technische universität
dortmund

Masterarbeit

Kernel-k-means Methoden zur spektralen Clusteranalyse von Graphen

Lukas Pradel
16. März 2015

Gutachter:
Prof. Dr. Christian Sohler
Dr. Melanie Schmidt

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl II - Effiziente Algorithmen und Komplexitätstheorie
http://ls2-www.cs.tu-dortmund.de/

An erster Stelle möchte ich mich bei Prof. Dr. Sohler in besonderem Maße für das interessante und ergiebige Thema bedanken. Darüber hinaus hat mir meine langjährige Zusammenarbeit mit den Mitarbeitern am Lehrstuhl 2 stets viel Freude bereitet. Ich habe dabei viel gelernt. Meiner Betreuerin Frau Dr. Schmidt danke ich für die hervorragende Betreuung bei dieser Arbeit und die nötigen Denkanstöße. Meiner Familie danke ich für ihre Unterstützung während des gesamten Studiums. Auf Eric Fiege, Michael Freimuth, Ersin Özdemir und David Sturm konnte ich mich während meines Studiums immer verlassen. Schließlich danke ich Sven Boyes, Sebastian Brameier und Chris Koch für die nötige Ablenkung vom Verfassen dieser Arbeit.

Inhaltsverzeichnis

1 Einleitung

Die Clusteranalyse beschäftigt sich mit Verfahren, die Ähnlichkeitsstrukturen in (großen) Datenmengen identifizieren. Der englische Begriff *cluster* lässt sich in diesem Kontext am ehesten mit Gruppe oder Ansammlung übersetzen. Dem Gebiet liegt die Annahme zu Grunde, dass in gegebenen Datenmengen typischerweise Strukturen existieren, die mit einem geeigneten Ähnlichkeitsmaß erkannt werden können. Diese grundlegende Annahme ist insofern plausibel, als dass die zu analysierenden Datenmengen in der Praxis in der Regel aus einem konkreten Anwendungsfall gesammelt oder aufgezeichnet werden, in dem bestimmte Muster oder sich wiederholende Prozesse existieren. Bei der Analyse von Daten aus natürlichen Prozessen sind diese beispielsweise oft gemäß einer bestimmten Wahrscheinlichkeitsverteilung verteilt. Hinzu kommt, dass Daten ohne jegliche Strukturen zumeist ohnehin nicht besonders aufschlussreich sind. Wenn die vorhandenen Strukturen einmal erkannt sind, kann der gesamte Datensatz in Gruppen oder *Cluster* gegliedert werden, sodass innerhalb eines Clusters große Ähnlichkeit zwischen Objekten besteht, während Objekte, die unterschiedlichen Clustern zugeordnet sind, geringe Ähnlichkeit aufweisen. Anhand der Cluster, die mit Verfahren der Clusteranalyse gebildet wurden, lassen sich dann häufig aufschlussreiche Aussagen über die Eingabedaten treffen, die beim bloßen Anblick der gesamten unstrukturierten Datenmenge nicht möglich gewesen wären. Das gilt insbesondere, wenn die Daten hochdimensional sind, sodass eine graphische Auswertung auch nicht möglich ist.

Wir betrachten dazu einleitend ein einfaches Beispiel der Gegenwart: die Analyse von Strukturen in sozialen Netzwerken. Die Eingabeobjektmenge besteht in diesem Fall aus Objekten, die Informationen über Beziehungen und Aktivitäten zwischen Nutzern des sozialen Netzwerks enthalten. Mit Hilfe der Clusteranalyse können beispielsweise Gruppen von Benutzern identifiziert werden, die viel miteinander interagieren und somit eine soziale Community, also eine Gruppe von Freunden oder Bekannten, bilden. In diesem Beispiel kann es auch aufschlussreich sein, sogenannte „schwarze Löcher“ zu identifizieren, also Benutzer, die eine große Menge von eingehenden Aktivitäten zahlreicher anderer Benutzer haben, selbst aber nur selten mit einzelnen Benutzern interagieren. Bei Benutzern mit diesem Verhalten handelt es sich häufig um Prominente mit einer großen Fangemeinde. Unabhängig vom konkreten Ziel wird schnell klar, dass entsprechende Aussagen gerade bei Netzwerken mit einer großen Nutzeranzahl, wie beispielsweise Facebook, nur dann möglich sind, wenn Verfahren der Clusteranalyse eingesetzt werden. Würde man lediglich die unstrukturierte gesamte Datenmenge betrachten, ließen sich nur sehr schwer

Informationen gewinnen, die keine Aussagen über das globale Netzwerk treffen. Es wird ebenso klar, dass es nötig ist, Verfahren zu entwickeln, die innerhalb der Grenzen gegenwärtiger Hardware Ergebnisse liefern, wenn die Eingabedatenmenge sehr große bis gigantische Größenordnungen annimmt.

In dieser Arbeit beschäftigen wir uns unter anderem mit einem weit verbreiteten Clusteringmodell, in dem das Ähnlichkeitsmaß die (quadrierte) geometrische Distanz ist. Die Eingabe besteht demnach aus einer Menge P von Punkten aus dem euklidischen Raum $\mathbb{R}^d$. Neben der Punktmenge P enthält unsere Eingabe zusätzlich eine positive ganze Zahl k, welche die Anzahl an Clustern angibt, in welche die Punkte unterteilt werden sollen. Die eigentliche Aufgabe besteht darin, die Summe der (quadrierten) euklidischen Distanzen der Punkte eines Clusters zu ihrem Clusterzentrum zu minimieren. Dieses Problem ist unter dem Namen „k-means-Problem“ bekannt. Den Algorithmen für das k-means-Problem, die wir betrachten wollen, ist gemein, dass sie eine initiale Auswahl von Clustern und zugehörigen *Zentren* iterativ weiter verbessern, bis eine zumindest lokal optimale Lösung ermittelt wurde.

Da das k-means-Problem schon für $k = 2$ NP-hart ist [ADHP09], sind für praktische Zwecke insbesondere Heuristiken, also Algorithmen ohne feste Schranken für Laufzeit oder Qualität der Lösung, und Approximationsalgorithmen, die eine nichtoptimale Lösung berechnen, welche jedoch garantiert nur um einen gewissen Faktor von der optimalen Lösung abweichen, interessant. Wir betrachten die Konstruktion von *Kernmengen*. Bei Eingabe einer Punktmenge P für das k-means-Problem handelt es sich bei einer Kernmenge um eine kleinere gewichtete Punktmenge S mit der Eigenschaft, dass für *jede* Wahl von k Clusterzentren die Summe der quadrierten Distanzen der Punkte in P zu den Zentren in etwa so groß ist wie die Summe der quadrierten Distanzen der Punkte in S zu den Zentren. Effektiv ist eine Kernmenge also ein hinsichtlich der Clusteringzielfunktion repräsentatives, (teilweise deutlich) kleineres Abbild der ursprünglichen Punktmenge. Üblicherweise bieten Kernmengenkonstruktionen Garantien einer $(1 + \epsilon)$-Approximation, das heißt, dass ein k-means-Clustering auf der Kernmenge höchstens