Geometrie.

Die algorithmische Geometrie hat viele Anwendungsgebiete, wie z. B. Computergrafik, Robotik, geographische Informationssysteme, Datenbanken, CAD/CAM, Modellierung von Mo-

lekülen oder Mustererkennung.

Wie gezeigt wird, gibt es auch im Bereich der Clusteranalyse Probleme, die dem Gebiet der algorithmischen Geometrie zugeordnet werden können. So erfolgt z. B. nach Anwendung eines Clusteringverfahrens die Klassifikation neuer Objekte häufig, indem diese dem Cluster zugeordnet werden, dessen Zentrum den kürzesten Abstand zu dem Vektor aufweist, der das neue Objekt repräsentiert. Es geht also darum, den kürzesten Abstand zwischen einem Punkt und einer Punktmenge effizient zu bestimmen — ein geometrisches Problem.

Bevor sich Kapitel 4 ausschließlich dem Ziel dieser Arbeit widmet, einen effizienten Algorithmus für die Lösung des sog. *Farthest-Pair-Problems* zu entwickeln, sollen in diesem Kapitel zunächst verschiedene Ansätze zur Berechnung der konvexen Hülle einer Punktmenge vorgestellt werden. Zum einen kann die konvexe Hülle wie später in Abschnitt 4.4 gezeigt zur Lösung des Farthest-Pair-Problems genutzt werden, zum anderen verdeutlicht sie die Komplexität geometrischer Probleme, insbesondere in höheren Dimensionen.

3.2 Was ist die konvexe Hülle?

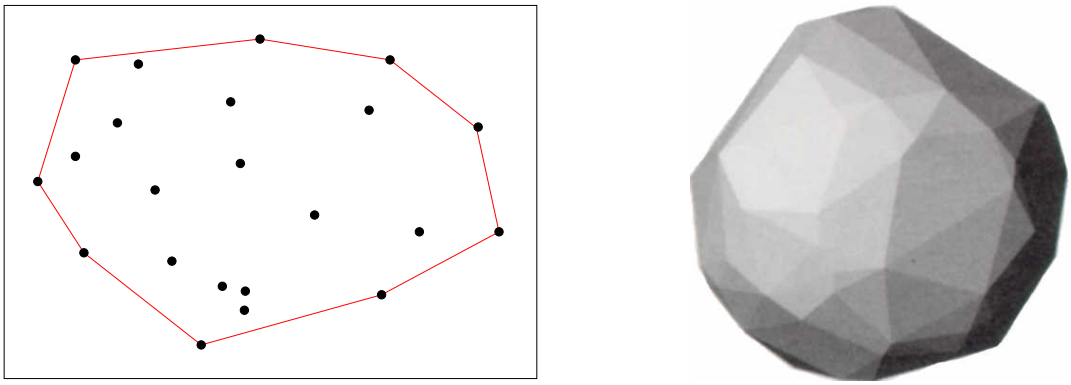
Vor einer Definition der *konvexen Hülle* wird der Begriff der *konvexen Menge* benötigt. Eine Teilmenge $L \subseteq \mathbb{R}^d$ ist eine *konvexe Menge*, wenn für jedes Paar von Punkten $p, q \in L$ das Liniensegment $\overline{pq}$ vollständig in L enthalten ist.

Definition 3.1 (konvexe Hülle) *Sei $L \subseteq \mathbb{R}^d$. Die* konvexe Hülle $CH(L)$ *von L ist die kleinste konvexe Menge, die L enthält.*

Zum Verständnis der konvexen Hülle einer endlichen Menge von Punkten seien im Folgenden einige weitere Begriffe eingeführt.

Ein *Polygon* P im $\mathbb{R}^2$ ist eine endliche Menge gerader Liniensegmente, so dass jeder Endpunkt zu genau zwei Segmenten gehört und keine Teilmenge von Segmenten die gleiche Eigenschaft hat. Die Segmente sind die *Kanten* des Polygons und die Endpunkte die *Ecken*.

Ein Polygon ist *einfach*, wenn alle Paare nicht aufeinander folgender Kanten keinen Punkt miteinander gemeinsam haben. Ein einfaches Polygon teilt die Ebene in zwei nicht zusammenhängende Regionen ein, eine (begrenzte) *innere* und eine (unbegrenzte) *äußere*, die durch das Polygon getrennt werden (*Jordan curve theorem*).



(a) Polygon in $\mathbb{R}^2$ (b) Polytop in $\mathbb{R}^3$

Abb. 3.1: Beispiele für die konvexe Hülle einer endlichen Punktmenge

Ein einfaches Polygon P ist *konvex*, wenn seine Begrenzung und die innere Region eine konvexe Menge sind.

> **Theorem 3.1** *Die konvexe Hülle einer endlichen Menge S von Punkten in der Ebene ist ein einfaches Polygon. Außerdem muss jede Ecke der Hülle ein Punkt aus S sein.*

Ein intuitiver Beweis findet sich in [Che96], ein formaler in [MS71]. Die intuitive Vorstellung der konvexen Hülle einer endlichen Menge von Punkten im $\mathbb{R}^2$ entspricht einem Gummiband, das vollständig um in ein Brett eingeschlagene Nägel gespannt ist (siehe Abb. 3.1(a)).

In [PS85] findet sich ein Theorem von McMullen und Shephard [MS71] für endliche Punktmengen in höheren Dimensionen:

> **Theorem 3.2** *Die konvexe Hülle einer endlichen Menge von Punkten im $\mathbb{R}^d$ ist ein konvexes Polytop. Umgekehrt ist ein konvexes Polytop die konvexe Hülle einer endlichen Menge von Punkten.*

Ein konvexes Polytop (siehe Abb. 3.1(b)) lässt sich durch seine Flächen beschreiben. Jede dieser Flächen ist eine konvexe Menge (und wiederum ein niedrigdimensionales konvexes Polytop). So besteht ein dreidimensionales Polytop z. B. aus zweidimensionalen Flächen (konvexe

Polygone), eindimensionalen (Kanten) und 0-dimensionalen (Ecken). Entsprechend lässt sich ein Polygon P über seine Ecken $p_1, p_2, \ldots, p_n$ beschreiben.

Die konvexe Hülle im dreidimensionalen Raum kann man sich etwa als eine stramm gezogene Plastikfolie vorstellen, welche eine Punktmenge vollständig einhüllt. Eine anschauliche Beschreibung der konvexen Hülle in höherdimensionalen Räumen ist bedingt durch die Grenzen der menschlichen Vorstellung jedoch schwierig [Ruc84].

Der nächste Abschnitt beschäftigt sich mit der Frage, welche Algorithmen es gibt, um die konvexe Hülle einer Menge von n Punkten effizient im zweidimensionalen und höherdimensionalen Raum zu bestimmen.

3.3 Algorithmen

Das Problem der Berechnung der konvexen Hülle wirkt auf den ersten Blick bereits in der Ebene schwierig. So ergibt eine erste Annäherung an das Problem durch den Algorithmus INTERIORPOINTS eine Zeitkomplexität von $O(n^4)$ und eine Verbesserung durch den Algorithmus EXTREMEEDGES eine Zeitkomplexität von $O(n^3)$ [O'R00].

Im Laufe der Zeit wurden jedoch immer schnellere und einfacher zu implementierende Algorithmen entwickelt, von denen einige bekannte in diesem Abschnitt vorgestellt werden. Nicht alle Algorithmen lassen sich auf höhere Dimensionen erweitern oder weisen eine vergleichbar gute Zeitkomplexität wie in der Ebene auf.

3.3.1 Gift Wrapping

Dieser Algorithmus ist auch unter dem Namen *Jarvis's March* oder *Package Wrapping* bekannt [CLRS01]. Er wurde ursprünglich 1970 von Chand und Kapur [CK70] entwickelt und stellte über Jahre den wichtigsten Algorithmus für die Berechnung der konvexen Hülle in höheren Dimensionen dar [O'R00]. Der Name rührt daher, dass der Algorithmus — in Anlehnung an eine intuitive Vorstellung der konvexen Hülle (siehe Abschnitt 3.2) — das Einpacken eines Geschenkes mit Papier simuliert.

Im zweidimensionalen Fall startet der Algorithmus am untersten Punkt der Punktmenge, der in jedem Fall zur konvexen Hülle gehört. Dann wird das „Papier“ nach rechts gezogen, stramm gehalten und so lange nach oben bewegt, bis ein weiterer Punkt berührt wird. So fährt

das Verfahren fort, bis es wieder bei dem Punkt angekommen ist, mit dem es begonnen hat. Alle besuchten Punkte bilden die konvexe Hülle.

Der Algorithmus macht sich die Eigenschaft zunutze, dass jeder Punkt der konvexen Hülle den jeweils kleinsten Polarwinkel zu seinem (bereits ermittelten) Vorgänger auf der Hülle besitzt. So müssen für jeden Punkt auf der Hülle n Winkel berechnet werden. Das führt zu einer Laufzeit von $O(nh)$, wenn h die Anzahl der Punkte der konvexen Hülle ist, im *worst case* also zu $O(n^2)$ [O'R00, CLRS01].

Der Algorithmus lässt sich auf höhere Dimensionen erweitern und hat zumindest in der dritten Dimension ebenfalls eine Laufzeit von $O(n^2)$ [O'R00].

3.3.2 Graham's Scan

Graham [Gra72] entwickelte 1972 den ersten Algorithmus für die Ebene mit einer Zeitkomplexität von $O(n \log n)$.

Der Algorithmus wählt zuerst den Punkt mit der kleinsten y-Koordinate aus (den am weitesten links liegenden, falls es mehrere Punkte gibt) und sortiert alle anderen Punkte anhand ihres Winkels zu diesem ersten Punkt. Danach besucht der Algorithmus nacheinander jeden Punkt in sortierter Reihenfolge und überprüft, ob die letzten beiden der Punkte, die er während des Durchlaufs auf einem Stapel ablegt, eine Kurve nach rechts bilden. Falls dies so ist, werden so lange Punkte vom Stapel entfernt, bis die letzte Bedingung nicht mehr zutrifft und der Algorithmus fährt mit dem nächsten Punkt fort, bis alle Punkte abgearbeitet sind.

Die Suche nach dem Punkt mit der kleinsten y-Koordinate benötigt $O(n)$ Schritte. Die Sortierung kann in $O(n \log n)$ Schritten bewältigt werden. Die Schleife zum Abarbeiten der sortierten Punkte wird $O(n)$ mal durchlaufen. Jede Pop-Operation entfernt einen Punkt dauerhaft, so dass insgesamt nur n Punkte vom Stapel entfernt werden können, d.h. die Schleife benötigt höchstens $2n$ Schritte. Damit ergibt sich eine Gesamtlaufzeit von $O(n \log n)$ [O'R00, CLRS01].

Der Algorithmus lässt sich aufgrund der Sortierung nach Winkeln nicht auf höherdimensionale Fälle erweitern.

3.3.3 Divide and Conquer

Der von Preparata und Hong [PH77] 1977 beschriebene Divide-and-Conquer-Algorithmus ist der einzige bekannte Algorithmus, der sowohl im zweidimensionalen als auch im dreidimensionalen Fall eine Laufzeit von $O(n \log n)$ im *worst case* erreicht [O'R00].

Zunächst werden die Punkte der Menge nach x-Koordinate sortiert und in eine linke Hälfte mit $\lceil n/2 \rceil$ Punkten und eine rechte Hälfte mit $\lfloor n/2 \rfloor$ Punkten geteilt. Im Rekursionsschritt wird für jede der beiden Hälften separat die konvexe Hülle berechnet. Zum Schluss werden die berechneten konvexen Hüllen in einem komplizierten Schritt zu einer einzigen zusammengefügt.

Die Punkte lassen sich in $O(n \log n)$ Schritten anhand ihrer x-Koordinate sortieren. Der sog. *Merge-Schritt* benötigt $O(n)$ Operationen, um die konvexen Hüllen zusammenzufügen. Gemäß der Rekursionsgleichung $T(n) = 2T(n/2) + O(n)$ ergibt sich eine Zeitkomplexität von $O(n \log n)$. Auch im dreidimensionalen Raum benötigt der Merge-Schritt $O(n)$ Operationen, was wiederum zu einer Gesamtlaufzeit von $O(n \log n)$ führt.

Trotz der guten Laufzeiteigenschaften im *worst case* werden aufgrund der schwierigen Implementation des Algorithmus in der Praxis oftmals asymptotisch langsamere Algorithmen eingesetzt oder solche, die nur eine erwartete Laufzeit von $O(n \log n)$ haben [O'R00].

3.3.4 Inkrementeller Algorithmus

Der inkrementelle Algorithmus konstruiert die konvexe Hülle Punkt für Punkt, wie sein Name bereits andeutet. Dazu werden alle Punkte der Menge sortiert, so dass sich die Folge $\langle p_0, p_1, \ldots, p_n \rangle$ ergibt. Im i-ten Schritt wird die bis dahin bereits berechnete konvexe Hülle der am weitesten links liegenden $i - 1$ Punkte um den aktuellen i-ten Punkt erweitert und von links her entsprechend so angepasst, dass sich wieder eine konvexe Hülle ergibt.

Der Algorithmus benötigt im *worst case* (d.h. wenn alle Punkte zur konvexen Hülle gehören) in jedem Schritt $O(n)$ Operationen, um die konvexe Hülle zu aktualisieren. Das ergibt eine Gesamtlaufzeit von $O(n^2)$. Durch die Sortierung der Punkte lässt sich der Schritt allerdings noch effizienter implementieren, was eine Gesamtlaufzeit von $O(n \log n)$ im *worst case* ergibt [O'R00].

Der Algorithmus lässt sich auf höhere Dimensionen erweitern. In der einfachen Version erreicht er in der dritten Dimension eine Gesamtlaufzeit von $O(n^2)$ im *worst case*, wobei der

```
function QuickHull(a, b, S)
    if S = ∅ then return ()
    else
        c ← Index des Punktes mit max. Abstand von ab.
        A ← Punkte rechts von (a, c).
        B ← Punkte rechts von (c, b).
        return QuickHull(a, c, A) + (c) + QuickHull(c, b, B)
```

Abb. 3.2: Pseudocode für den Quickhull-Algorithmus (aus [O'R00])

Algorithmus im Durchschnitt oft schneller ist und sich daher für die Praxis durchaus eignet. Eine randomisierte Version von Clarkson und Shor [CS89] erreicht eine erwartete Laufzeit von $O(n \log n)$, d.h. diese Laufzeit wird mit hoher Wahrscheinlichkeit für jede Eingabe erreicht [O'R00].

3.3.5 Quickhull

In den späten 70er Jahren schlugen mehrere Forscher unabhängig voneinander einen Algorithmus vor, der 1985 von Preparata und Shamos [PS85] als *QuickHull* bezeichnet wurde.

Die Grundidee des Algorithmus beruht auf der Beobachtung, dass für die meisten Punktmengen einige Punkte als „sicher innerhalb der konvexen Hülle liegend" bestimmt werden können. Diese spielen für die weitere Betrachtung keine Rolle mehr. Eine Implementation von Quickhull kommt im empirischen Teil dieser Diplomarbeit zum Einsatz (siehe Abschnitt 5.2). Aus diesem Grund soll Quickhull im Folgenden etwas ausführlicher beschrieben werden als die bisher vorgestellten Algorithmen.

Im ersten Schritt werden zwei unterschiedliche Extrempunkte der Menge bestimmt, z. B. der am weitesten links gelegene höchste Punkt y und der am weitesten rechts gelegene niedrigste Punkt x. Diese beiden Punkte sind in jedem Fall Punkte der konvexen Hülle. Die Punktmenge wird durch die Strecke xy in einen oberen Abschnitt S_1 und einen unteren Abschnitt S_2 geteilt, für welche die konvexe Hülle separat bestimmt werden kann.

Die konvexe Hülle für jeweils einen der Abschnitte wird nun bestimmt, indem zu zwei extremen Punkten (a, b) ein dritter Punkt c rechts von ab gesucht wird, der zu beiden Punkten maximalen Abstand hat und damit ebenfalls extrem ist. Dieser Punkt gehört ebenfalls zur konvexen Hülle. Die Punkte innerhalb des Dreiecks $\triangle abc$ können aus der Betrachtung aus-

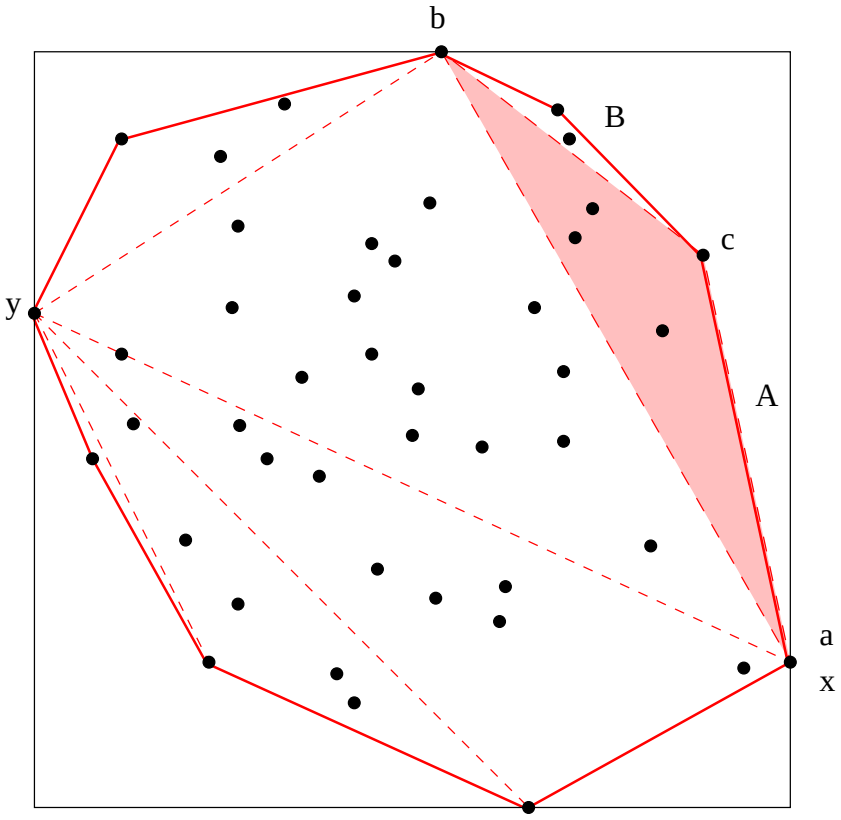


Abb. 3.3: QuickHull entfernt die Punkte in $\triangle abc$ und arbeitet weiter auf A ($A = \emptyset$) und B ($|B| = 2$).

geschlossen werden, da sie innerhalb der konvexen Hülle liegen. Interessant sind dementsprechend nur noch die beiden Punktmengen A rechts der Strecke ac und B rechts der Strecke cb. Für diese kann die konvexe Hülle rekursiv mit dem soeben beschriebenen Verfahren erneut bestimmt werden. Der Pseudocode ist in Abb. 3.2 dargestellt, wobei der Algorithmus eine Liste mit Punkten zurückgibt und '$+$' die Konkatenation zweier Listen bezeichnet. Nachdem x und y im ersten Schritt bestimmt wurden, ergibt sich die gesamte konvexe Hülle also aus $(x) + QuickHull(x, y, S_1) + (y) + QuickHull(y, x, S_2)$.

Die Abb. 3.3 zeigt einen Rekursionsschritt des Algorithmus, wobei die bis dahin generierten Dreiecke $\triangle abc$ mit gestrichelten Linien eingezeichnet sind.

Das Auffinden der beiden Extrempunkte im ersten Schritt und die Aufteilung in zwei Punktemengen S_1 und S_2 benötigt $O(n)$ Schritte. Die Berechnung des extremen Punktes c benötigt für eine Punktmenge S mit $n = |S|$ in einem Rekursionsschritt n Schritte, d.h. die Kosten der rekursiven Aufrufe hängen von der Größe der Menge S ab, also von A und B. Sei $|A| = \alpha$ und $|B| = \beta$ mit $\alpha + \beta \leq n - 1 = O(n)$. Wird der Algorithmus mit $n = |S|$ aufgerufen, so lässt sich

die Gesamtlaufzeit durch die Rekursionsgleichung $T(n) = O(n)+T(\alpha)+T(\beta)$ ausdrücken. Im besten Fall sind alle Punkte gleichmäßig und zufällig in der Ebene verteilt, so dass die Mengen A und B in jedem Rekursionsschritt gleich groß sind, d.h. $\alpha = \beta = n/2$. Damit ergibt sich die Rekursionsgleichung $T(n) = 2T(n/2) + O(n)$ mit der aus Abschnitt 3.3.3 bekannten Lösung $O(n \log n)$ [O'R00].

Im *worst case* sind die Größen der Mengen A und B so ungleichmäßig wie möglich, also z. B. $\alpha = 0$ und $\beta = n - 1$. Damit ergibt sich die Rekursionsgleichung $T(n) = T(n - 1) + O(n) = T(n - 1) + cn$. Wiederholtes Einsetzen führt insgesamt zu einer oberen Schranke von $O(n^2)$ [O'R00].

Der Algorithmus lässt sich auf höhere Dimensionen erweitern. Wie im Abschnitt 3.4 noch dargestellt wird, gibt es eine Grenze für die Optimierungsmöglichkeiten von Algorithmen, welche die konvexe Hülle in höheren Dimensionen berechnen.

3.4 Komplexität

Betrachtet sei zunächst eine untere Schranke für alle Algorithmen, welche die konvexe Hülle berechnen. Eine solche Schranke lässt sich durch Reduktion des Problems der Sortierung einer ungeordneten Liste von Zahlen auf das Problem der Bestimmung der konvexen Hülle finden. Eine allgemein bekannte untere Schranke für das Sortierproblem ist $\Omega(n \log n)$ [Güt92]. Falls es möglich ist, die konvexe Hülle zur Sortierung von Zahlen zu verwenden, so muss für die Algorithmen zur Bestimmung der konvexen Hülle die gleiche untere Schranke $\Omega(n \log n)$ gelten. Tatsächlich hat Shamos 1978 einen solchen Sortieralgorithmus, der auf der konvexen Hülle basiert, entwickelt [Sha87].

Für höhere Dimensionen hat Klee 1980 bewiesen, dass die konvexe Hülle bestehend aus n Punkten in d Dimensionen $\Omega(n^{\lfloor d/2 \rfloor})$ Flächen haben kann [Kle80]. Da alle Algorithmen mit den Flächen bzw. Hyperflächen der konvexen Hülle arbeiten und diese auch als Ergebnis ausgeben, ist bereits für $d = 4$ kein Algorithmus mit einer Gesamtlaufzeit von $O(n \log n)$ mehr möglich. Die bisher entwickelten Algorithmen erreichen im *worst case* eine Laufzeit von $O(n \log n + n^{\lfloor d/2 \rfloor})$ [O'R00].

Kapitel 4

Das Farthest-Pair-Problem

Zu Beginn des Kapitels wird der Schwerpunkt dieser Arbeit näher erläutert und eine formale Definition des *Farthest-Pair-Problems* gegeben. Danach werden zwei Lösungsmöglichkeiten für das Problem vorgestellt. Die erste basiert auf den in Kapitel 3 beschriebenen Algorithmen zur Berechnung der konvexen Hülle. Die zweite basiert auf einem *Pruning-Algorithmus*.

4.1 Problemstellung und Zielsetzung

Wie die Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02] zeigen, erzeugen die Clusteringverfahren ML und AML im Hinblick auf die Erosionsproblematik bessere Clusterungen als nicht überwacht lernende neuronale Netze. Der empirische Teil dieser Arbeit (Kapitel 5) kommt zu ähnlichen Ergebnissen auch im Vergleich mit anderen Clusteringverfahren. Mit der $GMMD$ (Formel 1.11) steht ein zuverlässiges Kriterium zur Verfügung, um die Güte von Clusterungen im Bereich der Erosionsproblematik bewerten zu können.

In Bezug auf die Güte der Clusterungen und die Interpretierbarkeit der Ergebnisse erweist sich der AML-Algorithmus daher als geeignet für die Erosionserkennung auf Luftaufnahmen. Die erzeugten Clusterungen erfüllen wesentliche Forderungen, die sich im Zusammenhang mit der Erosionsproblematik ergeben (hohe Trennschärfe, Bildung von zusammenhängenden Farbflächen und Erkennung seltener Elemente).

In der Praxis spielt neben der Güte der erzeugten Clusterungen allerdings auch die Laufzeit und der Speicherplatzverbrauch der eingesetzten Verfahren eine herausragende Rolle.

Witten [WF01] etwa macht darauf aufmerksam, wie wichtig eine Lernzeit ist, die sich li-

near zur Anzahl der Trainingsbeispiele verhält. Kohonen [Koh01] beschreibt, wie 6.840.568 englische, in elektronischer Form vorliegende, Abstracts von Patenten mit Hilfe einer selbstorganisierenden Karte (WEBSOM) zu klassifizieren sind. Für diese Aufgabe benötigte eine SGI O2000 mit sechs Prozessoren ungefähr 6 Wochen bei einem maximalen Speicherplatzverbrauch von 800 MB. Versuche mit der Anwendung PICANA ergeben für das Trainieren einer selbstorganisierenden Karte (hexagonale Topologie) mit 393.216 RGB-Instanzen bei 4000 Lernzyklen eine Laufzeit von über zwei Tagen [Zer01].

Die Implementation des ML-Algorithmus in PICANA ersetzt die speicherintensive Berechnung von Distanzmatrizen durch ein effizienteres Verfahren [Tsc02]. Auf weitere Überlegungen zum Speicherplatzverbrauch wird daher verzichtet.

Eine erste Analyse des ML-Algorithmus durch Zerbst [Zer01] ergab, dass das Verfahren im Vergleich mit selbstorganisierenden Karten auch bezüglich der Laufzeit besser abschneidet. Dies kann durch konkrete Zahlenbeispiele und empirische Ergebnisse belegt werden, die sich aus den in [Zer01] analysierten Bildern ergeben haben. Eine in der Informatik übliche Analyse, welche das asymptotische Laufzeitverhalten des Algorithmus beschreibt, ist bisher jedoch ausgeblieben.

Die Arbeit widmet sich daher in diesem Kapitel der Bestimmung des asymptotischen Laufzeitverhaltens des AML-Algorithmus (4.2) und der Frage, ob sich der Algorithmus bezüglich der Laufzeit verbessern lässt. Dadurch wird ein wesentlicher Beitrag zur möglichen Anwendung des Algorithmus auch auf andere, komplexere Problemgebiete als das der Erosionserkennung geleistet.

4.2 Laufzeitanalyse von AML

In diesem Abschnitt wird eine neue Laufzeitanalyse des AML-Algorithmus (siehe Abscnitt 1.2.3) durchgeführt, wie er in [Tsc02] beschrieben und in der Anwendung PICANA implementiert worden ist. Insbesondere die asymptotische Laufzeit des Algorithmus wird für die verschiedenen Fälle *best case*, *worst case* und *average case* näher untersucht.

Zu unterscheiden sind zwei verschiedene Schritte des Algorithmus. Im *Initialschritt* werden zwei Instanzen aus der Menge der unterschiedlichen Instanzen $U \subseteq S$ als Startwerte ermittelt, die maximalen Abstand zueinander haben. Dazu werden alle paarweisen Abstände zwischen den Instanzen berechnet. Unter einer Elementaroperation kann die Berechnung des Abstandes

zweier Instanzen verstanden werden, da sich die Dimension der Trainingsdaten nicht verändert. Zu berücksichtigen ist die Tatsache, dass aufgrund der Spiegelsymmetrie nur die Hälfte der Distanzmatrix ohne die Diagonalelemente für Abstandsberechnungen herangezogen werden muss. Die Anzahl $T_{init}(r)$ der Abstandsberechnungen im Initialschritt für $r = |U|$ unterschiedliche Instanzen ergibt somit [Zer01]:

$$T_{init}(r) = \frac{r(r-1)}{2} \quad . \tag{4.1}$$

Der *Iterationsschritt* des Algorithmus bestimmt in einer Schleife die restlichen $k-2$ Startwerte. Dazu wird in jedem Schritt für jeden bereits ermittelten Startwert der Abstand zu allen r Instanzen aus U berechnet, ausgenommen die bisher ermittelten Startwerte. Im ersten Schritt der Schleife sind also $2(r-2)$ Abstände zu berechnen, im zweiten Schritt $3(r-3)$ usw. bis zum letzten Schritt, in dem $(k-1)(r-(k-1))$ Abstände berechnet werden. Addiert ergibt sich für die Anzahl $T_{iter}(r,k)$ der Elementaroperationen folgende Summe [Zer01]:

$$T_{iter}(r,k) = \sum_{i=2}^{k-1} i(r-i) \quad . \tag{4.2}$$

Die Summe (4.2) lässt sich weiter wie folgt berechnen:

$$\begin{aligned} T_{iter}(r,k) &= \sum_{i=2}^{k-1} i(r-i) = \sum_{i=2}^{k-1} ir - i^2 = r \cdot \sum_{i=2}^{k-1} i - \sum_{i=2}^{k-1} i^2 \\ &= r \cdot \left(\frac{k(k-1)}{2} - 1 \right) - \left(\frac{k(k-1)(2k-1)}{6} - 1 \right) \\ &= r \cdot \left(\frac{3(k^2-k)-6}{6} \right) \left(\frac{(k^2-k)(2k-1)-6}{6} \right) \\ &= \frac{r \cdot (3k^2-3k-6) - (2k^3-3k^2+k-6)}{6} \quad . \end{aligned} \tag{4.3}$$

Wird das Einfügen der Startwerte in eine Liste ebenfalls als eine Elementaroperation aufgefasst, so werden der Startwertliste insgesamt $T_{list}(k) = k$ Startwerte hinzugefügt. Die Gesamtanzahl $T_{ges}(r,k)$ der Elementaroperationen für den AML-Algorithmus ergibt sich aus der Addition der Anzahlen $T_{init}(r)$ (4.1), $T_{iter}(r,k)$ (4.3) und $T_{list}(k)$ für die einzelnen Schritte:

$$T_{ges}(r,k) = T_{init}(r) + T_{iter}(r,k) + T_{list}(k) \quad . \tag{4.4}$$

Sei nun das Laufzeitverhalten des Algorithmus in den verschiedenen Fällen *best case*), *worst case* und *average case* betrachtet.

4.2.1 Best case

Die Gesamtlaufzeit hängt von der Anzahl unterschiedlicher Instanzen r und der Anzahl zu bestimmender Cluster k ab. Der Algorithmus ermittelt mindestens zwei Startwerte (Initialschritt), so dass für die Parameter $k \geq 2$ und $|U| \geq k$ gilt. Für den besten Fall sind k und $|U|$ so klein wie möglich zu wählen, d.h. $k = 2$ und $|U| = 2$:

$$T_{best} = T_{ges}(2,2) = \frac{2(2-1)}{2} + 2 = 3 \qquad . \tag{4.5}$$

Bezogen auf die empirischen Ergebnisse in Kapitel 5 und in den Arbeiten [Zer01] und [Tsc02] hat dieser beste Fall kaum eine Bedeutung, da die zu analysierenden Bilder mehr Pixel und Farben aufweisen. Interessanter sind daher die Analysen für *worst case* und *average case*.

4.2.2 Worst case

Zerbst [Zer01] erwähnt den Vorteil, dass AML im Vergleich zu neuronalen Netzen nicht auf der Anzahl $n = |S|$ der Instanzen insgesamt, sondern auf der Anzahl unterschiedlicher Instanzen $r = |U|$ arbeitet. Im Bereich der Erosionsproblematik entspricht n der Anzahl der Pixel in einem zu analysierenden Bild und r der Anzahl der Farben. Für den *worst case* ist anzunehmen, dass $U = S$ und damit auch $r = n$ gilt. Ferner entspricht im *worst case* die Anzahl der zu bestimmenden Cluster k der Anzahl der Instanzen, so dass jede Instanz in einen eigenen Cluster fällt. Damit ergibt sich für $T_{worst} = T_{ges}(n,n)$:

$$\begin{aligned} T_{worst} &= T_{ges}(n,n) \\ &= \frac{n(n-1)}{2} + \frac{n \cdot (3n^2 - 3n - 6) - (2n^3 - 3n^2 + n - 6)}{6} + n \\ &= \frac{3n^2 - 3n + (3n^3 - 3n^2 - 6n) - (2n^3 - 3n^2 + n - 6) - 6n}{6} \\ &= \frac{n^3 - 3n^2 - 14n - 6}{6} \qquad . \end{aligned} \tag{4.6}$$

Für das Wachstum von T_{worst} unter Verwendung der Θ-Notation gilt $T_{worst} = \Theta(n^3)$. Im

schlechtesten Fall hat AML also eine kubische Gesamtlaufzeit, die sich aus der Wahl $k = n$ ergibt. Im Sinne der Clusteranalyse (siehe Abschnitt 1.1.2) und einer erwünschten Informationsreduktion ist eine solche Wahl für k jedoch äußerst unrealistisch, da k soweit wie möglich minimiert werden soll, ohne dass die Güte der resultierenden Clusterung darunter leidet.

4.2.3 Average case

Die in Abschnitt 4.2.2 ermittelte kubische Laufzeit für den Schritt $T_{iter}(n, n)$ tritt nur in einer Extremsituation auf, in der genauso viele Cluster wie Instanzen erzeugt werden sollen. In der Regel ist k unabhängig von der jeweiligen Anwendung deutlich kleiner als die Anzahl der Instanzen.

Annahmen über die Anzahl r unterschiedlicher Instanzen für beliebige Anwendungen zu treffen ist schwierig. Im Bereich der Analyse von Bildern ist die Zahl der Farben oft kleiner als die Anzahl der Pixel, so dass für die meisten Eingaben $r < n$ angenommen werden darf (siehe Abschnitt 5.3.1). Zudem ist r für True-Color-Bilder in jedem Fall beschränkt durch die Farbtiefe von 24 bit, was einer Maximalzahl von $16.777.216$ Farben entspricht (siehe Abschnitt 2.3).

Eine ausführliche Analyse der durchschnittlichen Laufzeit für alle möglichen Eingaben wird hier nicht durchgeführt. Stattdessen wird — ohne weitere Annahmen über r zu machen — die Gesamtanzahl von Elementaroperationen bei kleinstmöglichem $k = 2$ betrachtet:

$$T_{ges}(r, 2) = \frac{r(r-1)}{2} + 2 = \frac{r^2 - r + 4}{2} \quad . \tag{4.7}$$

Für $k = 2$ wird der Iterationsschritt des Algorithmus nicht durchlaufen. $T_{init}(r)$ hängt allein von r ab. Bei Verwendung der Θ-Notation wird ersichtlich, dass $T_{ges}(r, 2) = \Theta(r^2)$ gilt und die quadratische Laufzeit des Initialschrittes die Gesamtlaufzeit des Algorithmus für ein kleinstmögliches k dominiert. Die Vermutung liegt nahe, dass diese Aussage allgemein für kleine k gilt, trotz des kubischen Laufzeitverhaltens von $T_{iter}(r, k)$ in k.

Die Vermutung wird durch den Graphen in Abb. 4.1 bestätigt, der die Anzahl der Elementaroperationen für die Schritte $T_{init}(r)$ und $T_{iter}(r, k)$ anhand einer logarithmischen Skala in Abhängigkeit der Parameter $0 \leq r \leq 100.000$ und $2 \leq k \leq 250$ darstellt. Für kleine k ist die Anzahl der Elementaroperationen im Schritt $T_{init}(r)$ tatsächlich größer als im Schritt $T_{iter}(r, k)$ (für sehr kleine k sogar um Größenordnungen). Ab einem bestimmten k, das von r abhängt, übersteigt die Anzahl der Operationen von $T_{iter}(r, k)$ dann allerdings die von $T_{init}(r)$ (im Gra-

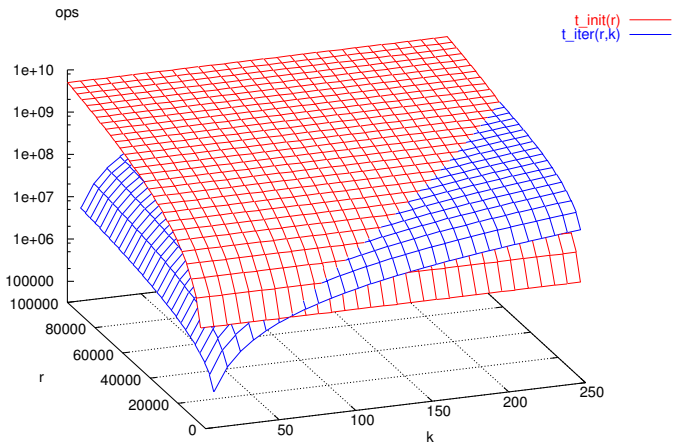


Abb. 4.1: Anz. der Elementaroperationen (ops) für $T_{init}(r)$ und $T_{iter}(r,k)$ des AML-Algorithmus in Abhängigkeit von $0 \leq r \leq 100.000$ und $2 \leq k \leq 250$.

phen der Schnitt der beiden dargestellten Flächen).

Die hier gewonnenen Ergebnisse relativieren die optimistischen Annahmen bezüglich der Laufzeit von ML in [Zer01], die basierend auf empirischen Zahlenbeispielen getroffen wurden. Die asymptotische Laufzeit des AML-Algorithmus ist mit $\Theta(r^2)$ selbst für zu erwartende kleine k für alle r quadratisch. Für den in [Zer01] durchgeführten Vergleich mit neuronalen Netzen bedeutet dies, dass sich für vorgegebenes k und eine vorgegebene Anzahl von Trainingszyklen immer ein r finden lässt, das zu einer höheren Laufzeit AMLs im Vergleich mit den in Abschnitt 1.6 vorgestellten selbstorganisierenden Merkmalskarten führt.

4.3 Definition des Farthest-Pair-Problems

Es stellt sich die Frage, ob die Gesamtlaufzeit des AML-Algorithmus für die meisten Eingaben verbessert werden kann. Zu diesem Zweck konzentrieren sich die nächsten Abschnitte auf den Initialschritt, dessen Laufzeit die Gesamtlaufzeit des Algorithmus für ein kleines k deutlich dominiert (siehe Abschnitt 4.2.3).

Im Initialschritt gilt es, zwei Instanzen aus U zu bestimmen, die maximalen Abstand zueinander haben. Dies entspricht der Bestimmung des Durchmessers einer Punktmenge. Dasselbe Problem muss auch durch das Clusteringverfahren Complete-Linkage (siehe Abschnitt 1.5.2) gelöst werden. In der algorithmischen Geometrie (siehe Kapitel 3) ist dieses Problem auch unter

dem Namen *Farthest-Pair-Problem* oder *Diameter-Problem* bekannt. Im Rahmen der vorliegenden Anwendung lässt sich das Problem formal wie folgt fassen:

Definition 4.1 (Farthest-Pair-Problem) *Sei $X \subseteq \mathbb{R}^d$ ein Vektorraum der Dimension d und $S \subseteq X$ eine Menge von Vektoren. Sei $U \subseteq S$ die Menge unterschiedlicher Vektoren aus S. Sei weiter*

$$dist : X \times X \rightarrow \mathbb{R}^+$$

eine auf dem Raum X definierte Abstandsfunktion. Gesucht ist ein Paar von Vektoren (x, y), für das gilt:

$$(x, y) \in \left\{ (p, q) \,\middle|\, \max_{p,q \in U} dist(p, q) \right\} \quad .$$

AML berechnet im Initialschritt paarweise die Abstände zwischen allen Punkten aus U, was zu einer asymptotischen Laufzeit von $\Theta(dr^2)$ führt. Im Rest dieses Kapitels werden zwei effizientere Lösungsmöglichkeiten für das Farthest-Pair-Problem vorgestellt. Eine bekannte Lösung basiert auf der Berechnung der konvexen Hülle (siehe Abschnitt 4.4), eine andere Lösung verwendet einen selbst entwickelten Pruning-Algorithmus (siehe Abschnitt 4.5).

4.4 Lösung mittels der konvexen Hülle

In [PS85] findet sich das folgende Theorem von Hocking-Young [HY61]:

Theorem 4.1 *Der Durchmesser einer Menge entspricht dem Durchmesser ihrer konvexen Hülle.*

In der Ebene lässt sich der Durchmesser einer Menge von n Punkten also durch Berechnung der konvexen Hülle bestimmen. Ein Algorithmus wie Quickhull (siehe Abschnitt 3.3.5) benötigt dafür im Durchschnitt $O(n \log n)$ Schritte. Die konvexe Hülle ist in diesem Fall ein konvexes Polygon P (Theorem 3.1). Danach wird der Durchmesser der konvexen Hülle bestimmt. Eine Berechnung aller paarweisen Abstände zwischen den Eckpunkten der konvexen Hülle ist jedoch nach wie vor ineffizient, falls die Ecken des Polygons den Punkten aus der

Menge entsprechen. Gesucht ist ein effizienter Algorithmus zur Berechnung des Durchmessers eines konvexen Polygons.

Ein Theorem von Yaglom-Boltyanskii [YB61] lautet:

Theorem 4.2 *Der Durchmesser einer konvexen Figur ist der größte Abstand zwischen parallelen Geraden des Trägers.*

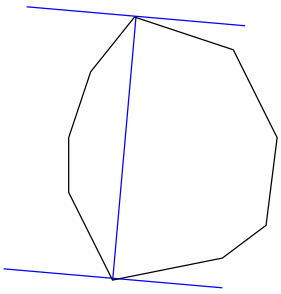


Abb. 4.2: Größter Abstand zwischen parallelen Geraden des Trägers

Wie in Abb. 4.2 gezeigt, können die parallelen Geraden nicht jedes Paar von Punkten tangieren. Tatsächlich gibt es nur bestimmte Paare von Punkten, deren Tangenten parallel zueinander sein können [PS85]. Diese Paare werden im Folgenden als *entgegengesetzt* bezeichnet. Das Problem besteht nun darin, die entgegengesetzten Paare zu finden, ohne alle Paare von Punkten testen zu müssen.

Seien die Ecken $u_1, u_2, \ldots, u_m$ eines konvexen Polygons P (angeordnet entgegen dem Uhrzeigersinn) gegeben. Eine Ecke u_i von P sei am weitesten von einer Kante $\overline{u_{k-1}u_k}$ entfernt, wenn u_i am weitesten von der Geraden entfernt ist, auf der $\overline{u_{k-1}u_k}$ liegt.

Beginnend mit der Ecke u_k seien die Ecken von P entgegen dem Uhrzeigersinn betrachtet. Sei u_R die Ecke, die am weitesten von $\overline{u_{k-1}u_k}$ entfernt ist. Sei analog u_L die Ecke, die am weitesten von $\overline{u_ku_{k+1}}$ entfernt ist (siehe Abb. 4.3(a)). Die Ecken der Folge von Ecken zwischen u_R und u_L (inklusive) bilden jeweils ein entgegengesetztes Paar mit u_k. Betrachte dazu den äußeren Winkel $\alpha_k > \pi$, der durch $\overline{u_{k-1}u_k}$ und $\overline{u_ku_{k+1}}$ gebildet wird. Das Paar (u_k, u_s) ist nur dann ein entgegengesetztes Paar, wenn es eine gerade Linie im Schnitt der beiden Winkel α_s und α_k gibt (siehe Abb. 4.3(b)) [PS85, Che96]. Jede Ecke der Folge von Ecken zwischen u_R und u_L besitzt diese Eigenschaft, da die Kanten einen konvexen Linienzug bilden. Alle anderen Ecken bilden kein entgegengesetztes Paar mit u_k.

Diese Beobachtung führt zu dem in Abb. 4.4 gezeigten Algorithmus. Dabei berechnet die Funktion Area(u_k,u_{k+1},u) die vorzeichenbehaftete Fläche des Dreiecks (u_k, u_{k+1}, u). Mit Hilfe dieser Funktion lässt sich der von einer Kante $\overline{u_ku_{k+1}}$ am weitesten entfernte Punkt bestimmen [PS85].

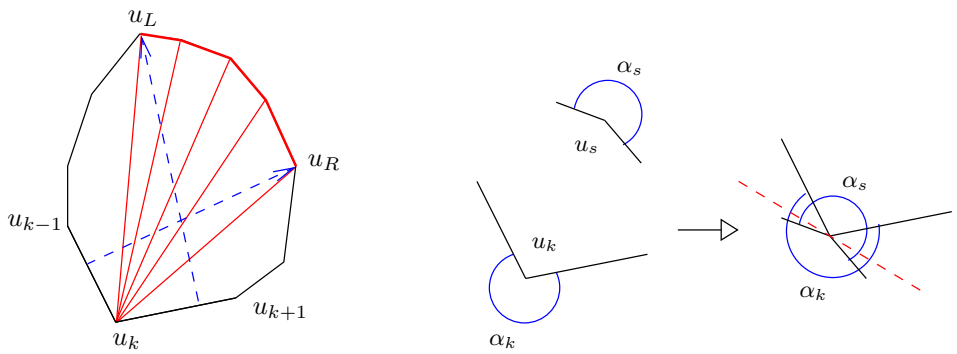


Abb. 4.3: Entwicklung des Algorithmus $AntipodalPairs$ (aus [PS85])

```
function AntipodalPairs(P)
    Pairs ← ()
    p ← last(P)
    q ← first(P)
    while Area(p, next(p), next(q)) > Area(p, next(p), q) do
        q ← next(q)
    q0 ← q
    while q ≠ first(P) do
        p ← next(p)
        insert(Pairs,(p,q))
        while Area(p, next(p), next(q)) > Area(p, next(p), q) do
            q ← next(q)
            if (p,q) ≠ (q0, first(P)) then
                insert(Pairs, (p,q))
        if Area(p, next(p), next(q)) = Area(p, next(p), q) then
            if (p,q) ≠ (q0, last(P)) then
                insert(Pairs, (p, next(q)))
    return Pairs
```

Abb. 4.4: Pseudocode für den Algorithmus $AntipodalPairs$ (aus [PS85])

p	q	insert
u_{12}		
	u_1	
	u_2	
	u_3	
	u_4	
	u_5	
	u_6	
u_1		(u_1, u_6)
	u_7	(u_1, u_7)
u_2		(u_2, u_7)
	u_8	(u_2, u_8)
	u_9	(u_2, u_9)
u_3		(u_3, u_9)
	u_{10}	(u_3, u_{10})
	u_{11}	(u_3, u_{11})
u_4		(u_4, u_{11})
u_5		(u_5, u_{11})
	u_1	

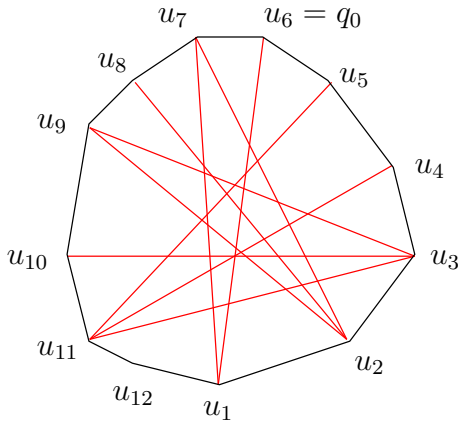


Abb. 4.5: Beispiel für einen Durchlauf von $AntipodalPairs$ (in Anlehnung an [PS85])

Der Ablauf des Algorithmus lässt sich durch das Beispiel in Abb. 4.5 nachvollziehen. Die erste while-Schleife bestimmt den Punkt u_R (im Pseudocode q_0 genannt), der am weitesten von der Kante $\overline{u_{k-1}u_k}$ entfernt liegt. Die nächste while-Schleife konstruiert die Folge der Ecken bis u_L, die entgegengesetzte Paare mit u_k bilden können. Die beiden Zeiger p und q werden dazu entgegen dem Uhrzeigersinn um das Polygon herumbewegt: p von der letzten Ecke u_m des Polygons bis u_R und q von u_R bis u_m. Wenn ein Zeiger weiterbewegt oder ein paralleles Paar von Kanten gefunden wird (Testen der Dreiecksflächen auf Gleichheit), wird das aktuelle entgegengesetzte Paar in die Liste Pairs eingefügt. Das letzte generierte Paar ist (u_R, u_m).

Jedes Paar wird nur einmal generiert und in die Liste Pairs eingefügt. In der zweiten while-Schleife werden die Zeiger insgesamt m mal bewegt, falls es keine parallelen Kanten gibt. Die Anzahl paralleler Kanten ist maximal $\lfloor m/2 \rfloor$, so dass für die Anzahl entgegengesetzter Paare $\leq 3m/2$ gilt [PS85]. Das bedeutet, dass sich der Durchmesser eines konvexen Polygons in linearer Zeit abhängig von der Anzahl der Ecken bestimmen lässt.

Gemäß den Theoremen 4.1 und 4.2 hat die Bestimmung des Durchmessers einer endlichen Menge von n Punkten in der Ebene somit eine Zeitkomplexität von $O(n \log n)$. Es lässt sich weiter festhalten, dass die Anzahl maximaler Abstände einer endlichen Punktmenge in der Ebene durch $O(n)$ beschränkt ist. Diese Schranke wird etwa von einem Polygon erreicht, dessen Kanten alle die gleiche Länge aufweisen (*reguläres Polygon*).

4.5 Entwicklung eines Pruning-Algorithmus

Für die Bestimmung des Durchmessers einer endlichen Punktmenge in der Ebene eignet sich die in Abschnitt 4.4 vorgestellte Lösung. In höheren Dimensionen ist das Problem jedoch ungleich komplexer. So besagt ein Theorem von Erdös [Erd60]:

> **Theorem 4.3** *Die maximale Distanz zwischen n Punkten im d-dimensionalen Raum kann $n^2/(2 - 2/\lfloor d/2 \rfloor) + O(n^{2-\epsilon})$ mal vorkommen $(d > 3, \epsilon > 0)$.*

Im schlechtesten Fall ist die Anzahl entgegengesetzter Paare so hoch, dass der in Abschnitt 4.4 vorgestellte Algorithmus $AntipodalPairs$ nicht mehr effizient ist. In diesem Zusammenhang bezeichnen Preparata und Shamos [PS85] das Problem der Bestimmung des Durchmessers als „Quelle der Frustration“. In Dimensionen > 3 kann die Struktur der konvexen Hülle zudem so komplex werden ($\Omega(n^{\lfloor d/2 \rfloor})$, siehe Abschnitt 3.4), dass die paarweise Abstandsberechnung zwischen allen Punkten effizienter ist.

Clusterprobleme weisen im Allgemeinen eine höhere Dimension als 2 auf. So arbeiten etwa Systeme zur Klassifikation von Dokumenten mit Vektoren der Dimension 315 [Koh01]. Eine effiziente Lösung des Farthest-Pair-Problems ist daher essentiell.

Für die dritte Dimension gibt es einen Pruning-Algorithmus, der für die meisten Eingaben eine Zeitkomplexität von $O(n \log n)$ erreicht. Die Grundideen dieses Ansatzes und eine Erweiterung auf höhere Dimensionen werden in den nächsten Abschnitten diskutiert.

4.5.1 Grundideen

Für die Berechnung des Durchmessers einer endlichen Menge von Punkten werden einige Ecken der konvexen Hülle und deren Flächen nicht benötigt. Die konvexe Hülle stellt lediglich eine Hilfskonstruktion dar. Es liegt nahe, dass es noch andere Wege zur Lösung des Farthest-Pair-Problems gibt.

Es sei bereits ein Abstand zwischen zwei beliebigen Punkten aus der Punktmenge bestimmt. Gibt es Punkte, die — basierend auf dem bis dahin ermittelten größten Abstand — aus der weiteren Betrachtung ausgeschlossen werden können (*pruning*)?

Einen solchen Ansatz verfolgt der von Clarkson und Shor [CS89] entwickelte randomisierte Pruning-Algorithmus, der in [AS98, AS00] vorgestellt wird (siehe Abb. 4.6).

```
function Diameter(U)
    wähle zufällig gleichverteilt einen Punkt p ∈ U
    q ← am weitesten von p entfernter Punkt
    I ← ⋂_{p'∈U} B(p', dist(p, q))
    U_1 ← U \ I
    if U_1 = ∅ then
        return (p, q)
    return Diameter(U_1)
```

Abb. 4.6: Pseudocode für den Algorithmus $Diameter$ (aus [AS98, AS00])

Betrachtet zunächst der dreidimensionale Fall. Es wird zufällig ein Punkt p aus der Menge U ausgewählt und der Punkt q bestimmt, der am weitesten davon entfernt ist. Danach wird um jeden Punkt $p' \in U$ eine Kugel $B(p', dist(p, q))$ eingezeichnet, deren Radius dem Abstand zwischen p und q entspricht. Alle Punkte, die in der Schnittmenge dieser Kugeln liegen, können aus der weiteren Betrachtung ausgeschlossen werden. Denn zu keinem dieser Punkte kann es einen Punkt geben, der einen größeren Abstand als den bisher bestimmten größten Abstand hat. Die Punkte werden aus U entfernt und der Algorithmus für die übrigen Punkte $U_1 = U_1 \setminus I$ rekursiv aufgerufen, bis keine Punkte mehr übrig sind ($U_1 = \emptyset$).

Die Berechnung der Differenz $U \setminus I$ hat eine zu erwartende Laufzeit von $O(r \log r)$ mit $r = |U|$, sofern ein effizienter Algorithmus für die Suche nach Punkten verwendet wird (siehe z. B. [ST86]). Die Schnittmenge kongruenter Kugeln in $\mathbb{R}^3$ hat lineare Komplexität. Clarkson und Shor [CS89] berechnen die Schnittmenge von r Kugeln in $\mathbb{R}^3$ mit einem randomisierten Algorithmus in $O(r \log r)$ zu erwartender Zeit. Damit ist die zu erwartende Laufzeit für jeden Rekursionsschritt gegeben durch $O(r \log r)$. Da p zufällig gewählt wird, ist $|U_1| \leq 2|U|/3$ mit hoher Wahrscheinlichkeit, was zu einer erwartenden Gesamtlaufzeit des Algorithmus von $O(r \log r)$ führt [AS98, AS00].

Eine von Amato et. al. [AGR94] entwickelte derandomisierte Version des Algorithmus erreichte zunächst eine Zeitkomplexität von $O(r \log^3 r)$. Weitere deterministische Algorithmen folgten [CEGS93, MS96, Ram97b]. Ramos [Ram97a] und Bespamyatnikh [Bes98, Bes01] konstruierten deterministische $O(r \log^2 r)$-Algorithmen für das Problem. Schließlich erreichte Ramos [Ram00b, Ram00a] eine Zeitkomplexität von $O(r \log r)$.

Laut Finocchiaro und Pellegrini [FP99] lässt sich der besprochene Pruning-Algorithmus nicht auf höhere Dimensionen erweitern. Ein von Yao [Yao82] entwickelter Algorithmus für

$d \geq 4$ erreicht eine Zeitkomplexität von $O(r^{2-a(d)} \log^{1-a(d)} r)$ mit $a(d) = 2^{-(d+1)}$. Neben exakten Algorithmen für die Bestimmung des Durchmessers in höheren Dimensionen existieren zudem Approximationsverfahren [EK89, BOR99, Ind99, FP99].

In den folgenden Abschnitten wird eine selbst entwickelte Modifikation des Pruning-Algorithmus vorgestellt, die den Durchmesser einer endlichen Punktmenge in höheren Dimensionen exakt bestimmt. Die Laufzeit ist nach unten durch $\Omega(dr + r \log r)$ und nach oben durch $O(dr^2)$ beschränkt, in der Praxis für die meisten Eingaben jedoch deutlich besser (siehe Abschnitt 5.2).

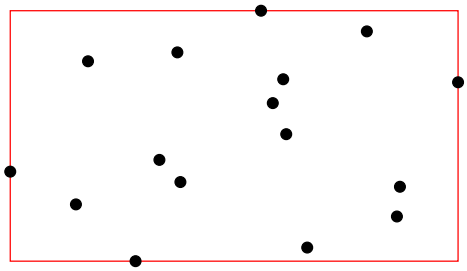
4.5.2 Modifikation des Pruning-Kriteriums

Die Grundideen aus Abschnitt 4.5.1 lassen sich auf höhere Dimensionen übertragen: schließe die Punkte aus der weiteren Betrachtung aus, zu denen es keine Punkte mehr gibt, die größeren Abstand als den bisher bestimmten größten Abstand haben. Der Algorithmus, in dem die Schnittmenge der $|U|$ Kugeln berechnet wird, bestimmt die auszuschließenden Punkte exakt. Im Folgenden wird ein weniger restriktives Kriterium vorgestellt, mit dem sich auszuschließende Punkte auch in höheren Dimensionen effizient bestimmen lassen.

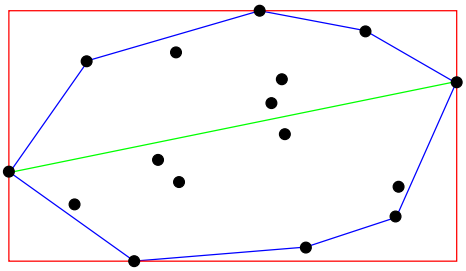
Entwicklung eines neuen Pruning-Kriteriums

Einige Algorithmen zur Berechnung der konvexen Hülle (siehe Abschnitt 3.3.1) bestimmen im ersten Schritt einen Punkt, der sicher zur konvexen Hülle gehört. Ein solcher Punkt ist extrem in einer seiner Koordinaten (z. B. am weitesten links/rechts oder oben/unten im zweidimensionalen Fall). In der Dimension d lassen sich auf diese Weise $2d$ Extrempunkte (und damit Punkte der konvexen Hülle) bestimmen. Werden durch jeden der Extrempunkte achsenparallele Geraden gezeichnet, so ergibt sich im zweidimensionalen Fall über die Schnittpunkte ein achsenparalleles Rechteck (siehe Abb. 4.7(a)), in höheren Dimensionen ein achsenparalleler Hyperquader. Durch den Hyperquader wird die Punktmenge und ihre konvexe Hülle vollständig eingeschlossen, d.h. kein Punkt liegt außerhalb des Hyperquaders.

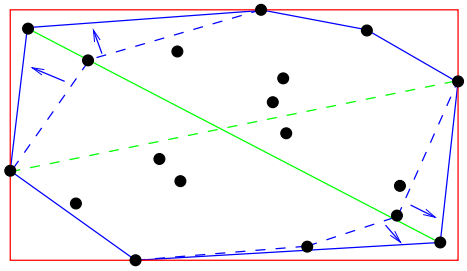
Sei die konvexe Hülle $CH(U)$ und ihr Durchmesser für eine endliche Menge von Punkten $U = \{x_1, x_2, \ldots, x_r\}$ in der Ebene bereits bestimmt und das sich aus den Extrempunkten ergebende Rechteck eingezeichnet (siehe Abb. 4.7(b)). Wo müssten sich in Bezug auf das Rechteck Punkte befinden, die einen größeren Abstand als den bereits bestimmten größten Abstand haben? Sicherlich in der Nähe der Ecken, denn der größtmögliche Abstand in einem Rechteck ist



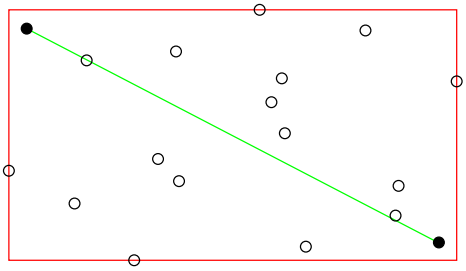
(a) Bestimme umschließendes achsenparalleles Rechteck



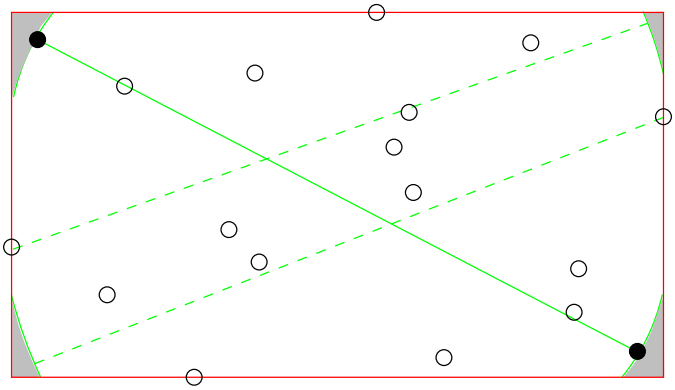
(b) Konvexe Hülle mit Durchmesser δ



(c) Zwei Punkte mit größerem Abstand als δ werden hinzugefügt



(d) Punkte (dargestellt als offene Kreise) werden gemäß restriktivem Pruning-Kriterium entfernt



(e) Zu vier Punkten könnte es noch Punkte innerhalb des Rechtecks mit Abstand $> \delta$ geben

Abb. 4.7: Entwicklung des modifizierten Pruning-Kriteriums

dessen Diagonale. Werden diese Punkte x_{r+1} und x_{r+2} der Menge hinzugefügt, so ergibt sich eine neue konvexe Hülle (siehe Abb. 4.7(c)).

Im Sinne des restriktiven Pruning-Kriteriums werden aus der Beispielmenge $U \backslash \{x_{r+1}, x_{r+2}\}$ die Punkte ausgeschlossen, die zur Schnittmenge aller Kreise mit dem Radius $dist(x_{r+1}, x_{r+2})$ gehören (siehe Abb. 4.7(d)). Zur Ermittlung eines weniger restriktiven Pruning-Kriteriums stellt sich die Frage, welche Eigenschaften die ausgeschlossenen Punkte und die neuen Punkte x_{r+1}, x_{r+2} in Bezug auf das umschließende Rechteck und dessen Ecken haben.

Für nahezu alle ausgeschlossenen Punkte des Beispiels gilt, dass der Kreis mit dem Radius $dist(x_{r+1}, x_{r+2})$ das Rechteck vollständig abdeckt. Für die beiden Punkte, die zuvor das Paar mit maximalem Abstand gebildet haben und für die neuen Punkte x_{r+1} und x_{r+2} hingegen gilt dies nicht. Für diese Punkte könnte es noch weitere Punkte innerhalb des Rechtecks geben, die größeren Abstand als den bisher bestimmten größten Abstand haben (siehe Abb. 4.7(e)). Diese Beobachtung lässt sich verallgemeinern.

Satz 4.1 *Sei $U = \{x_1, x_2, \ldots, x_r\}$ eine endliche Menge von Punkten im d-dimensionalen Raum. Sei $HC(U)$ ein d-dimensionaler achsenparalleler Hyperquader, der die Punktmenge ganz einschließt und an jeder Seite tangiert. Sei ferner*

$$\delta = dist(x_i, x_j) \quad \textit{mit} \quad i, j \in \{1, \ldots, r\}$$

eine Distanz zwischen zwei Punkten aus der Menge. Falls ein Punkt $x \in U$ zu allen Ecken von $HC(U)$ einen kleineren oder gleichen Abstand als δ hat, so gibt es zu x keinen Punkt $y \in U$ mit einem größeren Abstand als δ.

BEWEIS. Die Korrektheit ergibt sich aus der Eigenschaft des Hyperquaders $HC(U)$, die Punktmenge ganz einzuschließen. Hat ein Punkt $x \in U$ zu allen Ecken des Hyperquaders einen kleineren oder gleichen Abstand als δ, so deckt die Hyperkugel mit Radius δ um den Punkt x den Hyperquader vollständig ab. Folglich würde ein Punkt $y \in U$ mit größerem Abstand zu x als δ außerhalb der Hyperkugel und damit auch außerhalb des Hyperquaders liegen — also nicht mehr zur Punktmenge gehören. □

Das modifizierte Pruning-Kriterium im Algorithmus $TryToPrune$ (siehe Abb. 4.8) schließt damit gemäß Satz 4.1 alle Punkte aus, zu denen es keine Punkte innerhalb des Hyperquaders $HC(U)$ geben kann, die größeren Abstand als den bisher bestimmten größten Abstand haben.

```
function TryToPrune(U, p, δ, p', q')
    q ← am weitesten von p entfernter Punkt
    if dist(p, q) > δ then
        δ ← dist(p, q)
        (p', q') ← (p, q)
        I ← ⋃_{p'∈U} { p' | dist(p', c) ≤ δ ∀ c ∈ HC(U) }
        U ← U \ I
    U ← U \ {p}
    return (U, δ, p', q')
```

Abb. 4.8: Pseudocode für den Algorithmus $TryToPrune$

```
function HighDiameter_rand(U)
    δ ← 0
    while U ≠ ∅ do
        wähle zufällig gleichverteilt einen Punkt p ∈ U
        (U, δ, p', q') ← TryToPrune(U, p, δ, p', q')
    return (p', q')
```

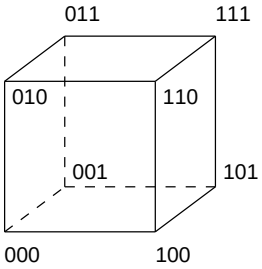
Abb. 4.9: Pseudocode für den Algorithmus $HighDiameter_{rand}$

Das modifizierte Kriterium ist im Gegensatz zum ursprünglichen Kriterium weniger restriktiv, da es unter Umständen nicht alle Punkte entfernt, die aus der Menge ausgeschlossen werden könnten. Betrachte dazu noch einmal Abb. 4.7(e): die Punkte auf der linken und rechten Kante des Rechtecks wären gemäß dem restriktiven Pruning-Kriterium ebenfalls aus der Menge ausgeschlossen worden.

Im Algorithmus $HighDiameter_{rand}$ (siehe Abb. 4.9) ist die Rekursion durch eine Iteration ersetzt worden und der Schritt zur Berechnung der Schnittmenge der $|U|$ Kugeln durch einen Test auf das neue Pruning-Kriterium. Da in jedem Iterationsschritt durch $TryToPrune$ mindestens der Punkt p aus U entfernt wird, ist der Abbruch des Algorithmus sichergestellt.

Bestimmung der Ecken des Hyperquaders

Die Ecken des Hyperquaders $HC(U)$ lassen sich vor Aufruf des Algorithmus ermitteln. Ein Hyperquader der Dimension d hat insgesamt 2^d Ecken [O'R00]. Für die Kantenlänge 1 ergeben sich die Koordinaten der Ecken durch binäres Zählen (siehe Abb. 4.10(a)). Im Falle eines



i	v_{i1}	v_{i2}	v_{i3}
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0
6	1	0	1
7	1	1	0
8	1	1	1

(a) Ecken $v_1, \ldots, v_8$ des Einheitswürfels

i	v_{i1}	v_{i2}	v_{i3}
1	$\min_1$	$\min_2$	$\min_3$
2	$\min_1$	$\min_2$	$\max_3$
3	$\min_1$	$\max_2$	$\min_3$
4	$\min_1$	$\max_2$	$\max_3$
5	$\max_1$	$\min_2$	$\min_3$
6	$\max_1$	$\min_2$	$\max_3$
7	$\max_1$	$\max_2$	$\min_3$
8	$\max_1$	$\max_2$	$\max_3$

(b) Ecken $v_1, \ldots, v_8$ eines beliebigen Hyperquaders

Abb. 4.10: Bestimmung der Ecken des Hyperquaders

beliebigen Hyperquaders müssen zunächst für alle Punkte komponentenweise die Extremwerte (Minima und Maxima) der d Koordinaten bestimmt werden. Seien

$$\begin{aligned} x_1 &= \{x_{11}, x_{12}, \ldots, x_{1d}\} \\ x_2 &= \{x_{21}, x_{22}, \ldots, x_{2d}\} \\ &\vdots \quad\quad \vdots \\ x_r &= \{x_{r1}, x_{r2}, \ldots, x_{rd}\} \end{aligned}$$

die Punkte der Menge U. Das Minimum $\min_c$ und das Maximum $\max_c$ für die Komponente c sind dann gegeben durch:

$$\min_c = \min_{i=1}^{r} x_{ic} \quad \text{und} \quad \max_c = \max_{i=1}^{r} x_{ic} \qquad .$$

Die Ecken ergeben sich aus der Binärdarstellung durch die Ersetzung einer 0 durch das Minimum und die Ersetzung einer 1 durch das Maximum der jeweiligen Komponente (siehe Abb. 4.10(b)). Die Ermittlung der Extremwerte benötigt $d \cdot r$ Schritte und die Konstruktion der Ecken 2^d Schritte.

Der Algorithmus $TryToPrune$ überprüft jeden der r Punkte daraufhin, ob er das modifizierte Pruning-Kriterium erfüllt. Dazu wird die Distanz zu allen 2^d Ecken berechnet. Wird das Kriterium erfüllt, so wird der Punkt aus U entfernt. Wird die Menge U als verkettete Liste implementiert, so benötigt das Entfernen eines Elements konstante Zeit. Der Pruning-Schritt benötigt also insgesamt $O(2^d r)$ Schritte. Die Laufzeit ist in Abhängigkeit von der Dimension exponentiell!

Der Test, ob ein Punkt das Pruning-Kriterium erfüllt, lässt sich jedoch vereinfachen. Die Ecke des Hyperquaders, die einem Punkt $x \in U$ gegenüberliegt, ist am weitesten von diesem Punkt entfernt. Ist der Abstand zwischen x und der gegenüberliegenden Ecke kleiner oder gleich als der bisher bestimmte größte Abstand, so ist auch der Abstand zu allen anderen Ecken kleiner oder gleich. Daher muss nur der Abstand zwischen x und der gegenüberliegenden Ecke berechnet werden.

Die gegenüberliegende Ecke zu einem Punkt x lässt sich in konstanter Zeit bestimmen. Dazu wird vor dem Start des Algorithmus der Mittelpunkt $mid(HC(U))$ des Hyperquaders $HC(U)$ aus den Extremwerten der jeweiligen Koordinaten berechnet:

$$mid(HC(U)) = \left(\frac{\min_1 + \max_1}{2}, \frac{\min_2 + \max_2}{2}, \ldots, \frac{\min_d + \max_d}{2} \right) \quad . \tag{4.8}$$

Wie in Abb. 4.10(a) zu sehen ist, entspricht die Binärdarstellung gegenüberliegender Ecken jeweils ihrer Negation. Die einem Punkt $x = (x_1, x_2, \ldots, x_d)$ gegenüberliegende Ecke $OC(x)$ ergibt sich damit zu

$$OC(x) = (o_1, o_2, \ldots, o_d)$$

$$\text{mit} \quad o_c = \begin{cases} \min_c , & x_c \leq \frac{\min_c + \max_c}{2} \\ \max_c , & x_c > \frac{\min_c + \max_c}{2} \end{cases} \quad , \quad \text{wobei} \;\; c \in \{1, \ldots, d\} \quad .$$

Durch die Änderung des Tests hat die Dimension geringeren Einfluss auf die Laufzeit von $TryToPrune$. Es ist weder nötig, den Abstand zu allen Ecken zu bestimmen, noch müssen alle Ecken vor dem Aufruf des Algorithmus konstruiert werden. Die Konstruktion der gegenüberliegenden Ecke benötigt d Schritte und die Anzahl der Schritte für das Pruning reduziert sich auf $O(dr)$.

Analyse

Die Gesamtlaufzeit von $HighDiameter_{rand}$ hängt davon ab, welche Punkte zufällig ausgewählt werden. Im *worst case* konzentrieren sich — wie in Abb. 4.11(a) gezeigt — nahezu alle Punkte in der Mitte, bis auf die Extrempunkte, die auf den Kanten des umgebenden Rechtecks (Hyperquaders) liegen. Sei als erster Punkt der am weitesten links gelegene Punkt in der Mitte ausgewählt und dann nacheinander die Punkte rechts davon.

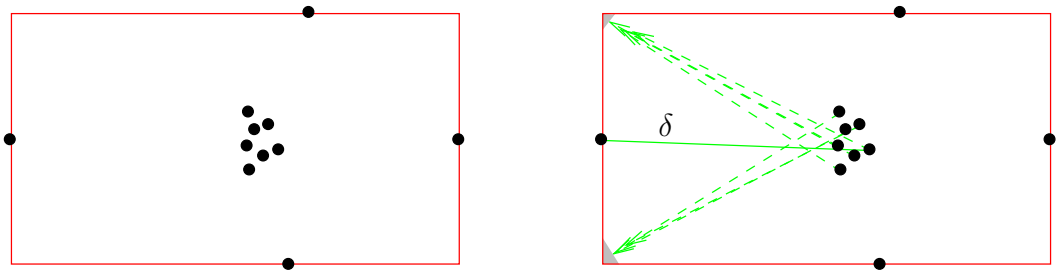


Abb. 4.11: *Worst case* für den Algorithmus $HighDiameter_{rand}$

In jedem Iterationsschritt wird ein Punkt q bestimmt, der am weitesten von p entfernt ist (in Abb. 4.11 ist das immer der Punkt auf der linken Kante des Rechtecks). Dazu werden in jedem Iterationsschritt jeweils $|U| - 1$ Abstände berechnet und ein Punkt aus U entfernt. Insgesamt werden also $\sum_{i=1}^{r-1} i$ Abstandsberechnungen durchgeführt. Der Abstand δ wird für die ausgewählten Punkte p von links nach rechts immer größer, so dass in jedem Iterationsschritt versucht wird, $|U|$ Punkte aus der Menge U auszuschließen. Die Abb. 4.11(b) zeigt, dass jeder der mittleren Punkte zu der jeweils gegenüberliegenden Ecke größeren Abstand als δ hat und es noch Punkte mit größerem Abstand nahe der Ecken geben könnte (graue Bereiche in Abb. 4.11(b)). Werden die Punkte auf der oberen und unteren Kante des Rechtecks erreicht, so sind bereits alle mittleren Punkte aus der Menge entfernt worden, da sie vorher besucht wurden. Durch den Algorithmus $TryToPrune$ werden daher in keinem Iterationsschritt Punkte mit Hilfe des Pruning-Kriteriums aus der Menge ausgeschlossen. Insgesamt ergeben sich $\sum_{i=2}^{r} i$ Versuche, Punkte aus der Menge zu entfernen. Die Gesamtanzahl $T_{worst}(r)$ von Operationen im *worst case* ergibt also:

$$\begin{aligned} T_{worst}(r) &= \sum_{i=1}^{r-1} i + \sum_{i=2}^{r} i \\ &= \frac{r(r-1)}{2} + \frac{r(r+1)}{2} - 1 \\ &= \frac{r^2 - r}{2} + \frac{r^2 + r}{2} - \frac{2}{2} \\ &= \frac{2r^2 - 2}{2} \\ &= r^2 - 1 \quad . \end{aligned} \tag{4.9}$$

Im schlechtesten Fall ist die Laufzeit von $HighDiameter_{rand}$ quadratisch in der Anzahl der Punkte (Instanzen) in U. Die asymptotische Laufzeit ist mit $O(dr^2)$ in keinem Fall schlechter

als bei der paarweisen Berechnung von Abständen zwischen allen Punkten.

Die durchschnittliche Laufzeit des Algorithmus lässt sich schwer ermitteln, da das weniger restriktive Pruning-Kriterium unter Umständen keine Punkte ausschließt und die Anzahl der entfernten Punkte stark von der Form der Eingabemenge und des Hyperquaders abhängt. Eine umfassende Analyse wird hier nicht durchgeführt. Es sei aber angemerkt, dass Punktmengen wie in Abb. 4.11(a), in der sich alle Punkte in der Mitte konzentrieren, nur eine Teilmenge aller möglichen Eingaben darstellen. Zudem ist bei zufälliger Auswahl der Punkte und für großes r die Wahrscheinlichkeit gering, dass Punkte in einer *worst case* Reihenfolge besucht werden: die Anzahl möglicher Durchlaufreihenfolgen (d.h. Permutationen der Punkte) ist für r Punkte mit $r!$ sehr groß.

Der nächste Abschnitt 4.5.3 diskutiert Möglichkeiten, wie die Auswahl der Punkte (und damit die Reihenfolge der Abstandsberechnungen) so verändert werden kann, dass schon in frühen Iterationsschritten mit hoher Wahrscheinlichkeit viele Punkte aus der Menge U entfernt werden.

4.5.3 Reihenfolge der Abstandsberechnungen

Werden die Punkte der Menge U dergestalt sortiert, dass sich die beiden Punkte mit maximalem Abstand nach der Sortierung in konstanter Zeit bestimmen lassen, so benötigt die Lösung des Farthest-Pair-Problems $O(dr \log r)$ Schritte. Eine solche Sortierung ist bisher nicht bekannt.

Mit der Ersetzung der zufälligen Auswahl von p im Algorithmus $HighDiameter_{rand}$ durch eine geordnete Auswahl lässt sich allerdings die Reihenfolge der Abstandsberechnungen beeinflussen. Je nach Gestalt der Eingabemenge ändert sich auf diese Weise die Wahrscheinlichkeit, mit der große Abstände in einem frühen Iterationsschritt gefunden werden. Dies wiederum kann Auswirkungen auf die Anzahl der Punkte haben, die sich in frühen Iterationsschritten aus der Menge U entfernen lassen — und damit auf die Laufzeit.

Sei dazu noch einmal Abb. 4.11(a) betrachtet. Wird schon im ersten Iterationsschritt der Abstand zwischen den beiden Punkten auf der linken und rechten Kante des Rechtecks berechnet, so können unmittelbar alle Punkte in der Nähe des Mittelpunktes von $HC(U)$ entfernt werden. Das entspricht dem *best case*.

In den folgenden Abschnitten werden drei mögliche Sortierungen vorgestellt.

```
function HighDiameter_lexi(U)
    sortiere die Punkte in U lexikografisch
    δ ← 0
    while U ≠ ∅ do
        p ← erster Punkt aus U
        (U, δ, p', q') ← TryToPrune(U, p, δ, p', q')
    return (p', q')
```

Abb. 4.12: Pseudocode für den Algorithmus $HighDiameter_{lexi}$

Sortierung der Punkte in lexikografischer Ordnung

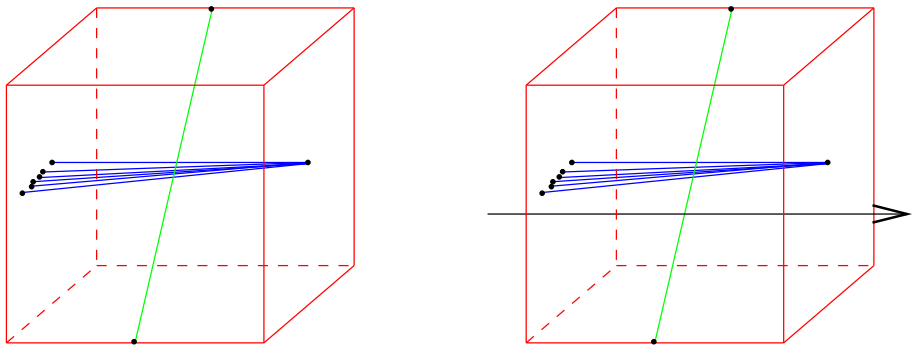
Die Implementation des AML-Algorithmus in der Anwendung PICANA sortiert die Instanzen aus softwaretechnischen Gründen *lexikografisch*. Die *lexikografische Ordnung* wird gewöhnlich auf Zeichenketten definiert [CLRS01], sie lässt sich jedoch für die vorliegende Anwendung auf Vektoren gleicher Dimension übertragen.

> **Definition 4.2 (Lexikografische Ordnung)** *Gegeben seien zwei Vektoren $x = (x_0, x_1, \ldots, x_d)$ und $y = (y_0, y_1, \ldots, y_d)$ aus $\mathbb{R}^d$. Der Vektor x ist lexikografisch kleiner als y, wenn eine ganze Zahl j mit $0 \leq j \leq d$ existiert, so dass $x_i = y_i$ für alle $i = 0, 1, \ldots, j-1$ und $x_j < y_j$.*

Die praktische Anwendung des AML-Algorithmus auf Erosionsbilder mit der Anwendung PICANA zeigt, dass der maximale Abstand für viele Bilder bereits im ersten Iterationsschritt gefunden wird. Das hängt damit zusammen, dass einige der Erosionsbilder viele Farben in der Mitte des RGB-Farbraumes aufweisen, aber nur wenige extreme Farben (z. B. Schwarz $= (0, 0, 0)$ und Weiß $= (1, 1, 1)$) in der Nähe der Ecken liegen (und damit großen Abstand haben). Die Sortierung der Punkte in lexikografischer Ordnung sorgt dafür, dass sich die Farben mit Extremen in jeder Komponente am Anfang und am Ende der sortierten Liste befinden.

Werden die Punkte aus U lexikografisch sortiert und die zufällige Auswahl von p im Algorithmus $HighDiameter_{rand}$ durch eine geordnete Auswahl ersetzt (siehe Abb. 4.12), so werden im *best case* bereits im ersten Iterationsschritt alle Punkte aus U entfernt. Der Algorithmus hat damit eine Zeitkomplexität von $\Omega(dr \log r)$.

Im *worst case* hat der Algorithmus $HighDiameter_{lexi}$ nach wie vor quadratische Laufzeit (die Rechnung entspricht der für $HighDiameter_{rand}$). Für das Beispiel in Abb. 4.13(a) wird



(a) (b)

Abb. 4.13: *Worst case* für den Algorithmus $HighDiameter_{lexi}$

der maximale Abstand erst im letzten Iterationsschritt gefunden. Das hängt damit zusammen, dass die Punkte bei lexikografischer Ordnung nur nach einer maßgeblichen Komponente (z. B. der ersten) sortiert werden. Die Abstände der Punkte auf der linken Seite zum Punkt ganz rechts werden mit wachsendem Komponentenwert immer größer. Der Algorithmus bewegt sich nur in einer Richtung durch den mehrdimensionalen Raum (siehe Abb. 4.13(b)). Dadurch werden Extremwerte in anderen Komponenten unter Umständen erst zuletzt gefunden. Das Problem kann für die gezeigte Art von Eingaben durch die im folgenden Abschnitt vorgestellte Durchlaufstrategie behoben werden.

Unabhängige Sortierung der Komponenten

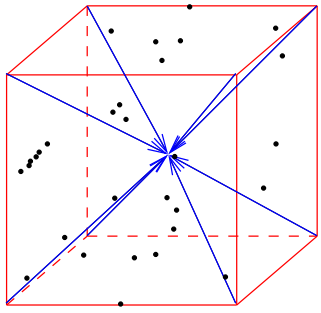


Abb. 4.14: Bewegung durch den mehrdimensionalen Raum aus allen Richtungen

Punkte mit maximalem Abstand gehören zu den Ecken der konvexen Hülle (Theorem 4.1). Punkte am Rand der Punktmenge weisen daher eine höhere Wahrscheinlichkeit auf, ein Paar mit maximalem Abstand zu bilden, als Punkte im Inneren der Menge. Die Abstände zwischen Punkten in der Nähe des Randes sollten deshalb möglichst früh berechnet werden.

Eine Durchlaufstrategie, die Punkte vom Rand des Hyperquaders aus zur Mitte hin besucht, erweist sich als geeignet. Dazu werden die Vektorkomponenten unabhängig voneinander sortiert. Dies erlaubt eine Bewegung aus mehreren verschiedenen Richtungen (z. B. von oben nach unten oder von links nach rechts im zweidimensionalen Fall) durch den mehrdimensionalen Raum (siehe Abb. 4.14).

	$c=1$	$c=2$	$c=3$
x_1	0.129	0.430	0.869
x_2	0.487	0.963	0.023
x_3	0.274	0.067	0.001
x_4	0.500	0.751	0.312
x_5	0.945	0.839	0.676
x_6	1.000	0.998	1.000
x_7	0.427	0.286	0.713
x_8	0.877	0.028	0.580
x_9	0.374	0.971	0.109

(a) Ungeordnete Menge U

i	IT_{i1}	IT_{i2}	IT_{i3}
1	$1_{0.129}$	$1_{0.430}$	$1_{0.869}$
2	$2_{0.487}$	$2_{0.963}$	$2_{0.023}$
3	$3_{0.274}$	$3_{0.067}$	$3_{0.001}$
4	$4_{0.500}$	$4_{0.751}$	$4_{0.312}$
5	$5_{0.945}$	$5_{0.839}$	$5_{0.676}$
6	$6_{1.000}$	$6_{0.998}$	$6_{1.000}$
7	$7_{0.427}$	$7_{0.286}$	$7_{0.713}$
8	$8_{0.877}$	$8_{0.028}$	$8_{0.580}$
9	$9_{0.374}$	$9_{0.971}$	$9_{0.109}$

(b) Ungeordnete Indextab.

i	IT_{i1}	IT_{i2}	IT_{i3}
1	$1_{0.129}$	$8_{0.028}$	$3_{0.001}$
2	$3_{0.274}$	$3_{0.067}$	$2_{0.023}$
3	$9_{0.374}$	$7_{0.286}$	$9_{0.109}$
4	$7_{0.427}$	$1_{0.430}$	$4_{0.312}$
5	$2_{0.487}$	$4_{0.751}$	$8_{0.580}$
6	$4_{0.500}$	$5_{0.839}$	$5_{0.676}$
7	$8_{0.877}$	$2_{0.963}$	$7_{0.713}$
8	$5_{0.945}$	$9_{0.971}$	$1_{0.869}$
9	$6_{1.000}$	$6_{0.998}$	$6_{1.000}$

(c) Geordnete Indextab.

Abb. 4.15: Erzeugung der Indextabelle IT

Für die Sortierung wird aus der ursprünglichen Menge U von Vektoren $x_1, x_2, \ldots, x_r$ (siehe Abb. 4.15(a)) eine Indextabelle IT erzeugt (siehe Abb. 4.15(b)). Die Tabelle IT besteht aus d Spalten für die unterschiedlichen Komponenten und die Indizes in den Zellen verweisen auf die ursprünglichen Vektoren der Menge U. Danach werden die Spalten nacheinander unabhängig sortiert, indem die Werte der ursprünglichen Vektoren in der jeweiligen Komponente miteinander verglichen werden (siehe Abb. 4.15(c)).

Gegeben sei folgende Durchlaufstrategie:

1. Wähle p nacheinander für Spalte $c = 1, \ldots, d$ als den Vektor $x_{IT_{1c}}$, auf den der Index IT_{1c} in der ersten Zeile IT_1, Spalte c der Indextabelle verweist. Entferne danach die erste Zeile aus der Indextabelle.

2. Falls die aktuelle Anz. der Zeilen > 0, wähle p nacheinander für Spalte $c = 1, \ldots, d$ als den Vektor $x_{IT_{|U|c}}$, auf den der Index $IT_{|U|c}$ in der letzten Zeile $IT_{|U|}$, Spalte c der Indextabelle verweist. Entferne danach die letzte Zeile aus der Indextabelle.

3. Wiederhole die Schritte 1 und 2, solange die Indextabelle noch Zeilen enthält.

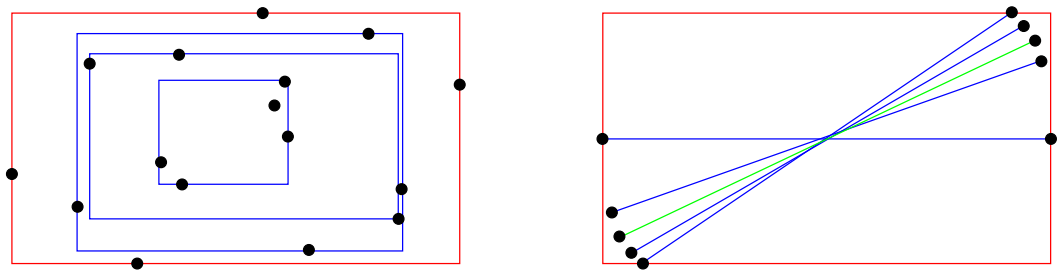
Der Algorithmus $HighDiameter_{comp}$ in Abb. 4.16 zeigt eine entsprechend modifizierte Version von $HighDiameter_{rand}$. Die Indizes in der Tabelle IT verweisen mehrfach (insgesamt d mal) auf die ursprünglichen Vektoren. Ein Vektor, der bereits ausgewählt wurde, muss nicht mehr betrachtet werden. Dafür sorgt jeweils der Test, ob p in U enthalten ist. Der Test verhindert zudem, dass $TryToPrune$ für Punkte aufgerufen wird, die bereits zuvor aus der Menge U entfernt wurden.

```
function HighDiameter_comp(U)
    erzeuge Indextabelle IT aus U
    sortiere Indextabelle IT komponentenweise
    δ ← 0
    while Anz. Zeilen von IT > 0 do
        foreach c ∈ {1, ..., d} do
            p ← x_{IT_{1c}}
            if p ∈ U then
                (U, δ, p', q') ← TryToPrune(U, p, δ, p', q')
        IT ← IT \ {IT_1}
        if Anz. Zeilen von IT > 0 do
            foreach c ∈ {1, ..., d} do
                p ← x_{IT_{|U|c}}
                if p ∈ U then
                    (U, δ, p', q') ← TryToPrune(U, p, δ, p', q')
            IT ← IT \ {IT_{|U|}}
    return (p', q')
```

Abb. 4.16: Pseudocode für den Algorithmus $HighDiameter_{comp}$

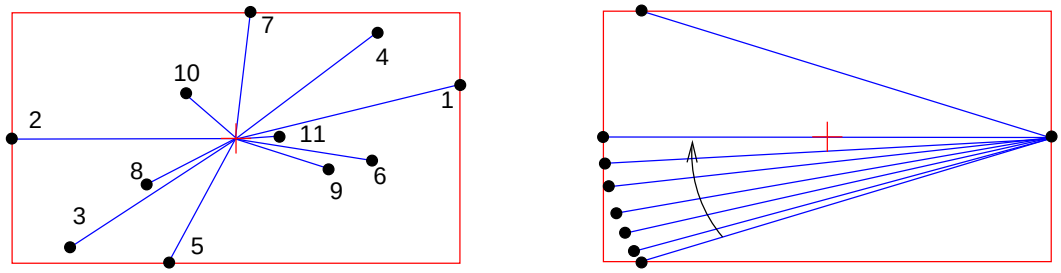
Eine Veranschaulichung der Durchlaufstrategie ist durch Abb. 4.17(a) gegeben. Es werden nacheinander immer kleinere achsenparallele Rechtecke (im mehrdimensionalen Fall Hyperquader) bestimmt, auf deren Kanten die jeweils nächsten Punkte (zur Mitte hin) liegen. Die Sortierung erzeugt auch den äußersten achsenparallelen Hyperquader, der im Zusammenhang mit dem modifizierten Pruning-Kriterium in Abschnitt 4.5.2 beschrieben wurde.

Mit der veränderten Durchlaufstrategie wird der in Abb. 4.11(a) dargestellte *worst case* zum *best case*. Der Algorithmus bestimmt bereits im ersten Iterationsschritt den maximalen Abstand zwischen den Punkten auf der linken und rechten Kante des Rechtecks. Danach werden alle



(a) Durchlaufstrategie (b) Anordnung im *worst case*

Abb. 4.17: Veranschaulichung von $HighDiameter_{comp}$



(a) Durchlaufstrategie (b) Anordnung im *worst case*

Abb. 4.18: Veranschaulichung von $HighDiameter_{mid}$

mittleren Punkte aus der Menge entfernt und der Algorithmus stoppt. Die Erstellung der Indextabelle benötigt $O(dr)$ und die Sortierung der Tabelle $O(dr \log r)$ Schritte. Die Entfernung aller Tabellenzeilen benötigt $O(dr)$ Schritte. Die asymptotische Laufzeit von $HighDiameter_{comp}$ ist also nach unten durch $\Omega(dr \log r)$ beschränkt.

Für die neue Durchlaufstrategie lässt sich ebenfalls ein *worst case* konstruieren, für den der Algorithmus insgesamt $\Theta(dr^2)$ Schritte benötigt. Betrachte dazu Abb. 4.17(b). Die Punkte sind so angeordnet, dass der Algorithmus einen immer größeren Abstand findet, jedoch ohne Punkte aus der Menge entfernen zu können.

Im Vergleich mit $HighDiameter_{lexi}$ ist für andere Arten von Eingabemengen jedoch die Wahrscheinlichkeit höher, große Abstände in frühen Iterationsschritten zu finden und damit mehr Punkte aus U zu entfernen.

Sortierung nach Abstand zum Mittelpunkt

In Abschnitt 4.5.2 wurden die Extremwerte der Koordinaten aller Punkte bestimmt und daraus der Mittelpunkt $mid(HC(U))$ des Hyperquaders $HC(U)$ errechnet (4.8).

Werden die Punkte nach ihrem Abstand zum Mittelpunkt des Hyperquaders (siehe Abb. 4.18(a)) sortiert, so kann der Punkt $p \in U$ in der Reihenfolge der Elemente aus der sortierten Liste ausgewählt werden. Dies führt zum Algorithmus $HighDiameter_{mid}$ (siehe Abb. 4.19).

Die Sortierung nach Abstand zum Mittelpunkt $mid(HS(U))$ sorgt dafür, dass die Punkte aus U nacheinander „von außen nach innen“ besucht werden.

Im Vergleich mit $HighDiameter_{comp}$ hat der Algorithmus $HighDiameter_{mid}$ zwei Vorteile. Zum einen ist der Algorithmus einfacher zu implementieren. So werden etwa in vie-

```
function HighDiameter_mid(U)
    sortiere die Punkte in U nach ihrem Abstand zu mid(HC(U))
    δ ← 0
    while U ≠ ∅ do
        p ← erster Punkt aus U
        (U, δ, p', q') ← TryToPrune(U, p, δ, p', q')
    return (p', q')
```

Abb. 4.19: Pseudocode für den Algorithmus $HighDiameter_{mid}$

len Programmiersprachen für die Verwaltung der Indextabelle komplizierte Zeigeroperationen benötigt. Zum anderen weist der Algorithmus im *best case* eine etwas bessere Laufzeit auf, da die Abstände zum Mittelpunkt als Skalare in $O(dr)$ Schritten berechnet und dann in $O(r \log r)$ Schritten sortiert werden können. Die Sortierung wird damit unabhängiger von der Dimension d und die untere Schranke für die Laufzeit verändert sich zu $\Omega(dr + r \log r)$.

Die obere Schranke verändert sich mit $O(dr^2)$ jedoch nicht. Betrachte dazu Abb. 4.18(b). Die Punkte haben im *worst case* alle gleichen Abstand zum Mittelpunkt und bleiben damit unsortiert. Werden die Punkte zufällig in der durch den Pfeil angedeuteten Reihenfolge besucht, so hat jeder der Punkte (bis auf den auf der oberen Kante) aufeinander folgend einen größeren Abstand zum Punkt auf der rechten Kante des Rechtecks. Allerdings könnte es zu jedem der Punkte innerhalb des Hyperquaders noch Punkte geben, die größeren Abstand haben. Das Pruning-Kriterium entfernt daher keine Punkte aus der Menge. Es ergibt sich eine ähnliche *worst case* Analyse wie für $HighDiameter_{rand}$.

4.5.4 Weitere mögliche Modifikationen

Bisher wird im Algorithmus $TryToPrune$ (siehe Abb. 4.8) für den Punkt p in $|U| - 1$ Schritten der am weitesten entfernte Punkt q bestimmt. Es stellt sich die Frage, ob sich die Anzahl der Abstandsberechnungen basierend auf dem bis dahin berechneten größten Abstand auch in $TryToPrune$ (und damit in jedem Iterationsschritt) weiter beschränken lässt.

Eine mögliche Modifikation von $TryToPrune$ besteht darin, immer dann Punkte aus der Menge zu entfernen, sobald ein größerer Abstand als δ gefunden wurde. Wird für das Beispiel in Abb. 4.11(a) etwa als erstes der Abstand zwischen den Punkten auf der linken und rechten Kante des Rechtecks berechnet, so können unmittelbar alle Punkte im mittleren Bereich aus der

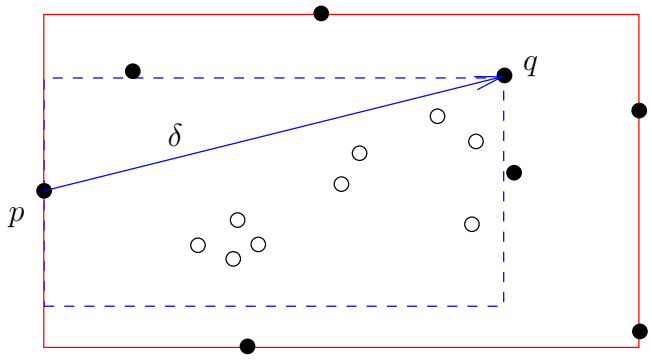


Abb. 4.20: Rechteck, in dem Punkte (dargestellt als offene Kreise) ausgeschlossen werden können

Menge U ausgeschlossen werden, ohne zusätzlich $|U| - 2$ Abstände berechnen zu müssen. Allgemein bietet es sich an, als erstes die Abstände zwischen p und gegenüberliegenden Punkten zu berechnen. Denn diese Punkte haben mit höherer Wahrscheinlichkeit großen Abstand zu p. Unter ungünstigen Umständen kann es passieren, dass der Abstand zwischen p und den übrigen Punkten immer größer wird, ohne dass Punkte aus der Menge entfernt werden. In diesem Fall könnte $TryToPrune$ quadratische Laufzeit annehmen.

Weiter erweist sich folgende Beobachtung als nützlich. Sei der Abstand zwischen einem Punkt p und einem Punkt q der bisher bestimmte größte Abstand δ. Die Strecke zwischen p und q bildet die Hälfte der Diagonale eines achsenparallelen Rechtecks (siehe Abb. 4.20). In diesem Rechteck können sich keine Punkte befinden, die größeren Abstand zu p als δ haben. Die Abstände zwischen p und den Punkten im Inneren des Rechtecks müssen nicht berechnet werden. Bisher ist unklar, ob sich die Menge dieser Punkte so bestimmen lässt, dass die Laufzeit von $TryToPrune$ linear bleibt.

4.5.5 Der Algorithmus FastAML

Die modifzierte Version des AML-Algorithmus, die im Initialschritt auf einen der in diesem Kapitel entwickelten $HighDiameter$-Algorithmen zurückgreift, wird im Folgenden FastAML genannt. Die Berücksichtigung der Häufigkeitsverteilung durch Gewichtung der euklidischen Distanz (siehe Abschnitt 1.2.3) kann bei FastAML nur beim Algorithmus $HighDiameter_{rand}$ (siehe Abb. 4.9) unverändert übernommen werden. Bei einer Sortierung der Punkte müssen diese über ihre Häufigkeiten stattdessen schon vorher transformiert worden sein.

Für eine große Anzahl zu bestimmender Cluster k hat FastAML nach wie vor kubische Lauf-

zeit (siehe Abschnitt 4.2.2). Für zu erwartendes kleines k wird die asymptotische Laufzeit von FastAML jedoch von der Laufzeit des entsprechenden $HighDiameter$-Algorithmus bestimmt. Diese ist — unabhängig von der gewählten Sortierung der Punkte in U und der Durchlaufstrategie — durch $O(dr^2)$ nach oben beschränkt und damit in keinem Fall schlechter als die Berechnung der Distanzmatrix in $\Theta(dr^2)$ Schritten. Nach unten ist die Laufzeit entsprechend dem Algorithmus $HighDiameter_{mid}$ beschränkt durch $\Omega(dr + r \log r)$.

Obwohl eine umfassende Analyse der durchschnittlichen Laufzeit der $HighDiameter$-Algorithmen ausbleibt, ist die Wahrscheinlichkeit für das Eintreten des *worst case* als gering einzuschätzen. Insbesondere bei einer Durchlaufstrategie vom äußeren Rand der Instanzenmenge nach innen ist stattdessen die Wahrscheinlichkeit hoch, große Abstände in frühen Iterationsschritten zu finden und dadurch eine hohe Zahl von Abstandsberechnungen einzusparen. Die empirischen Ergebnisse in Abschnitt 5.2 bestätigen diese Annahme.

Kapitel 5

Empirischer Vergleich der Clusteringverfahren

Der praktische Teil dieser Arbeit führt die Verfahren aus den Kapiteln 1, 2 und 3 und die theoretischen Überlegungen aus Kapitel 4 zusammen. Zunächst wird die Laufzeit von AML, FastAML und Quickhull anhand zufällig erzeugter Datenpunkte verglichen. Danach wird die Qualität der Clusterungen zweier Luftaufnahmen von Erosionsgebieten mit SOM, k-Mittelwert, EM, AML und FastAML beurteilt.

5.1 Implementation der Verfahren

Um die Ergebnisse in diesem Kapitel nachvollziehen zu können, sei an dieser Stelle kurz die Implementation der Verfahren, die Umgebung und der Clustering-Prozess beschrieben.

Für eine bessere Vergleichbarkeit sind alle Clusteringverfahren in der gleichen Programmiersprache JAVA implementiert. Die Verfahren k-Mittelwert, EM und CLASSIT stammen aus der *Weka-Bibliothek* der Universität von Waikato, Neuseeland [WF01]. AML, FastAML, WTAN, SOM und eine *QSOM* (*QuickSOM*) genannte Variante der SOM sind eigens für diese Diplomarbeit implementiert. Für den Vergleich von FastAML und Quickhull wird eine Implementation des Quickhull-Algorithmus in C verwendet. Der Quellcode findet sich im WWW (World Wide Web) unter `http://www.geom.umn.edu/software/qhull/`.

Der Unterschied zwischen QSOM und SOM liegt in einer an die Erosionsproblematik angepassten Nachbarschaftsfunktion. Dadurch benötigt die QSOM weniger Durchläufe, um ähnliche Ergebnisse wie die SOM zu erzielen. Auch die Nachbarschaftsfunktion des WTANs ist

leicht modifiziert, um zu verhindern, dass Gewichte übereinander geschoben werden. Die Clusterungen von WTAN, QSOM und SOM weisen bereits in den Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02] kaum Unterschiede bezüglich der Qualität auf. Dieses Ergebnis wird in den Abschnitten 5.3.2 und 5.3.3 bestätigt. Auf eine ausführliche Darstellung der Nachbarschaftsfunktionen und Lernparameter im Unterschied zu der in Abschnitt 1.13 beschriebenen SOM wird daher verzichtet. Eine detaillierte Darstellung findet sich in den Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02].

Für den Laufzeitvergleich zwischen AML, FastAML und Quickhull kommt ein Rechner mit einer Intel Pentium III Mobile CPU (1,1 GHz) und 256 MB Hauptspeicher zum Einsatz. Das Clustering wurde auf einem Rechner mit einer AMD Athlon XP 2000+ CPU (1,67 GHz) und 512 MB Hauptspeicher durchgeführt. Verwendet wird das Java Runtime Environment (JRE) von SUN in der Version 1.4.02 unter dem Betriebssystem Microsoft Windows XP Professional.

Eine Luftaufnahme durchläuft den folgenden Clustering-Prozess:

1. Die RGB-Vektoren aus der Bilddatei werden wie in Abschnitt 2.5 beschrieben skaliert und in Merkmalsvektoren mit Attributwerten aus dem Intervall $[0, 1]$ umgewandelt. Das Ergebnis ist eine Textdatei im ARFF-Format (für eine Beschreibung siehe [WF01]). Die Zuordnung der Trainingsinstanzen (Farben) zu Pixeln in der ursprünglichen Luftaufnahme bleibt in der ARFF-Datei durch die Reihenfolge implizit erhalten.

2. Wird mit den Verfahren WTAN, SOM und QSOM geclustert, so müssen die Häufigkeiten der Instanzen transformiert werden. Dadurch wird eine Überanpassung der neuronalen Netze an die Daten verhindert. Die Häufigkeiten seltener Elemente werden dazu etwas erhöht, die häufiger Elemente abgesenkt. Für eine detaillierte Beschreibung der Transformation siehe [Zer01] und [Tsc02].

3. Die Trainingsinstanzen werden von einem der Clusteringverfahren in die spezifizierte Anzahl von k Klassen eingeteilt. Das gewonnene Modell wird für den nächsten Schritt gespeichert.

4. Eine Segmentierung der Trainingsinstanzen erfolgt anhand des Modells durch Zuordnung zu einem Cluster (Klassifikation). Der Merkmalsvektor einer so klassifizierten Instanz wird durch das Zentrum des Clusters ersetzt, dem sie angehört. Dadurch wird die Anzahl der unterschiedlichen Instanzen (Farben) auf die spezifizierte Anzahl der Cluster reduziert. Ergebnis ist eine ARFF-Datei mit den neuen Merkmalsvektoren.

5. Die Farben der Pixel in der ursprünglichen Luftaufnahme werden durch die neuen Merkmalsvektoren (Farben) aus der ARFF-Datei ersetzt und im PNG-Format gespeichert.

6. Aus Kenngrößen wie der Größe des Bildes, der Anzahl unterschiedlicher Farben, SST, SSW, SSB, MMD, GMMD und der Laufzeit wird eine HTML-Datei generiert, welche die Ergebnisse eines Durchlaufs (d.h. einer Clusterung) zusammenfasst. Dazu gehört auch die Erstellung einer Grafik, die das Modell (sofern verfügbar) und die Clusterzentren im Merkmalsraum (also dem RGB-Farbwürfel, siehe Abschnitt 2.3) zeigt.

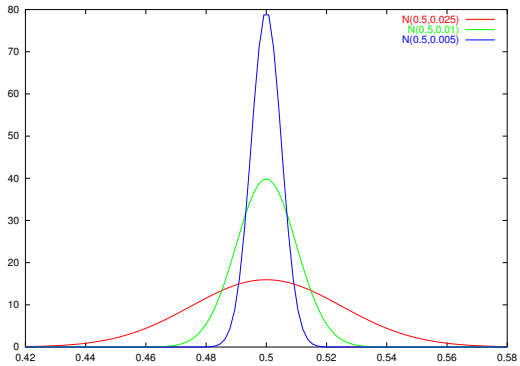
Für jeden Durchlauf wird anhand der spezifizierten Parameter ein separates Verzeichnis mit den Ergebnissen generiert. Die daraus entstehende Verzeichnisstruktur kann z. B. über einen Web-Server im WWW oder in einem Intranet veröffentlicht werden. Die Ergebnisdateien lassen sich mit einem Web-Browser anzeigen, der Cascading-Style-Sheets (CSS), JavaScript und das PNG-Format unterstützt. Die Luftaufnahmen und die Ergebnisse der für diese Diplomarbeit durchgeführten Clusterungen finden sich auch im WWW unter der Adresse `http://www.picana.de/stolpe/`. Es sei auf diese Quelle verwiesen, da die in den folgenden Abschnitten dargestellten Luftaufnahmen gegenüber den Originalen aus drucktechnischen Gründen an Qualität verlieren.

5.2 Die Laufzeiten von FastAML, AML und Quickhull

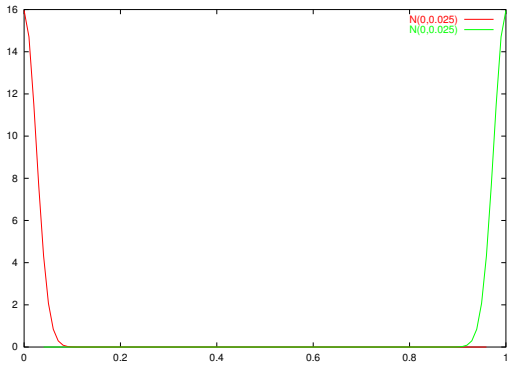
Es wird überprüft, inwieweit sich die in den Abschnitten 4.5.2, 4.5.3 und 4.5.5 getroffenen Annahmen bezüglich der Laufzeit von FastAML empirisch bestätigen lassen. Dazu werden verschiedene Varianten des Algorithmus, die sich aus der Art des verwendeten $HighDiameter$-Algorithmus ($rand$, $lexi$, $comp$ und mid) ergeben, mit dem AML-Algorithmus bezüglich der Laufzeit verglichen. Für den Vergleich reicht es aus, die für die Lösung des Farthest-Pair-Problems benötigte Laufzeit zu betrachten.

Es werden zufällig dreidimensionale Merkmalsvektoren mit Attributwerten aus einer Gleichverteilung, einer Gauss'schen Normalverteilung $N(\mu, \sigma)$ mit $\mu = 0.5$ und unterschiedlichem σ (siehe Abb. 5.1(a)) sowie einer Mischung zweier Normalverteilungen $N_1(\mu_1, \sigma), N_2(\mu_2, \sigma)$ mit $\mu_1 = 0, \mu_2 = 1$ und $\sigma = 0.025$ (siehe Abb. 5.1(b)) über dem Intervall $[0, 1]$ erzeugt.

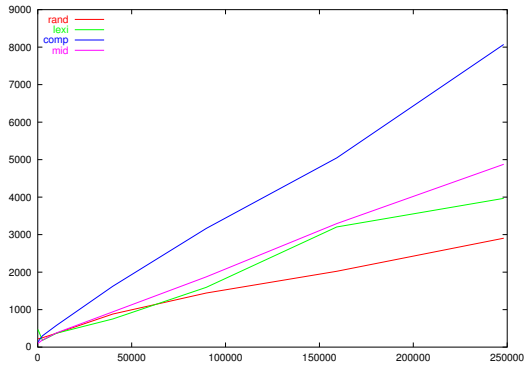
Die Verwendung unterschiedlicher Verteilungen dient dazu, die Varianten von FastAML auf ihre Schwächen bezüglich bestimmter Arten von Eingabemengen zu testen. Die Auswahl fiel auf die genannten Verteilungen, da sich für eine Konzentration aller Datenpunkte in der



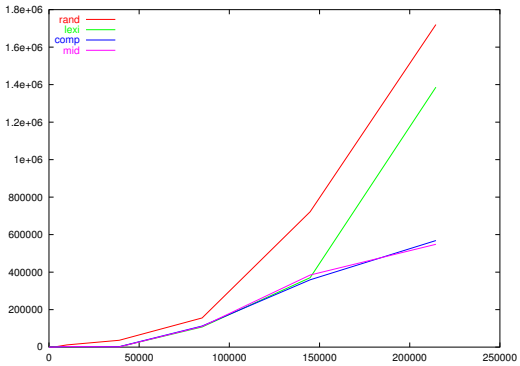
(a) $N(0.5, \sigma)$ für $\sigma \in \{0.025, 0.01, 0.005\}$



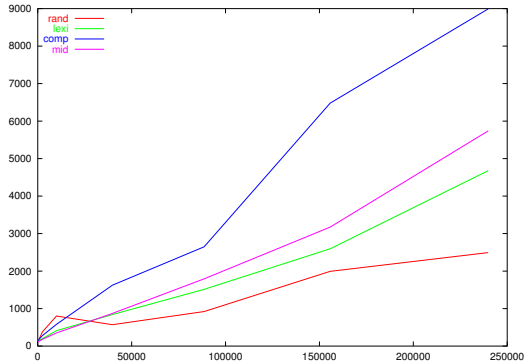
(b) Mischung aus $N(0, 0.025)$ und $N(1, 0.025)$



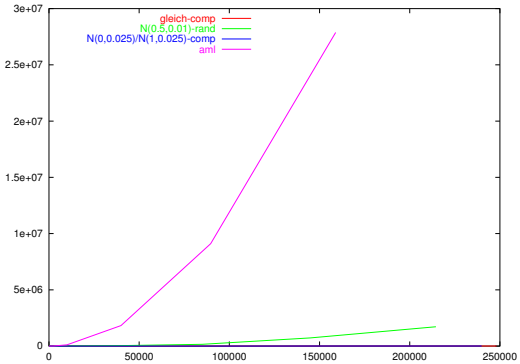
(c) Laufzeiten der Varianten von FastAML für eine Gleichverteilung



(d) Laufzeiten der Varianten von FastAML für $N(0.5, 0.01)$



(e) Laufzeiten der Varianten von FastAML für eine Mischung aus $N(0, 0.025)$ und $N(1, 0.025)$



(f) Laufzeiten der schlechtesten Durchläufe von FastAML und im Vergleich zur Laufzeit von AML

Abb. 5.1: Grafische Darstellung der Laufzeiten der Verfahren für verschiedene Verteilungen.

Mitte des Einheitswürfels ($N(0.5, \sigma)$) für $HighDiameter_{rand}$ unter Umständen der *worst case* ergeben kann (siehe Abb. 4.11(b)). Bei einer Konzentration aller Datenpunkte an den Rändern und Ecken des Einheitswürfels (Mischung aus $N(0, 0.025)$ und $N(1, 0.025)$) könnte sich bei ungünstiger Anordnung der *worst case* für $HighDiameter_{comp}$ (siehe Abb. 4.17(b)) und für $HighDiameter_{mid}$ (siehe Abb. 4.18(b)) ergeben.

Für unterschiedliche Anzahlen von Datenpunkten wird eine Ziehung aus den angegebenen Verteilungen durchgeführt. Eine einzelne Ziehung von Zufallszahlen ist zwar im statistischen Sinne nicht repräsentativ, im Zusammenhang mit den theoretischen Ausführungen in den Abschnitten 4.5.2 und 4.5.3 ergibt die Auswertung dennoch ein recht differenziertes Bild für die unterschiedlichen Varianten.

Betrachte dazu die Zeitangaben in Abb. 5.2 für die unterschiedlichen Verteilungen und Varianten des FastAML-Algorithmus. Der Gewinner ist für eine bestimmte Anzahl von Datenpunkten hell markiert. Die zu berücksichtigende Anzahl $|U|$ von Datenpunkten ergibt sich aus der Menge der unterschiedlichen Instanzen in S (d.h., einige Datenpunkte werden mehrfach erzeugt). In der Spalte „übrig aus $rand$“ ist exemplarisch die Anzahl der Datenpunkte eingetragen, die durch $HighDiameter_{rand}$ nicht ausgeschlossen werden können.

Bei einer Gleichverteilung der Datenpunkte schneidet $HighDiameter_{rand}$ ab einer hohen Anzahl von Datenpunkten mehrfach gut ab. Die Laufzeiten für die verschiedenen Varianten sind durch interpolierte Kurven in Abb. 5.1(c) dargestellt. Es wird deutlich, dass alle Kurven im Intervall $[0, 250000]$ einen mehr oder weniger linearen Verlauf haben. Die Laufzeiten liegen alle im Bereich von wenigen Sekunden (oder gar Millisekunden).

Für die Normalverteilung mit $\sigma = 0.025$ gelten ähnliche Aussagen wie bei einer Gleichverteilung, obwohl sich eine Sortierung der Punkte im Unterschied zur zufälligen Durchlaufstrategie als etwas besser erweist. Ein großer Sprung in der Laufzeit ergibt sich dann — wie zu erwarten — für die beiden Verteilungen $N(0, 0.01)$ und $N(0, 0.005)$, bei denen sich die Datenpunkte stärker in der Mitte des Einheitswürfels konzentrieren. Es sei darauf hingewiesen, dass die Laufzeiten im Hinblick auf die hohe Anzahl von Datenpunkten und den quadratischen *worst case* noch immer als sehr gut zu bezeichnen sind. So ergibt die schlechteste Laufzeit von $HighDiameter_{rand}$ bei $\sigma = 0.01$ für 214493 Datenpunkte in etwa 28 Min., die beste Laufzeit von $HighDiameter_{comp}$ in etwa 9 Min. Aus Abb. 5.1(d) ergibt sich, dass eine Sortierung der Punkte und der nachfolgende Durchlauf vom Rand der Punktmenge zur Mitte ($comp$ und mid) wesentlich bessere Laufzeiten als die zufällige Auswahl von Punkten ($rand$) oder die Sortierung nach nur einer Komponente ($lexi$) liefert. Diese Beobachtung stützt die theoretischen

$\|S\|$	$\|U\|$	übrig aus $rand$	Laufzeit Farthest-Pair in ms				
			$rand$	$lexi$	$comp$	mid	AML
100	100	0					130
2500	2500	0					5759
10000	9998	0					111280
40000	39935	0					1823993
90000	89652	0					9101297
160000	158934	0					27878207
250000	247537	0					$>$ 11 Std.
gleichverteilt über dem Intervall $[0, 1]$							
100	100	37	220	470	140	100	
2500	2500	28	251	181	301	191	
10000	9997	45	371	371	581	381	
40000	39947	25	881	751	1623	941	
90000	89749	38	1442	1593	3165	1873	
160000	159261	35	2022	3205	5047	3295	
250000	248179	60	2904	3966	8071	4877	
$N(0.5, 0.025)$ über dem Intervall $[0, 1]$							
100	100	42	151	90	140	120	
2500	2500	733	350	200	291	230	
10000	9983	686	1001	420	491	341	
40000	39678	1106	1362	922	1823	931	
90000	88616	976	1853	1823	3104	1893	
160000	155838	1594	4796	2884	5017	3215	
250000	239671	1390	8091	5958	9373	5898	
$N(0.5, 0.01)$ über dem Intervall $[0, 1]$							
100	100	35	140	80	121	100	
2500	2496	242	170	221	250	180	
10000	9913	5244	11587	2694	2854	2925	
40000	38945	4683	36542	1582	2173	1282	
90000	84907	17531	155704	107414	112571	110238	
160000	144788	35438	721989	370262	358907	384503	
250000	214493	46058	1720424	1386294	568207	548319	
$N(0.5, 0.005)$ über dem Intervall $[0, 1]$							
100	100	31	150	120	151	110	
250	2491	430	220	170	301	191	
100	9810	942	1652	281	401	351	
40000	37136	9018	52967	16754	14481	13710	
90000	76989	16231	153711	66255	74217	70361	
160000	123373	27259	525225	215701	198616	199456	
250000	171301	25883	567577	106484	111140	102817	
Mischung aus $N(0, 0.025)$ und $N(1, 0.025)$ über dem Intervall $[0, 1]$							
100	100	36	120	140	150	110	
2500	2499	55	381	200	280	180	
10000	9986	52	801	411	581	351	
40000	39731	34	571	841	1622	871	
90000	88595	16	922	1513	2684	1793	
160000	155739	18	1993	2594	6480	3174	
250000	239850	19	2494	4677	8992	5739	

Abb. 5.2: Laufzeiten der verschiedenen Varianten von FastAML und dem AML-Algorithmus für zufällig erzeugte Datenpunkte aus den angegebenen Verteilungen.

Aussagen in Abschnitt 4.5.2, die den *worst case* im Falle einer Konzentration der Datenpunkte in der Mitte beim Algorithmus $HighDiameter_{rand}$ sehen.

Bei einer Konzentration der Datenpunkte an den Rändern und Ecken des Einheitswürfels drehen sich die Verhältnisse — wie ebenfalls zu erwarten — um (siehe dazu Abb. 5.1(e)). Die Durchlaufstrategie vom Rand zur Mitte ($comp$ und mid) liefert schlechtere Ergebnisse als die rein zufällige Auswahl der Datenpunkte ($rand$) oder die Sortierung nach nur einer Komponente ($lexi$). Die Beobachtung stützt die theoretischen Aussagen aus Abschnitt 4.5.3.

Bezüglich der Durchlaufstrategie vom Rand zur Mitte ist anzumerken, dass der Algorithmus $HighDiameter_{mid}$ meist etwas besser abschneidet als die unabhängige Sortierung der Komponenten. Das kann damit zusammenhängen, dass die Verwaltung der Indextabelle während des Durchlaufs Zeit kostet.

Insgesamt lässt sich feststellen, dass sich die Entwicklung verschiedener Durchlaufstrategien in Abschnitt 4.5.2 für die Praxis als sinnvoll erweist. Ist die Verteilung der Datenpunkte im Voraus bekannt, so kann die Durchlaufstrategie entsprechend angepasst werden.

Für AML spielt es keine Rolle, aus welcher Verteilung die Datenpunkte gezogen werden. Die Abb. 5.1(f) zeigt einen Vergleich der Laufzeit des AML-Algorithmus für die jeweiligen „Verlierer“ der zuvor betrachteten Durchläufe. Selbst für die schlechtesten Durchläufe übersteigt die Laufzeit des AML-Algorithmus die Laufzeiten von FastAML bei weitem. Für 158934 Datenpunkte benötigt AML über 7 Std., während FastAML für 214493 im schlechtesten Durchlauf die Berechnung in nur 28 Min. abschließt. Die Berechnung mit 247537 Datenpunkten wurde für AML manuell nach 11 Std. abgebrochen.

Der Vergleich der Laufzeiten von FastAML mit einer Implementation von Quickhull zeigt, dass Quickhull in höheren Dimensionen nicht mehr effizient ist. So musste etwa die Berechnung der konvexen Hülle in Dimensionen ≥ 6 bei 10000 gleichverteilt erzeugten Datenpunkten abgebrochen werden, da bereits Auslagerungsspeicher für die Berechnungen genutzt wurde. Der verwendete Rechner ist mit 256 MB Hauptspeicher ausgestattet. Die Lösung des Farthest-Pair-Problems mit FastAML führt dagegen auch bei 100000 gleichverteilt erzeugten Datenpunkten in der Dimension 9 zu einer Laufzeit von 93 Sekunden.

5.3 Vergleich anhand qualitativer Gütekriterien

Die Arbeit folgt den drei in [Zer01] und [Tsc02] verwendeten Methoden, die Clusterung von Luftaufnahmen unter qualitativen Gesichtspunkten im Sinne der Erosionsproblematik zu bewerten.

Ein unmittelbarer Eindruck der Clusterung ergibt sich aus der *Ansicht der segmentierten Luftaufnahme* nach Augenmaß. So lässt sich z. B. schnell erkennen, ob seltene Elemente identifiziert und Farbflächen gebildet werden und wie scharf die Übergänge zwischen diesen Flächen sind. Eine andere Methode der Bewertung orientiert sich an den berechneten *numerischen Kenngrößen* für die Qualität einer Clusterung (siehe Abschnitt 1.1.5). Äußerst aufschlussreich ist zudem die anschauliche *Darstellung der Clusterzentren* und — sofern verfügbar — des zu Grunde liegenden Modells im Merkmalsraum. Diese Darstellung wird durch die niedrige Dimensionalität der Merkmalsvektoren ermöglicht. In anderen Anwendungsgebieten lassen sich auf diese Weise nur ausgewählte Komponenten darstellen.

Auf die Darstellung der SOMs in U-Matrix-Form (siehe [Zer01]) wird verzichtet. Die topologische Anordnung der Gewichtsvektoren auf dem zweidimensionalen Gitter ist ein Maß für die Qualität der antrainierten Merkmalskarte, nicht für die Qualität der resultierenden Einteilung in Klassen.

Das in Abschnitt 1.4.2 beschriebene Clusteringverfahren CLASSIT wird für den empirischen Vergleich anhand qualitativer Gütekriterien nicht herangezogen. Mehrere Versuche ergeben, dass sich die Implementation aus der Weka-Bibliothek [WF01] für Bilder der im Folgenden dargestellten Größenordnung in der Praxis nicht eignet. Die Ressourcen der in Abschnitt 5.1 aufgeführten Rechner reichen für das Clustering nicht aus.

5.3.1 Auswahl der Beispiele

Die beiden für das Clustering verwendeten Luftaufnahmen sind in Abb. 5.3(a) und Abb. 5.3(b) gezeigt. Die Anzahl der Farben ist mit 10457 (für Abb. 5.3(a)) und 36067 (für Abb. 5.3(b)) deutlich geringer als die Anzahl von $768 \times 512 = 393216$ Pixeln. Bezüglich der Laufzeit verschafft dieser Sachverhalt Algorithmen wie AML oder FastAML einen gewissen Vorteil gegenüber den anderen Verfahren (siehe Abschnitt 4.2.3).

Die geclusterten Luftaufnahmen werden im Folgenden auf drei Merkmale hin untersucht,

(a) Luftaufnahme mit 10573 verschiedenen Farben bei 393216 Pixeln. Aufgenommen im Rahmen des SFB 308 „Adapted Farming in West Africa“, Region Chikal, Niger.

(b) Luftaufnahme aus einem Gebiet des Oman mit 36067 Farben bei 393216 Pixeln.

Abb. 5.3: Luftaufnahmen von zwei Erosionsgebieten

die für die Erkennung von Erosion wichtig sind: Bildung zusammenhängender Farbflächen (Merkmal 1), hohe Heterogenität der Klassen (Merkmal 2) und Erkennung seltener Elemente (Merkmal 3). Zudem folgt eine Anmerkung zu den Laufzeiten.

5.3.2 Clusterungen der Luftaufnahme in Abb. 5.3(a)

Für eine bessere Vergleichbarkeit der Clusterungen wurde die Anzahl zu bestimmender Cluster bei alle verwendeten Verfahren übereinstimmend gewählt. Im Rahmen dieser Arbeit ist die Zahl zu bestimmender Cluster durch eine schnelle Clusterung mit dem FastAML-Algorithmus und nachfolgender Auswahl der besten Clusterung bestimmt worden. Als „beste“ Clusterung gelte in diesem Zusammenhang eine Segmentierung der Luftaufnahme dergestalt, dass die wesentlichen Strukturen bei kleinstmöglicher Klassenzahl erhalten bleiben. Eine Auswahl nach Augenmaß des Autors ergibt für die Luftaufnahme in Abb. 5.3(a) 21 zu bestimmende Cluster.

Ansicht der segmentierten Luftaufnahme

Betrachtet seien Clusterungen der Luftaufnahme in Abb. 5.3(a) mit EM (siehe Abb. 5.4), QSOM (siehe Abb. 5.5) und FastAML (siehe Abb. 5.6). Die Clusterungen der Aufnahme mit anderen Verfahren sind nicht dargestellt, da sie der Clusterung mit dem EM-Algorithmus sehr nahe kommen und kaum Unterschiede aufweisen.

Es kann festgestellt werden, dass alle dargestellten Clusterungen die wesentlichen Strukturen des Ursprungsbildes erhalten. Dabei beträgt die Anzahl der Farben in den geclusterten Bildern nur 21 Farben im Vergleich zu 10457 Farben in der Originalaufnahme (das entspricht in etwa einer Reduktion um den Faktor 500). Die Farbwerte der Farben im Sinne des HSI-Modells (siehe Abschnitt 2.3) bleiben für diese Aufnahme bei allen Verfahren nur teilweise erhalten. So finden sich etwa ursprünglich grüne Flächen in der geclusterten Luftaufnahme nicht als grün eingefärbte Flächen wieder.

Im Gegensatz zu der QSOM bildet der EM-Algorithmus in der linken oberen Ecke des Bildes eine zusammenhängende Fläche gleicher Farbe (Merkmal 1). Der Abstand zwischen den Farbklassen ist gering, d.h. die Farbgebung unterscheidet sich nur geringfügig (Merkmal 2). Das lässt sich daran erkennen, dass die Übergänge — außer beim Übergang von dunklen zu hellen Farben — zwischen den Farbflächen fließend sind und damit der Originalaufnahme sehr nahe kommen. Eine Unterscheidung der Farbflächen mit bloßem Auge ist so schwer möglich. Der

Abb. 5.4: Einteilung der Luftaufnahme aus Abb. 5.3(a) in 21 Cluster mit EM und einer maximalen Iterationstiefe von 500

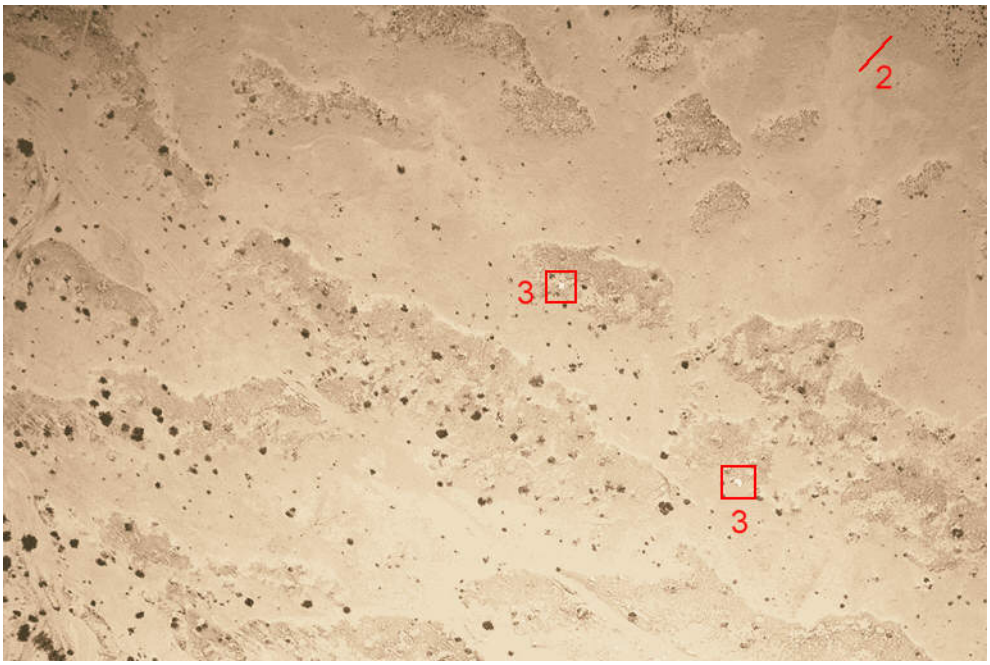


Abb. 5.5: Einteilung der Luftaufnahme aus Abb. 5.3(a) in 21 Cluster mit einer QSOM bei 10 Lernzyklen

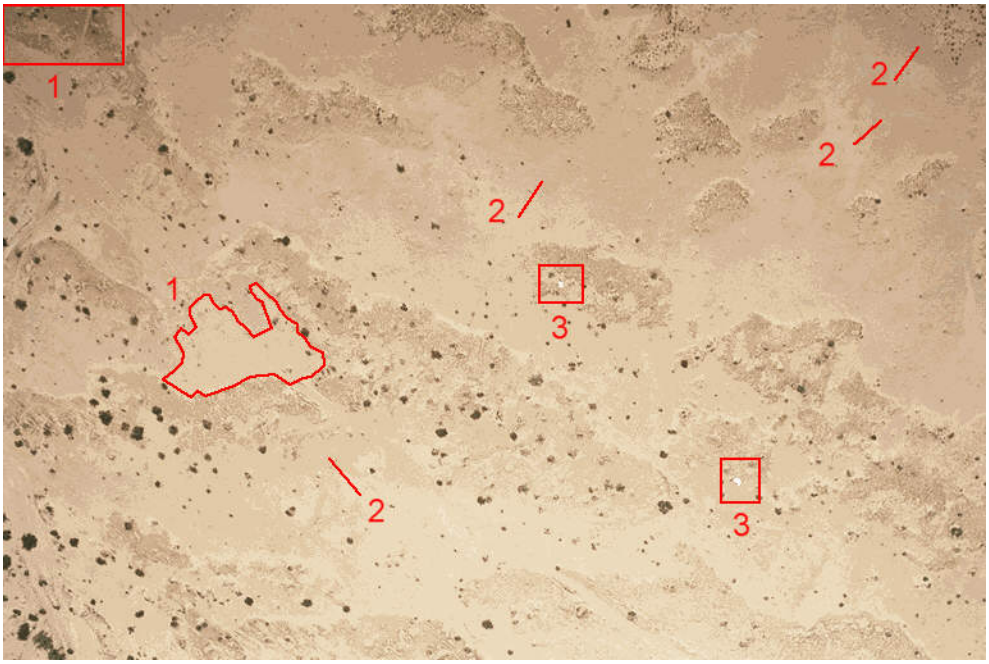


Abb. 5.6: Einteilung der Luftaufnahme aus Abb. 5.3(a) in 21 Cluster mit FastAML und $\rho = 0.5$

Algorithmus erkennt keine seltenen Elemente (Merkmal 3).

Die Clusterung mittels QSOM kommt dem Originalbild am nächsten. Es werden kaum zusammenhängende Farbflächen gebildet (Merkmal 1) und die Übergänge der Flächen sind fließend (Merkmal 2). Seltene Elemente bleiben allerdings erhalten (Merkmal 3). Im Sinne der Reduktion von Information bei gleichzeitiger Bewahrung der Struktur liefert die QSOM die beste Clusterung. Aufgrund der großen Ähnlichkeit zur Originalaufnahme bietet sie dem Substanzwissenschaftler jedoch kaum neue Möglichkeiten der Interpretation.

Die mit dem FastAML-Algorithmus erzeugte Clusterung erfüllt hingegen alle Forderungen, die von Zerbst [Zer01] und Tschiersch [Tsc02] im Zusammenhang mit der Erosionsproblematik aufgestellt wurden. Es werden teilweise große, zusammenhängende Flächen gleicher Farbe gebildet (Merkmal 1). Die Übergänge zwischen den Farbflächen sind abrupter als in der Originalaufnahme, was auf eine höhere Heterogenität der gebildeten Cluster im Vergleich zu den anderen Verfahren schließen lässt (Merkmal 2). Die beiden seltenen Elemente werden ebenfalls erkannt (Merkmal 3).

In Bezug auf die Erosionsproblematik kann die mit FastAML erzeugte Clusterung rein op-

tisch als die beste identifiziert werden.

Betrachtung der berechneten Kenngrößen

In Abb. 5.7 sind die Kenngrößen verschiedener Clusterungen der Luftaufnahme in Abb. 5.3(a) für eine Einteilung in 21 Cluster aufgeführt. Die absoluten „Gewinner“ bezüglich der jeweiligen Größe sind dunkel markiert, die relativen Gewinner für eine Gruppe von Clusterungen eines Verfahrens hell.

Aus der Tabelle wird ersichtlich, warum EM, QSOM und FastAML mit den jeweiligen Parametern examplarisch für den optischen Vergleich ausgewählt worden sind. Betrachtet seien die Werte der $GMMD$ (mit $a = 0.1$, $a = 0.5$ und $a = 0.9$) für die jeweiligen Gruppengewinner. Für die genannten Verfahren ist der Unterschied zwischen den Werten am höchsten. Dagegen liegen die Werte der Gruppengewinner von WTAN, SOM und k-Mittelwert (SKM, Simple-K-Means) sehr nahe an den Werten des Gruppengewinners der QSOM. Tatsächlich sehen sich die durch diese Verfahren segmentierten Bilder auch rein optisch sehr ähnlich (auf eine Darstellung wird verzichtet, die Bilder finden sich jedoch im WWW unter der in Abschnitt 5.1 angegebenen Adresse).

Es fällt auf, dass die Werte der MMD und $GMMD$ für das Verfahren FastAML ($\rho = 0.5$) am höchsten sind. Dabei sind die numerischen Unterschiede der $GMMD$ zu den Clusterungen der anderen Verfahren wesentlich größer als die Unterschiede der MMD. Das Ergebnis stimmt mit der rein optischen Beurteilung aus Abschnitt 5.3.2 überein, in dem die Clusterung durch FastAML als die für die Erosionsproblematik beste identifiziert wird. Den niedrigsten Wert aller höchsten Werte für die $GMMD$ hat die Clusterung durch den EM-Algorithmus.

Die Tabelle zeigt auch, dass eine Bewertung der Clusterungen im Sinne einer niedrigen Intra-Class-Varianz für die vorliegende Anwendung problematisch ist. Die in 2000 Zyklen antrainierte SOM weist den kleinsten Wert von 0.000625 für die SSW auf, dicht gefolgt von der in 26 Zyklen antrainierten QSOM, k-Mittelwert (SKM) und dem in 4 Zyklen antrainierten WTAN. Gemäß einer Bewertung durch herkömmliche Qualitätsmaße wie der SSW wären die Clusterungen dieser Verfahren somit als beste bewertet worden, obwohl sie die im Rahmen der Anwendung gestellten Anforderungen nicht erfüllen. Die SSW der mit FastAML ($\rho = 0.5$) erzeugten Clusterungen ist mit 0.00156 deutlich größer. Im Verhältnis zur Gesamtvarianz ($SSW + SSB = 0.030549$) bzw. zur SSB ist der Wert allerdings noch immer klein. Die SSB unterscheidet sich für alle Verfahren nur geringfügig.

Verfahren	Parameter	SSW	SSB	MMD	GMMD			Laufzeit
					$a = 0.1$	$a = 0.5$	$a = 0.9$	in ms
FastAML	$\rho = 0$	0.001562	0.028987	0.075402	0.179206	0.183911	0.186536	1632
FastAML	$\rho = 0.1$	0.001562	0.028987	0.075402	0.179206	0.183911	0.186536	1653
FastAML	$\rho = 0.25$	0.001562	0.028987	0.075412	0.179284	0.183997	0.186626	1662
FastAML	$\rho = 0.5$	0.00156	0.028989	0.075437	0.179506	0.184192	0.186806	1672
FastAML	$\rho = 0.75$	0.00156	0.028989	0.075432	0.179365	0.184051	0.186664	1652
FastAML	$\rho = 0.9$	0.001562	0.028987	0.075402	0.179206	0.183911	0.186536	1653
FastAML	$\rho = 0.999$	0.001062	0.029487	0.070396	0.093665	0.104673	0.110814	1652
FastAML	$\rho = 0.9999$	0.002159	0.02839	0.052785	0.045436	0.062322	0.07174	1653
FastAML	$\rho = 1$	0.001357	0.029192	0.054153	0.047852	0.063455	0.072159	1642
AML	$\rho = 0.5$	0.001562	0.028987	0.075402	0.179206	0.183911	0.186536	76470
EM	iter $= 100$	0.001224	0.029325	0.046522	0.053802	0.065161	0.071496	1780039
EM	iter $= 250$	0.000999	0.02955	0.048875	0.058869	0.069392	0.075261	4237934
EM	iter $= 500$	0.000999	0.02955	0.048875	0.058869	0.069392	0.075261	4231965
SKM		0.000642	0.029907	0.058275	0.076844	0.089252	0.096174	713126
WTAN	$c = 2$	0.000692	0.029857	0.051367	0.063693	0.075337	0.081832	20600
WTAN	$c = 3$	0.000694	0.029855	0.050455	0.060597	0.072352	0.078909	30083
WTAN	$c = 4$	0.000667	0.029882	0.052591	0.067592	0.079023	0.085399	39517
WTAN	$c = 5$	0.000664	0.029885	0.052211	0.067028	0.078443	0.08481	48870
WTAN	$c = 6$	0.000668	0.029881	0.052529	0.067062	0.078217	0.08444	58344
WTAN	$c = 7$	0.00067	0.029879	0.05152	0.06424	0.075767	0.082197	68108
WTAN	$c = 8$	0.000677	0.029872	0.051754	0.064252	0.075465	0.08172	77271
WTAN	$c = 9$	0.000679	0.02987	0.051728	0.063758	0.074882	0.081088	87036
QSOM	$c = 5$	0.000668	0.029881	0.058586	0.075651	0.086908	0.093188	46387
QSOM	$c = 7$	0.000664	0.029884	0.059732	0.07877	0.089608	0.095654	63802
QSOM	$c = 10$	0.000669	0.02988	0.058835	0.080363	0.090811	0.096639	90781
QSOM	$c = 20$	0.00066	0.029889	0.056218	0.075062	0.085334	0.091065	179268
QSOM	$c = 26$	0.000643	0.029906	0.054347	0.072504	0.08378	0.090069	233717
QSOM	$c = 30$	0.000652	0.029896	0.053202	0.068893	0.08049	0.086958	268446
SOM	$c = 10$	0.005021	0.025528	0.029354	0.042908	0.049381	0.052991	80305
SOM	$c = 25$	0.000771	0.029778	0.0502	0.054903	0.066829	0.073482	198896
SOM	$c = 50$	0.000755	0.029794	0.051835	0.064525	0.075526	0.081662	397312
SOM	$c = 75$	0.000714	0.029835	0.051091	0.065428	0.076042	0.081963	595887
SOM	$c = 100$	0.000743	0.029806	0.052155	0.064278	0.075481	0.08173	792079
SOM	$c = 200$	0.000663	0.029886	0.052481	0.068314	0.078937	0.084862	1584859
SOM	$c = 500$	0.000658	0.02989	0.052821	0.067711	0.078747	0.084902	3998610
SOM	$c = 700$	0.000658	0.029891	0.05288	0.067873	0.078742	0.084804	5542841
SOM	$c = 1000$	0.000649	0.0299	0.053407	0.069684	0.081002	0.087314	8028264
SOM	$c = 1500$	0.000634	0.029915	0.054372	0.07318	0.084345	0.090572	11902735
SOM	$c = 2000$	0.000625	0.029924	0.05495	0.076565	0.087257	0.09322	15860336

Abb. 5.7: Kenngrößen von Clusterungen der Luftaufnahme in Abb. 5.3(a) für unterschiedliche Verfahren und Parametersätze. Es wurde eine Einteilung in 21 Cluster vorgenommen.

Ein größerer Wert der SSW ist aufgrund der Konstruktion des AML-Algorithmus zu erwarten. Das k-Mittelwert-Verfahren führt mehr Mittelungen durch und nähert sich auf diese Weise den Clusterzentren. Da eine enge Verwandtschaft zwischen den WTANs bzw. SOMs und dem k-Mittelwert-Verfahren besteht [Tsy71], gilt diese Aussage auch für die neuronalen Netze. Im Vergleich dazu führt der AML-Algorithmus nur eine einzige Mittelung durch. Das der Klassifikation zu Grunde liegende Modell entspricht bei AML den Startwerten, die von den Clusterzentren abweichen können (siehe dazu auch die Darstellung der Startwerte und Zentren im Merkmalsraum).

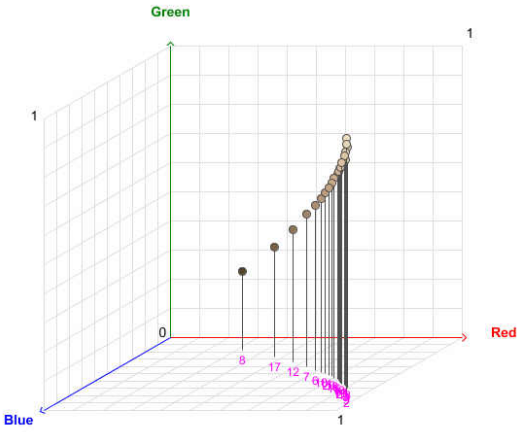
Bezogen auf den Clustering-Prozess (siehe Abschnitt 5.1) unterscheiden sich die Verfahren nur im Schritt der Clusterung. Die Laufzeiten gelten daher für diesen Schritt. FastAML ($\rho = 0.5$) liegt mit 1 Sek. und 672 Millisek. deutlich vorn. Die Laufzeit von AML ist mit etwas über 1 Min. im Vergleich zu den anderen Verfahren noch immer gut, was allerdings auf die geringe Anzahl von Farben in der Luftaufnahme zurückzuführen ist. Gegenüber AML ist die Implementation von FastAML etwa um den Faktor 46 schneller. Die WTANs und QSOMs liegen aufgrund der geringen Anzahl von Trainingszyklen ebenfalls im Minutenbereich. Etwas abgeschlagen mit fast 12 Min. ist das k-Mittelwert-Verfahren. Einen großen Sprung hinsichtlich der Laufzeit gibt es bei der Anwendung von EM (max. 500 Iterationen) und SOM (2000 Zyklen): die beste von diesen Verfahren erzeugte Clusterung wurde in mehreren Stunden erzeugt.

Für die neuronalen Netze wird deutlich, dass die Laufzeit linear mit der Anzahl von Trainingszyklen ansteigt. Für 10457 Farben ist AML den SOMs deutlich überlegen, wie schon von Zerbst [Zer01] und Tschiersch [Tsc02] gezeigt wurde.

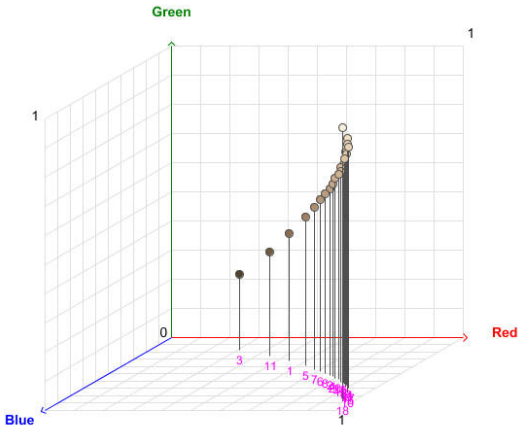
Darstellung der Clusterzentren

Neben der Bewertung von Clusterungen durch Kenngrößen bietet sich für die Beurteilung die Darstellung der Clusterzentren im Merkmalsraum (siehe Abb. 5.8) an, in diesem Fall dem RGB-Farbwürfel (siehe Abschnitt 2.3).

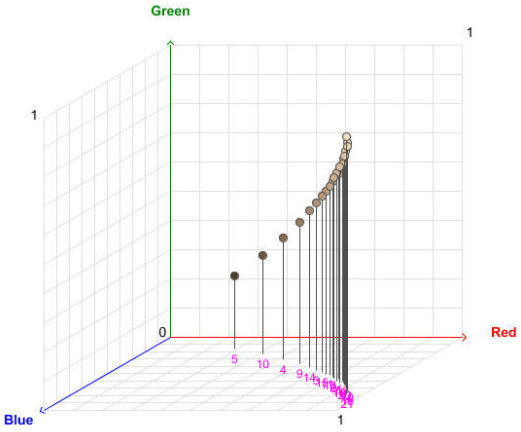
Zunächst fällt wie bei der optischen Betrachtung der geclusterten Luftaufnahme auf, dass sich die Darstellung der Zentren von WTAN, QSOM, SOM, k-Mittelwert und EM erzeugten Clusterungen nur sehr geringfügig unterscheidet. Die Zentren bilden alle eine gleichmäßige (gedachte) Kurve im Merkmalsraum. Insbesondere die Zentren, welche die helleren Brauntöne des Bildes repräsentieren, liegen bei allen Verfahren sehr dicht beieinander. Der geringe Abstand führt zu fließenden Übergängen auf der geclusterten Luftaufnahme, so dass sich keine zusam-



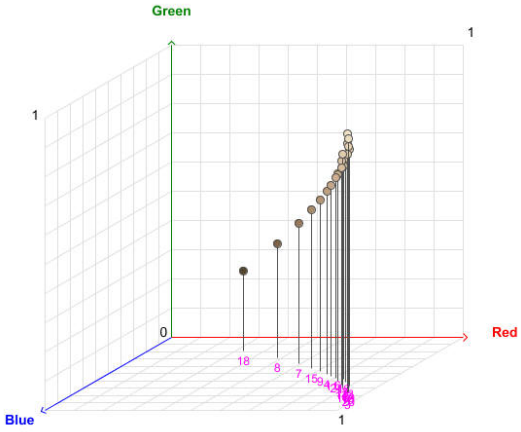
(a) WTAN (4 Zyklen)



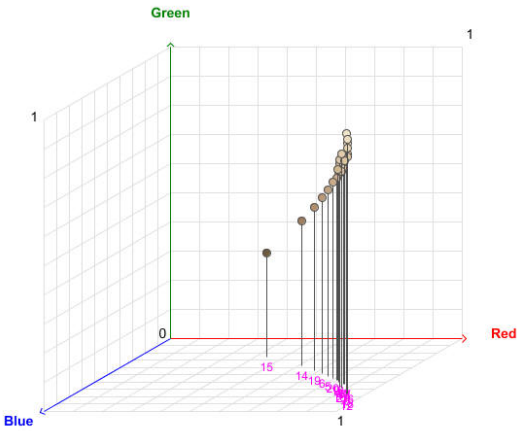
(b) QSOM (10 Zyklen)



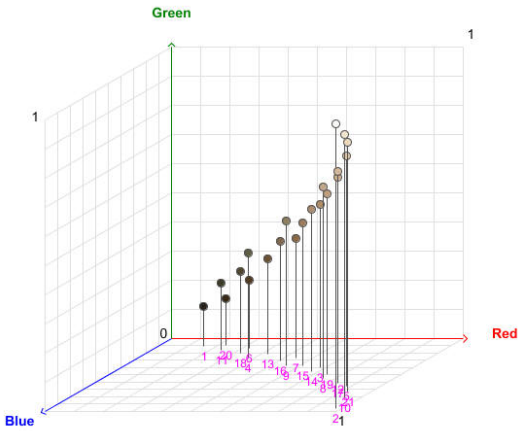
(c) SOM (2000 Zyklen)



(d) k-Mittelwert



(e) EM (max. Iterationstiefe 500)



(f) FastAML ($\rho = 0.5$)

Abb. 5.8: Zentren von Clusterungen der Luftaufnahme aus Abb. 5.3(a) mit den verschiedenen Clusteringverfahren. Die Aufnahme wurde in 21 Cluster eingeteilt (die jeweilige Nummer des Clusters ist teilweise unterhalb des Würfels zu erkennen).

menhängenden Flächen einer Farbe ausbilden können (Merkmal 1). Ein geringer Abstand der Zentren bedeutet auch, dass die Heterogenität der Cluster eher gering ist (Merkmal 2). Das seltene Element mit weißer Farbe findet sich in der Darstellung bei der QSOM ebenfalls wieder: es steht etwas abseits von den anderen Zentren, allerdings nur geringfügig (Merkmal 3).

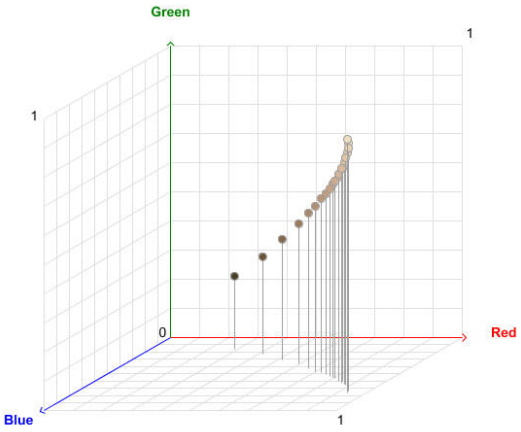
Die Darstellung der von FastAML gebildeten Zentren ergibt ein anderes Bild. Die Zentren decken den Merkmalsraum besser ab und weisen einen höheren Abstand zueinander auf. Die Clusterzentren unterscheiden sich stärker voneinander (Merkmal 2). Sie liegen daher auch weiter von der durch die Zentren der anderen Verfahren gebildeten „Kurve“ entfernt. FastAML erkennt im Gegensatz zu den anderen Verfahren auch die dunklen Brauntöne im Bild, die eine geringere Häufigkeit als die hellen aufweisen (Merkmal 3). Das seltene Element mit weißer Farbe liegt in der Darstellung deutlich abseits von den anderen Zentren und wird insofern noch besser als von der QSOM erkannt (Merkmal 3).

Für eine Demonstration, dass durch die Verfahren k-Mittelwert, Wettbewerbslernen und EM die Clusterzentren sehr gut angenähert werden, sei die Abb. 5.9 betrachtet. In Abb. 5.9(a) sind die Gewichtsvektoren der in 2000 Lernzyklen antrainierten SOM im Merkmalsraum dargestellt. Die Gewichtsvektoren entsprechen dem Modell, welches zur späteren Klassifikation der Instanzen herangezogen wird. Im Vergleich zu den in Abb. 5.9(b) dargestellten Clusterzentren ergibt sich kaum ein sichtbarer Unterschied.

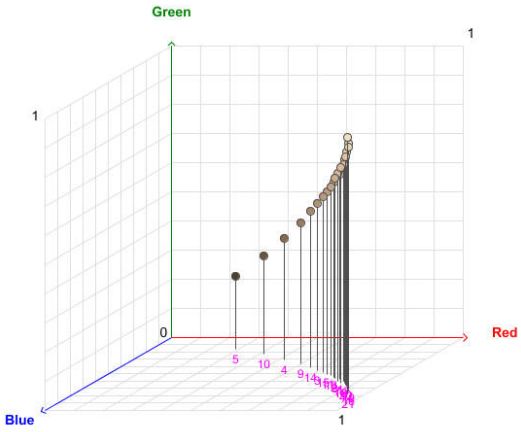
Vergleiche nun den Übergang des durch FastAML gebildeten Modells (den Startwerten) in Abb. 5.9(c) zu den durch Klassifikation gewonnen Clusterzentren in Abb. 5.9(d). Es ergibt sich ein deutlicher Unterschied zwischen dem Modell und den Clusterzentren. Zudem unterscheidet sich das von FastAML gebildete Modell stark von dem der SOM.

Eine Erklärung ergibt sich aus der Arbeitsweise der Verfahren. Die Verfahren k-Mittelwert, Wettbewerbslernen und EM reagieren stark auf die Häufigkeiten der Instanzen in der Trainingsmenge. Dazu müssen die Verfahren in einem oft langwierigen Prozess an die Eingaben angepasst werden. Seltene Elemente verschwinden bei diesem Prozess. Bei FastAML wird dagegen nur eine Mittelwertbildung im Klassifikationsschritt durchgeführt. Die Startwerte sind mit großem Abstand so gewählt, dass sie den Merkmalsraum besser abgedecken. Aus Abb. 5.9(c) und Abb. 5.9(d) wird ersichtlich, wie stark sich die Startwerte durch eine einzige Mittelwertbildung in Richtung der Datenschwerpunkte verschieben.

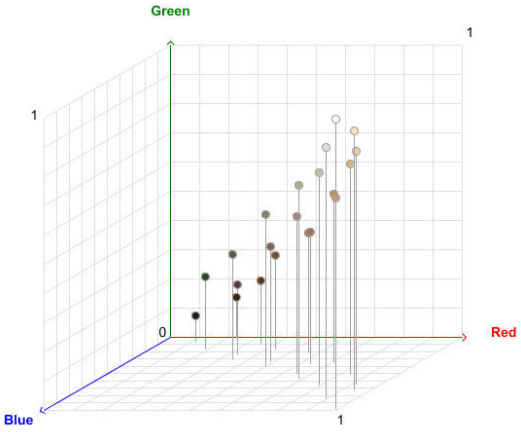
Bisher ist der Einfluss des Parameters ρ auf die resultierende Clusterung weitestgehend unberücksichtigt geblieben. Bei der Betrachtung der Kenngrößen fällt auf, dass der Wert der



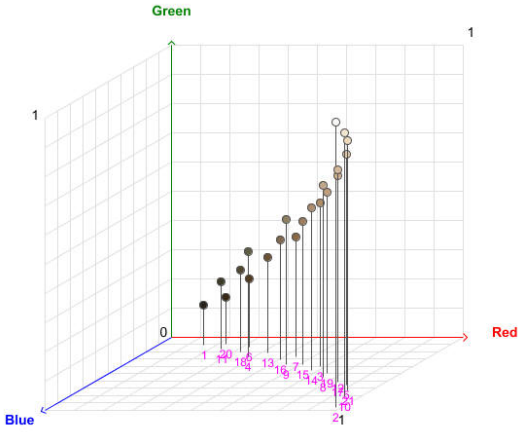
(a) Von SOM (2000 Zyklen) gebildetes Modell



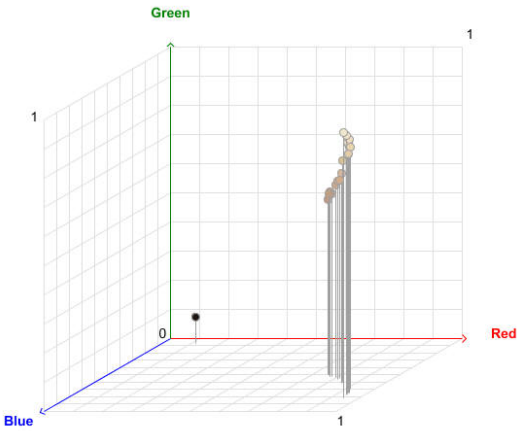
(b) Von SOM (2000 Zyklen) gebildete Zentren



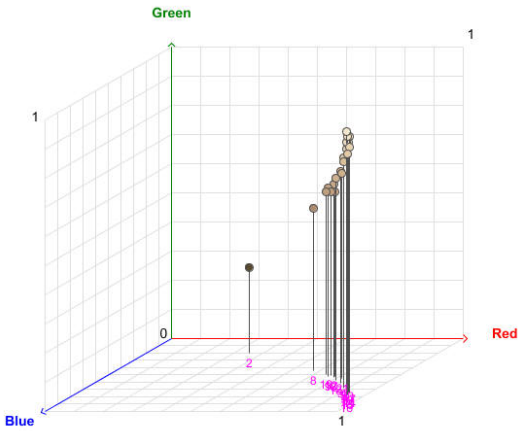
(c) Von FastAML ($\rho = 0.5$) gebildetes Modell



(d) Von FastAML ($\rho = 0.5$) gebildete Zentren



(e) Von FastAML ($\rho = 0.9999$) gebildetes Modell



(f) Von FastAML ($\rho = 0.9999$) gebildetes Modell

Abb. 5.9: Gegenüberstellung von Modell und Clusterzentren für SOM und FastAML. Es wurden 21 Klassen gebildet.

$GMMD$ bei $\rho = 0.9999$ einbricht. Betrachte dazu das von FastAML gebildete Modell in Abb. 5.9(e) und die resultierenden Clusterzentren in Abb. 5.9(f). Die Berücksichtigung der Häufigkeitsstruktur durch die in Abschnitt 1.2.3 eingeführte Gewichtungsfunktion führt zu einer starken Anpassung an die Daten. Die Startwerte und Clusterzentren konzentrieren sich im Bereich der häufig vorkommenden hellen Brauntöne. Seltene Elemente bleiben unberücksichtigt. Aus Abb. 5.9(f) wird auch ersichtlich, warum der Wert der $GMMD$ nahe an den Wert einer von EM erzeugten Clusterung heranreicht. Die Darstellung der Clusterzentren im Merkmalsraum ähnelt — verglichen mit den Darstellungen der anderen Clusterungen — am stärksten der von EM erzeugten Clusterung in Abb. 5.8(e).

Über den Parameter ρ ermöglicht es der FastAML-Algorithmus in diesem Fall, die Häufigkeitsstruktur so zu berücksichtigen, dass ähnliche Clusterungen wie mit dem EM-Algorithmus erzeugt werden können. Im Unterschied zu FastAML, dessen Laufzeit im Bereich von Sekunden liegt, benötigt der EM-Algorithmus für ein ähnliches Ergebnis über 1 Stunde.

5.3.3 Clusterungen der Luftaufnahme in Abb. 5.3(b)

Als günstig für die Luftaufnahme in Abb. 5.3(b) hat sich eine Einteilung in 15 Klassen erwiesen. Die Ergebnisse dieser Einteilung sind im Folgenden beschrieben.

Ansicht der segmentierten Luftaufnahme

Exemplarisch betrachtet seien Clusterungen der Luftaufnahme in Abb. 5.3(b) mit k-Mittelwert (siehe Abb. 5.10) und AML (siehe Abb. 5.11). Die Bilder der Clusterungen mit WTAN, QSOM, SOM und EM gleichen der Clusterung mit k-Mittelwert so stark, dass sie nicht dargestellt werden. Auch auf eine Darstellung der Clusterung mit FastAML wird verzichtet.

Ähnlich wie in Abschnitt 5.3.3 bleiben die wesentlichen Strukturen des Ursprungsbildes bei beiden Clusterungen erhalten. Zugleich ist die Anzahl der Farben von 36067 auf 15 reduziert, was in etwa dem Faktor 2400 entspricht. Es fällt allerdings auf, dass die mit AML durchgeführte Clusterung die Farben des Ursprungsbildes besser erhält. Dagegen wirkt das Bild der Clusterung mit k-Mittelwert etwas blasser und grauer. Insgesamt sind sich die geclusterten Luftaufnahmen etwas ähnlicher als die in Abschnitt 5.3.2 gezeigten Clusterungen.

Das Verfahren k-Mittelwert erfüllt die Anforderungen im Sinne der Anwendung nur für ein Merkmal, nämlich die Abstände zwischen den Farben (Merkmal 2). Die Übergänge zwischen

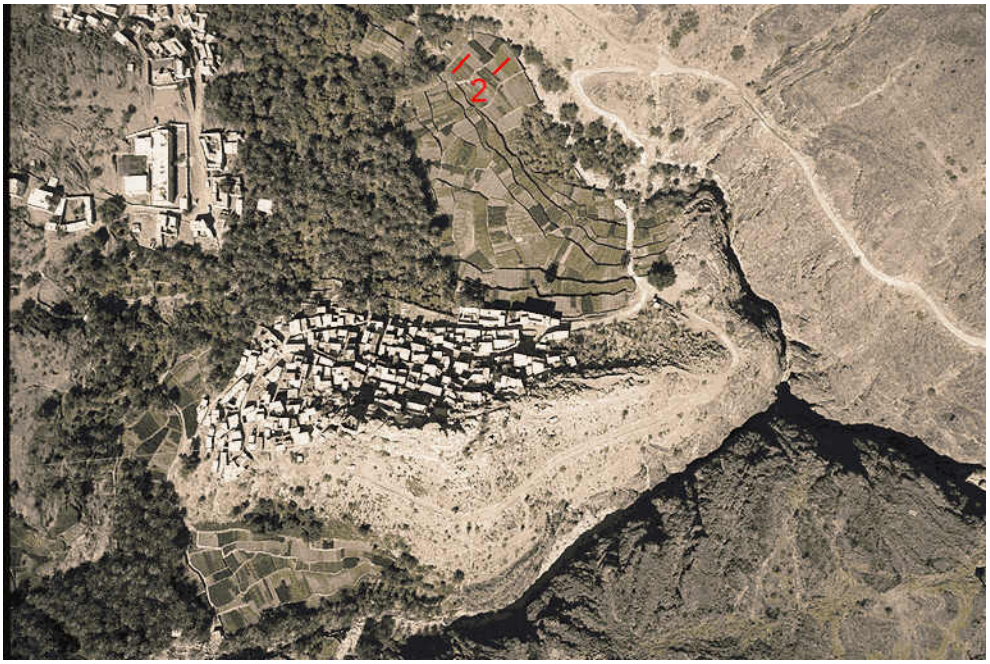


Abb. 5.10: Einteilung der Luftaufnahme aus Abb. 5.3(b) in 15 Cluster mit k-Mittelwert

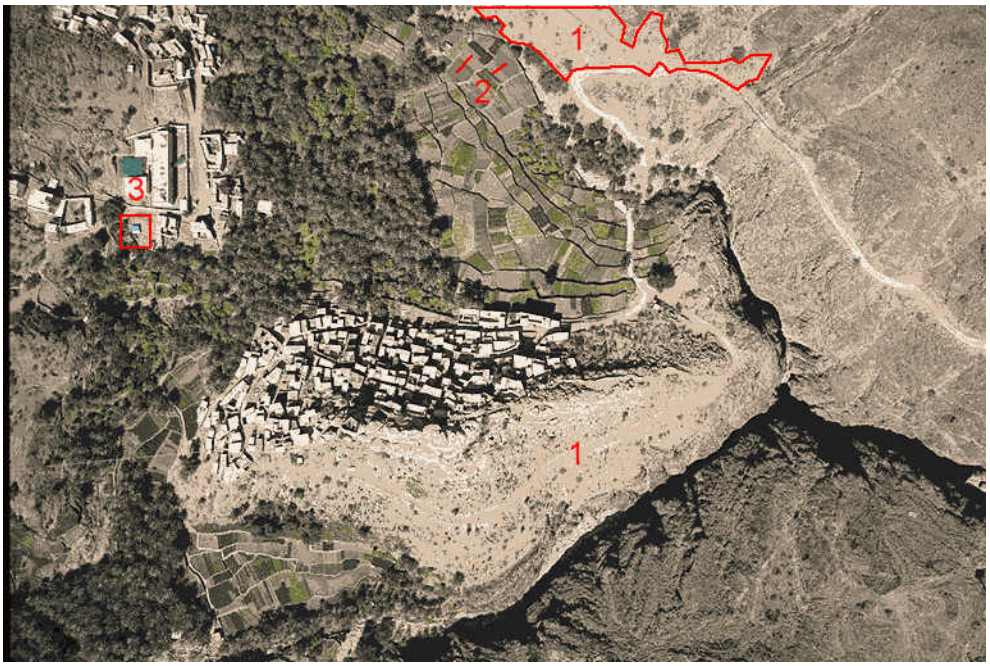


Abb. 5.11: Einteilung der Luftaufnahme aus Abb. 5.3(b) in 15 Cluster mit AML ($\rho = 0.999$)

den gebildeten Farbflächen sind ähnlich abrupt wie bei der Clusterung durch AML. Zusammenhängende Farbflächen (Merkmal 1) werden zwar auf den Feldern gebildet (wobei diese relevant für die Erkennung von Erosion sind), bezogen auf das gesamte Bild erzeugt AML jedoch größere Flächen. Seltene Elemente (Merkmal 3) werden nicht erkannt.

Im Vergleich zu der Clusterung der Luftaufnahme in Abb. 5.3(a) fällt jedoch auch bei AML die Flächenbildung etwas geringer aus (Merkmal 1). Die Heterogenität der Farbflächen ist hingegen hoch (Merkmal 2). Zudem werden seltene Elemente eindeutig erkannt (Merkmal 3).

Betrachtung der berechneten Kenngrößen

In Abb. 5.12 sind die Kenngrößen verschiedener Clusterungen der Luftaufnahme in Abb. 5.3(b) für eine Einteilung in 15 Cluster aufgeführt. Die absoluten „Gewinner“ bezüglich der jeweiligen Größe sind dunkel markiert, die relativen Gewinner für eine Gruppe von Clusterungen eines Verfahrens hell.

Die Werte aller Kenngrößen (bis auf Ausnahmen bei der Laufzeit) sind für die jeweiligen Verfahren und Parameter größer als in Abb. 5.7. Der Wert der SSW ist bei den AML-Algorithmen — wie zu erwarten — etwas größer als bei den anderen Verfahren. Erneut identifiziert eine reine Bewertung durch die Intra-Class-Varianz nicht die beste Clusterung im Sinne der Anwendung.

Der AML-Algorithmus und das k-Mittelwert-Verfahren erreichen die höchsten Werte für die $GMMD$ (und wurden aus diesem Grund exemplarisch für den optischen Vergleich ausgewählt). Dabei ist der Unterschied zwischen dem höchsten und zweithöchsten Wert für die $GMMD$ nicht ganz so groß wie in Abb. 5.7. Das Ergebnis stimmt mit der optischen Bewertung der Clusterungen überein. Eine Erklärung wird auf den nächsten Abschnitt verschoben, in dem die Clusterzentren im Merkmalsraum dargestellt sind.

FastAML erreicht für $\rho = 0.99$ im Vergleich mit AML, k-Mittelwert und EM keinen guten Wert für die $GMMD$. Dabei ist zu berücksichtigen, dass die Häufigkeitsverteilung der Farben für die Clusterung der Luftaufnahme in Abb. 5.3(b) offensichtlich eine Rolle spielt. So erreicht AML einen hohen Wert der $GMMD$ nur für $\rho = 0.999$.

Der Unterschied beruht auf dem in Abschnitt 4.5.5 beschriebenen Detail, dass bei FastAML die Abstände zwischen den Instanzen aufgrund der Sortierung bereits vor Ablauf des Algorithmus transformiert werden müssen. Um eine ähnliche Clusterung wie mit AML zu erreichen, ist

Verfahren	Parameter	SSW	SSB	MMD	GMMD			Laufzeit
					$a = 0.1$	$a = 0.5$	$a = 0.9$	in ms
FastAML	$\rho = 0$	0.006832	0.151612	0.155377	0.321564	0.337333	0.345995	4396
FastAML	$\rho = 0.5$	0.006833	0.151611	0.155376	0.321548	0.337333	0.346004	4497
FastAML	$\rho = 0.75$	0.006833	0.151611	0.155353	0.321464	0.337249	0.345919	4456
FastAML	$\rho = 0.9$	0.006832	0.151612	0.155377	0.321564	0.337333	0.345995	4457
FastAML	$\rho = 0.99$	0.00729	0.151154	0.158129	0.33152	0.344181	0.351136	4456
FastAML	$\rho = 0.999$	0.008959	0.149485	0.147009	0.284789	0.297831	0.304995	4497
FastAML	$\rho = 0.9999$	0.007917	0.150527	0.135009	0.218833	0.231066	0.237786	4536
AML	$\rho = 0.99$	0.0069	0.151544	0.155505	0.320306	0.336014	0.344642	591871
AML	$\rho = 0.999$	0.006765	0.151679	0.189691	0.466802	0.478827	0.478827	600404
EM	iter $= 100$	0.003739	0.154705	0.118152	0.328546	0.336003	0.340099	1226663
EM	iter $= 250$	0.003799	0.154645	0.117838	0.358192	0.359615	0.360396	3083564
EM	iter $= 500$	0.003807	0.154637	0.117097	0.387626	0.388445	0.388894	6173898
SKM		0.003473	0.154971	0.117592	0.43235	0.439907	0.444059	502783
WTAN	$c = 5$	0.003813	0.154631	0.110037	0.230806	0.240323	0.245551	39888
WTAN	$c = 10$	0.00383	0.154614	0.109722	0.22726	0.232416	0.235248	75979
WTAN	$c = 16$	0.003794	0.15465	0.110261	0.235154	0.244317	0.24935	119311
WTAN	$c = 20$	0.00382	0.154624	0.109864	0.230064	0.235896	0.239099	148153
WTAN	$c = 24$	0.003817	0.154627	0.10992	0.229529	0.235103	0.238165	177145
WTAN	$c = 30$	0.003821	0.154623	0.109906	0.230149	0.23845	0.24301	220508
QSOM	$c = 5$	0.003809	0.154635	0.108825	0.231407	0.240427	0.245382	38034
QSOM	$c = 10$	0.00374	0.154704	0.109746	0.246318	0.255525	0.260583	72494
QSOM	$c = 14$	0.003751	0.154693	0.109052	0.24216	0.247161	0.249908	99974
QSOM	$c = 20$	0.003814	0.15463	0.10862	0.227661	0.234228	0.237836	151297
QSOM	$c = 30$	0.003723	0.154721	0.110051	0.250907	0.259622	0.264409	210212
QSOM	$c = 32$	0.003751	0.154693	0.109604	0.241634	0.249614	0.253998	224062
QSOM	$c = 35$	0.003714	0.15473	0.109846	0.253264	0.260314	0.264188	244701
QSOM	$c = 40$	0.00376	0.154684	0.109309	0.239615	0.247467	0.25178	279101
SOM	$c = 10$	0.008641	0.149803	0.087284	0.14977	0.16236	0.169275	63911
SOM	$c = 50$	0.004327	0.154117	0.103875	0.186829	0.196154	0.201276	305730
SOM	$c = 100$	0.004183	0.154261	0.107668	0.190214	0.201074	0.20704	607784
SOM	$c = 200$	0.003981	0.154463	0.109003	0.206135	0.213506	0.217555	1211912
SOM	$c = 500$	0.003737	0.154707	0.110468	0.249732	0.259229	0.264445	3023978
SOM	$c = 700$	0.003717	0.154727	0.110032	0.252846	0.261289	0.265927	4232155
SOM	$c = 1000$	0.00372	0.154724	0.108313	0.248813	0.252963	0.255243	6078160
SOM	$c = 1500$	0.003675	0.154769	0.108488	0.261841	0.265613	0.267685	9062691
SOM	$c = 2000$	0.003661	0.154783	0.108298	0.267291	0.270861	0.272821	12081752
SOM	$c = 3000$	0.00357	0.154874	0.112726	0.316288	0.320203	0.322353	18240639
SOM	$c = 4000$	0.003557	0.154887	0.113777	0.324533	0.328261	0.330309	24234867

Abb. 5.12: Kenngrößen von Clusterungen der Luftaufnahme in Abb. 5.3(b) für unterschiedliche Verfahren und Parametersätze. Es wurde eine Einteilung in 15 Cluster vorgenommen.

ρ bei FastAML daher etwas anders zu wählen. Der große Unterschied zwischen den Werten der $GMMD$ ergibt sich daher, dass für $\rho \in [0.9, 1]$ bereits kleine Änderungen des Parameters zu großen Veränderungen der Clusterung führen. Trotz der Implementation des von Heim [Hei02] vorgeschlagenen λ^ρ-Ansatzes gestaltet sich die Adjustierung des Parameters ρ in der Praxis nach wie vor schwierig. So liegt z. B. für $\rho = 0.99$ der Wert der $GMMD$ bei Verwendung von AML bereits unter dem von FastAML erreichten Wert. Die Aussage aus Abschnitt 5.3.2, dass über den Parameter ρ eine ähnliche Clusterung wie mit EM erzeugt werden kann, ist auf die Luftaufnahme in Abb. 5.3(b) nicht direkt übertragbar.

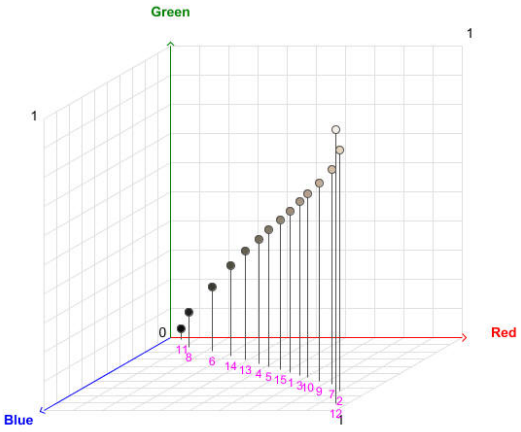
In Hinblick auf die Laufzeiten ist anzumerken, dass die Laufzeit von AML für 36067 Farben mit ungefähr 10 Min. deutlich über der in Abb. 5.7 aufgeführten Laufzeit von AML liegt. Für das Bild mit 10457 Farben benötigt AML nur ungefähr 1 Min., obwohl die Anzahl von 21 gegenüber 15 zu bestimmenden Clustern größer ist. Die empirischen Ergebnisse bestätigen die in Abschnitt 4.2.3 durchgeführte Rechnung und Aussagen. Die höhere Laufzeit von FastAML in Abhängigkeit von der Anzahl der Farben ist auf den Schritt der Erstellung der Häufigkeitstabelle zurückzuführen. Die Tabelle wurde durch eine Hash-Tabelle (siehe [CLRS01]) implementiert, so dass der zu erwartende Anstieg der Laufzeit linear ist.

Darstellung der Clusterzentren

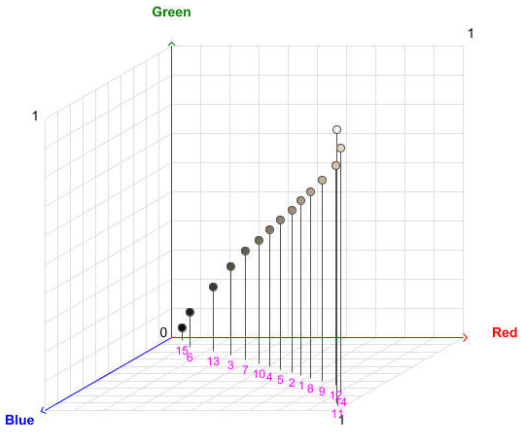
In Abb. 5.13 sind die Zentren von Clusterungen der verschiedenen Verfahren dargestellt.

Analog zu Abschnitt 5.3.2 besteht eine sehr große Ähnlichkeit zwischen den von WTAN, QSOM, SOM, k-Mittelwert und EM erzeugten Clusterzentren. Dagegen erzeugt AML Clusterzentren, die sehr weit von der gedachten Linie entfernt sind, auf welcher die Zentren der anderen Clusterungen liegen (siehe Abb. 5.13(f)). Erst mit der Darstellung der Zentren lässt sich erklären, warum sich beim optischen Vergleich der segmentierten Luftaufnahme durch AML und k-Mittelwert gewisse Ähnlichkeiten ergeben. Es lässt sich weiter erklären, warum k-Mittelwert einen so hohen Wert für die $GMMD$ erreicht.

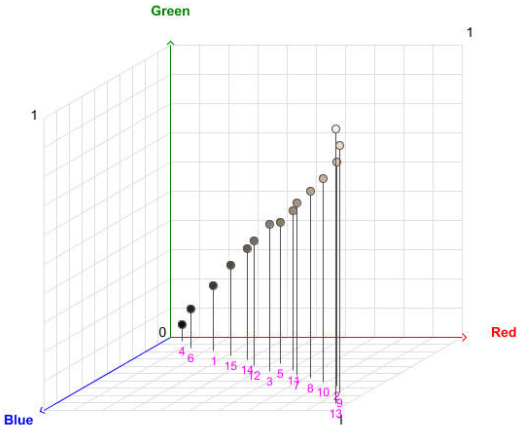
Die auf den Häufigkeiten der Datenpunkte basierenden Verfahren nähern sich gut an die Datenschwerpunkte an. Das lässt sich daran erkennen, dass die ermittelten Zentren der Grauwert-Skala des RGB-Farbwürfels (siehe Abb. 2.3) sehr nahe kommen. Tatsächlich wird die Originalaufnahme in Abb. 5.3(b) weitestgehend von den Grautönen der Gesteinsformationen und den mittleren Grüntönen der Vegetation und der Anbaugebiete beherrscht. Die Annäherung an die Datenschwerpunkte erklärt zunächst, warum die Farben in dem von k-Mittelwert erzeugten



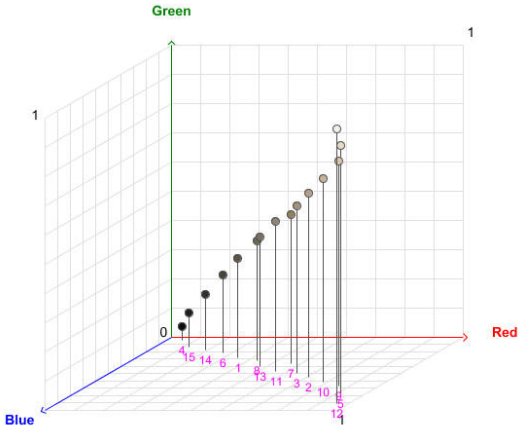
(a) WTAN (16 Zyklen)



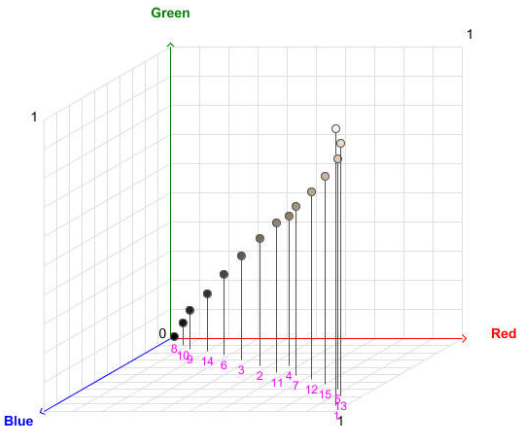
(b) QSOM (35 Zyklen)



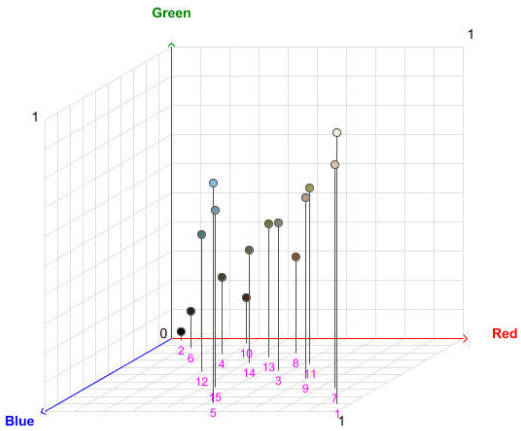
(c) SOM (4000 Zyklen)



(d) k-Mittelwert



(e) EM (max. Iterationstiefe 500)



(f) AML ($\rho = 0.999$)

Abb. 5.13: Zentren von Clusterungen der Luftaufnahme aus Abb. 5.3(b) mit den verschiedenen Clusteringverfahren. Die Aufnahme wurde in 15 Cluster eingeteilt (die jeweilige Nummer des Clusters ist teilweise unterhalb des Würfels zu erkennen).

Bild etwas blass und grau wirken.

Im Unterschied zur Konzentration der hellen Brauntöne in den Darstellungen aus Abb. 5.8 sind die Datenschwerpunkte des Originalbildes aus Abb. 5.3(b) weiter gestreckt. Sie verteilen sich besser im Merkmalsraum und haben größeren Abstand zueinander. Ein höherer Abstand der Zentren und die äquidistante Verteilung im Merkmalsraum werden durch das Kriterium der $GMMD$ hoch bewertet (siehe Abschnitt 1.1.5). Das k-Mittelwert-Verfahren kann folglich einen sehr guten Wert für die $GMMD$ erreichen, weil die Verteilung der Datenpunkte im Ursprungsbild bereits günstig ausfällt. Aufgrund des größeren Abstandes der Datenschwerpunkte bilden sich bei Klassifikation mit dem k-Mittelwert-Verfahren daher auch etwas größere Farbflächen aus. Dadurch verringern sich die Unterschiede der von AML und k-Mittelwert erzeugten Clusterungen beim optischen Vergleich der segmentierten Luftaufnahme.

Die noch größeren Abstände der von AML gebildeten Zentren werden in diesem Fall durch das Kriterium der MMD etwas besser wiedergegeben. Der Wert ist deutlich höher als beim k-Mittelwert-Verfahren (siehe Abb. 5.12). Auch die Tatsache, dass sich die Darstellungen der Verfahren WTAN, QSOM, SOM, EM und k-Mittelwert ähnlich sehen, wird durch die MMD besser abgebildet. Die numerischen Unterschiede der $GMMD$ für diese Verfahren lassen sich durch Betrachtung der Zentren im Merkmalsraum nicht erklären.

Eine Gesamtbewertung von Clusterungen der Luftaufnahme in Abb. 5.3(b) anhand der Darstellung im Merkmalsraum ergibt, dass AML die Zentren besser verteilt und dadurch seltene Elemente erhält. Im Gegensatz zu den anderen Verfahren genügt die durch AML erzeugte Clusterung somit als einzige den Anforderungen im Sinne der Anwendung.

Kapitel 6

Zusammenfassung und Ausblick

Es ist theoretisch gezeigt worden, dass der AML-Algorithmus abhängig von der Anzahl r der Datenpunkte für eine kleine Anzahl zu bestimmender Cluster k quadratische, für eine große Anzahl kubische Laufzeit annimmt. Die hohe Laufzeit ist für zu erwartende kleine k auf den Initialschritt zurückzuführen, in dem das Farthest-Pair-Problem gelöst werden muss. Das Problem besteht darin, die beiden Punkte mit dem größten Abstand in einer Punktmenge zu finden.

Für zweidimensionale Vektoren lässt sich das Farthest-Pair-Problem über die Berechnung der konvexen Hülle in $O(r \log r)$ Schritten lösen. Sowohl theoretische Überlegungen als auch praktische Durchläufe mit einer Implementation des Quickhull-Algorithmus zeigen, dass Hull-Algorithmen in höheren Dimensionen nicht mehr effizient arbeiten. Für die dritte Dimension erreichen randomisierte und deterministische Pruning-Algorithmen eine zu erwartende Zeitkomplexität von $O(r \log r)$. Dabei wird ein exaktes Pruning-Kriterium verwendet, das alle nicht mehr benötigten Punkte aus der Menge ausschließt. Der Algorithmus lässt sich nicht auf höhere Dimensionen erweitern. Ein von Yao entwickelter Algorithmus erreicht in Dimension $d \geq 4$ eine Zeitkomplexität von $O(r^{2-a(d)} \log^{1-a(d)} r)$ mit $a(d) = 2^{-(d+1)}$.

Der in dieser Arbeit entwickelte Pruning-Algorithmus $HighDiameter$ setzt ein modifiziertes, weniger restriktives Pruning-Kriterium ein. Es ist gezeigt worden, dass die Laufzeit bei zufälliger Reihenfolge der Abstandsberechnungen nach oben durch $O(dr^2)$ beschränkt ist. Varianten des Algorithmus, welche die Reihenfolge der Abstandsberechnungen durch eine unterschiedliche Sortierung der Punkte verändern, weisen die gleiche obere Schranke auf. Theoretische Überlegungen führen zu der Annahme, dass der *worst case* für die meisten Eingaben nur mit geringer Wahrscheinlichkeit eintritt. Erste empirische Untersuchungen mit simulierten und realen Datensätzen stützen die theoretischen Ausführungen. Mit den unterschiedlichen Va-

rianten bietet sich dem Anwender die Möglichkeit, die Art des Durchlaufs an die Struktur der Daten anzupassen. Schlechte Durchlaufzeiten können so mit hoher Wahrscheinlichkeit vermieden werden.

Der FastAML-Algorithmus, der auf $HighDiameter$ zurückgreift, zeigt bezüglich der Laufzeit in Theorie und Praxis eine deutliche Überlegenheit gegenüber dem AML-Algorithmus. Die für Clusterungen benötigte Zeit liegt teilweise im Bereich von Sekunden- und Millisekunden, auch für hohe Anzahlen von Datenpunkten.

Clusterungen zweier Luftaufnahmen im praktischen Teil der Arbeit demonstrieren, dass sich das Kriterium der gewichteten mittleren Minimaldistanz ($GMMD$) zur Beurteilung von Clusterungen im Sinne der Anwendung bewährt. Die traditionelle Beurteilung durch die Intra-Class-Varianz ist nicht in der Lage, die beste Clusterung im Sinne der Anwendung zu identifizieren. Zugleich kann gezeigt werden, dass die AML-Algorithmen im Vergleich zu den anderen Verfahren für beide Luftaufnahmen die beste Clusterung im Sinne der Anwendung erzeugen. Die Ergebnisse im empirischen Teil stimmen mit den in den Arbeiten von Zerbst [Zer01] und Tschiersch [Tsc02] gewonnenen überein.

Mit dem FastAML-Algorithmus steht ein äußerst effizienter Algorithmus zur Verfügung, um Luftaufnahmen im Bereich der Erosionsproblematik zu clustern. Durch die hohe Effizienz ergeben sich weitere Anwendungsgebiete. Denkbar sind etwa Anwendungen, die eine Clusterung von Bildern in Echtzeit benötigen. Oft sind es gerade die seltenen Elemente in einem Bild, die erkannt werden müssen.

Ein Beispiel für eine solche Anwendung ist etwa der Roboterfußball. In der untersten Liga werden von einem Zentralcomputer per Funk Befehle an drei Roboter (pro Mannschaft) auf dem Spielfeld übermittelt. Bilder des Spielfeldes gelangen zur weiteren Auswertung über eine Kamera an den Zentralcomputer. Es gilt, die Roboter, d.h. Freund und Gegner, und den Ball eindeutig zu identifizieren. Je nach Ausleuchtung des Spielfeldes kann es dabei passieren, dass der orangene Farbton des Balls beinahe vollständig verschwindet: der Ball wird zu einem seltenen Element. Die Roboter wiederum sind mit sog. Patches ausgestattet, auf denen Farbflächen in Polygonform angebracht sind. Die Bildung zusammenhängender Farbflächen durch den FastAML-Algorithmus kann sich auch in diesem Zusammenhang als nützlich erweisen: je schärfer die Übergänge und je höher der Abstand zwischen den Farben, desto besser lassen sich die Polygone bestimmen. Als Clusteringverfahren könnte der FastAML-Algorithmus eingesetzt werden, um sich regelmäßig an die neuen Lichtverhältnisse und der damit verbundenen dynamisch wechselnden Ausleuchtung des Spielfeldes zu adaptieren. Dadurch könnte einem nach-

geschalteten Klassifikator — d.h. einem überwachten Lernverfahren — die Arbeit erleichtert werden. Allgemein sollte die starke Bildung von Farbflächen die nachfolgende Segmentierung von Bildern erleichtern können.

Im Zusammenhang mit nachgeschalteten überwachten Lernverfahren könnte es sich zudem als nützlich erweisen, das Modell numerischer Clusteringverfahren in eine symbolische Repräsentation (z.B. prädikatenlogische Regeln) umzuwandeln. Dadurch ließen sich die Ergebnisse durch den Menschen unter Umständen noch besser interpretieren.

Literaturverzeichnis

[AGR94] AMATO, N. M., M. T. GOODRICH und E. A. RAMOS: *Parallel algorithms for higher-dimensional convex hulls*. In: *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, Seiten 683–694, 1994.

[AS98] AGARWAL, PANKAJ K. und MICHA SHARIR: *Efficient algorithms for geometric optimization*. ACM Computing Surveys, 30(4):412–458, 1998.

[AS00] AGARWAL, PANKAJ K. und SANDEEP SEN: *Randomized Algorithms for Geometric Optimization Problems*, 2000.

[ASRJ77] ANDERSON, J. A., J. W. SILVERSTEIN, S. A. RITZ und R. S. JONES: *Distinctive features, categorical perception and probability learning: Some applications of a neural model*. Psychological Review, 84:413–451, 1977.

[Bes98] BESPAMYATNIKH, S.: *An efficient algorithm for the three-dimensional diameter problem*. In: *Proc. 9th Annu. ACM-SIAM Sympos. Discrete Algorithms*, Seiten 137–146, 1998.

[Bes01] BESPAMYATNIKH, S. N.: *An efficient algorithm for the three-dimensional diameter problem*. Discrete Comp. Geom., 25:235–255, 2001.

[BOR99] BORODIN, A., R. OSTROVSKY und Y. RABANI: *Subquadratic approximation algorithms for clustering problems in high dimensional spaces*. In: *Proc. 31th Annu. ACM Sympos. Theory Comput.*, Seiten 435–444, 1999.

[CEGS93] CHAZELLE, B., H. EDELSBRUNNER, L. J. GUIBAS und M. SHARIR: *Diameter, width, closest line pair and parametric searching*. Discrete Comput. Geom., 10:183–196, 1993.

[Che96] CHEN, JIANER: *Computational Geometry: Methods and Applications*. Computer Science Department, Texas A&M University, 1996.

[CK70] CHAND, D. R. und S. S. KAPUR: *An algorithm for convex polytopes*. J. Assoc. Comput. Mach., 17:78–86, 1970.

[CLRS01] CORMEN, THOMAS H., CHARLES E. LEISERSON, RONALD L. RIVEST und CLIFFORD STEIN: *Introduction to Algorithms*. MIT Press, 2. Auflage, 2001.

[CS89] CLARKSON, K. L. und P. W. SHOR: *Applications of randomized sampling in computational geometry*. Discrete Comput. Geom., 4:387–421, 1989.

[CSV01] CHEONG, OTFRIED, CHAN-SU SHIN und ANTOINE VIGNERON: *Computing Farthest Neighbors on a Convex Polytope*. Lecture Notes in Computer Science, 2108:159–??, 2001.

[dBvKOS00] BERG, MARK DE, MARC VAN KREFELD, MARK OVERMARS und OTFRIED SCHWARZKOPF: *Computational Geometry: algorithms and applications*. Springer, 2. Auflage, 2000.

[DH73] DUDA, RICHARD O. und PETER E. HART: *Pattern classification and scene analysis*. John Wiley & Sons, Inc., 1973.

[DMS01] DUDEL, J., R. MENZEL und R. F. SCHMIDT (Herausgeber): *Neurowissenschaft: vom Molekül zur Kognition*. Springer-Verlag, 2. Auflage, 2001.

[EK89] EGECIOGLU, O. und B. KALANTARI: *Approximating the Diameter of a Set of Points in the Euclidean Space*. Inform. Process. Lett., 32:205–211, 1989.

[Erd60] ERDÖS, P.: *On sets of distances of n points in Euclidian space*. Magy. Tud. Akad. Mat. Kut. Int. Kozi., 5:165–169, 1960.

[Eve86] EVERITT, BRIAN: *Cluster Analysis*. Gower Publishing Company Limited, 2. Auflage, 1986.

[Fis87] FISHER, D.: *Knowledge acquisition via incremental conceptual clustering*. Machine Learning, 2(2):139–172, 1987.

[Fis99] FISCHER, PAUL: *Algorithmisches Lernen*. Leitfäden der Informatik. B. G. Teubner, Stuttgart, Leipzig, 1999.

[FP99] FINOCCHIARO, DANIELE V. und MARCO PELLEGRINI: *On Computing the Diameter of a Point Set in High Dimensional Euclidean Space*. In: *European Symposium on Algorithms*, Seiten 365–377, 1999.

[GLF90] GENNARI, J. H., P. LANGLEY und D. FISHER: *Models of incremental concept formation*. Artificial Intelligence, 40:11–61, 1990.

[Gra72] GRAHAM, R. L.: *An efficient algorithm for determining the convex hull of a finite planar set*. Inform. Process. Lett., 1:132–133, 1972.

[Güt92] GÜTING, RALF HARTMUT: *Datenstrukturen und Algorithmen*. Leitfäden und Monographien der Informatik. B. G. Teubner, Stuttgart, 1992.

[GW92] GONZALEZ, RAFAEL C. und RICHARD E. WOODS: *Digital image processing*. Addison-Wesley, 1992.

[Har85] HARTUNG, JOACHIM: *Statistik: Lehr- und Handbuch der angewandten Statistik*. Oldenbourg, München, Wien, 4. Auflage, 1985.

[Heb49] HEBB, D. O.: *The Organization of Behaviour*. Wiley, New York, 1949.

[Hei02] HEIM, SUSANNE: *Berücksichtigung der Häufigkeitsverteilung im Maximum Linkage Clusteralgorithmus*. Diplomarbeit, Universität Dortmund, 2002.

[HN87] HECHT-NIELSEN, R.: *Counterpropagation networks*. Applied Optics, 26:4979–4984, 1987.

[HN89] HECHT-NIELSEN, R.: *Neurocomputing*. Addison-Wesley, 1989.

[Hop82] HOPFIELD, J. J.: *Neural networks and physical systems with emergent collective computational abilities*. In: *Proceedings of the National Academy of Sciences 79*, 1982.

[HPK91] HERTZ, JOHN A., RICHARD G. PALMER und ANDERS S. KROGH: *Introduction to the theory of neural computation*. Computational and neural systems series. Addison-Wesley, 1991.

[HY61] HOCKING, J. G. und G. S. YOUNG: *Topology*. Addison-Wesley, Reading, MA, 1961.

[Ind99] INDYK, P.: *A sublinear-time approximation scheme for clustering in metric spaces*. In: *Proc. 40th Sympos. on Foundations of Computer Science*, Seiten 154–159, 1999.

[JW92] JOHNSON, RICHARD A. und DEAN W. WICHERN: *Applied Multivariate Statistical Analysis*. Prentice-Hall, Inc., 3. Auflage, 1992.

[Kle80] KLEE, V.: *On the complexity of d-dimensional Voronoi diagrams*. Archiv der Mathematik, 34:75–80, 1980.

[Koh72] KOHONEN, T.: *Correlation matrix memories*. IEEE Transactions Computers, 4:353–359, 1972.

[Koh82] KOHONEN, T.: *Self-Organized Formation of Topologically Correct Feature Maps*. Biological Cybernetics, 43:59–69, 1982.

[Koh84] KOHONEN, T.: *Self-Organizing and Associative Memory*. Springer-Verlag, Berlin, 1984.

[Koh01] KOHONEN, TEUVO: *Self-organizing maps*, Band 30 der Reihe *Springer series in information sciences*. Springer, 3. Auflage, 2001.

[Lug01] LUGER, GEORGE F.: *Künstliche Intelligenz: Strategien zur Lösung komplexer Probleme*. Pearson Studium, 4. Auflage, 2001.

[MP43] MCCULLOCH, W. S. und W. PITTS: *A logical calculus of the ideas immanent in nervous activity*. Bullletin of Mathematical Biophysics, 5:115–133, 1943.

[MS71] MCMULLEN, P. und G. C. SHEPHARD: *Convex Polytopes and the Upper Bound Conjecture*. Cambridge University Press, Cambridge, England, 1971.

[MS96] MATOUŠEK, J. und O. SCHWARZKOPF: *A deterministic algorithm for the three-dimensional diameter problem*. Comput. Geom. Theory Appl., 6:253–262, 1996.

[Nil65] NILSSON, N. J.: *Learning Machines*. McGraw-Hill, New York, 1965.

[O'R00] O'ROURKE, JOSEPH: *Computational geometry in C*. Cambridge University Press, 2. Auflage, 2000.

[PH77] PREPARATA, F. P. und S. J. HONG: *Convex hulls of finite sets of points in two and three dimensions*. Commun. ACM, 20:87–93, 1977.

[PS85] PREPARATA, F. P. und M. I. SHAMOS: *Computational Geometry: An Introduction*. Springer-Verlag, New York, 1985.

[Ram97a] RAMOS, E.: *Construction of 1-d lower envelopes and applications*. In: *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, Seiten 57–66, 1997.

[Ram97b] RAMOS, E.: *Construction of unit-balls and diameter of a point set in* R^3. Comput. Geom. Theory Appl., 9:57–65, 1997.

[Ram00a] RAMOS, E. A.: *Deterministic algorithms for 3-D diameter and some 2-D lower envelopes*. In: *Proc. 16th Sympos. Comput. Geom.*, 2000.

[Ram00b] RAMOS, E. A.: *An optimal deterministic algorithm for computing the diameter of a 3-d point set*. In: *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, 2000.

[Ran89] RANKIN, JOHN R.: *Computer graphics software construction*. Prentice Hall, Australia, 1989.

[RHW86] RUMELHART, D. E., G. E. HINTON und R. J. WILLIAMS: *Learning internal representations by error propagation*. In: MCCLELLAND, J. L., D. E. RUMELHART und THE PDP RESEARCH GROUP (Herausgeber): *Parallel Distributed Processing*. MIT Press, Cambridge, 1986.

[RN95] RUSSEL, STUART J. und PETER NORVIG: *Artificial Intelligence: a modern approach*. Prentice Hall, Inc., 1995.

[Ros58] ROSENBLATT, F.: *The perceptron: A probabilistic model for information storage and organization in the brain*. Psychological Review, 65:386–408, 1958.

[Ruc84] RUCKER, R.: *The Fourth Dimension: Toward a Geometry of Higher Reality*. Houghton Mifflin, Boston, 1984.

[Sha87] SHAMOS, M. I.: *Computational Geometry*. Doktorarbeit, Dept. Comput. Sci., Yale Univ., New Haven, CT, 1987.

[ST86] SARNAK, N. und E. TARJAN: *Planar point location using persistent search trees*. Commun. ACM, 29:669–679, 1986.

[Tre99] TREPEL, MARTIN: *Neuroanatomie: Struktur und Funktion*. Urban & Fischer Verlag, 2. Auflage, 1999.

[Tsc02] TSCHIERSCH, LARS: *Strategie zur Lösung von Erosionsproblemen unter Verwendung von Luft- und Bodendaten*. Doktorarbeit, Universität Dortmund, 2002.

[Tsy71] TSYPKIN, Y. Z.: *Foundations of the Theory of Learning Systems*. Academic Press, New York, 1971.

[UNC94] UNCCD, 241/27: *Elaboration of an international convention to combat desertification in countries experiencing serious drought and/or desertification, particularly in Africa*, 1994.

[WF01] WITTEN, IAN H. und EIBE FRANK: *Data Mining: Praktische Werkzeuge und Techniken für das maschinelle Lernen*. Carl Hanser Verlag, 2001.

[WMJ00] WROBEL, STEFAN, KATHARINA MORIK und THORSTEN JOACHIMS: *Maschinelles Lernen und Data Mining*. In: GÖRZ, GÜNTHER, CLAUS-RAINER ROLLINGER und JOSEF SCHNEEBERGER (Herausgeber): *Handbuch der künstlichen Intelligenz*, Kapitel 14, Seiten 517–597. Oldenbourg, München, 3. Auflage, 2000.

[Yao82] YAO, A. C.: *On constructing minimum spanning trees in k-dimensional spaces and related problems*. SIAM J. Computation, 11(4):721–736, 1982.

[YB61] YAGLOM, I. M. und V. G. BOLTYANSKII: *Convex Figures*. Holt, Rinehart and Winston, New York, 1961.

[Zer01] ZERBST, MATTHIAS: *Die pixelbasierte Clusterung von Luftaufnahmen im Rahmen von Erosionsuntersuchungen*. Doktorarbeit, Universität Dortmund, 2001.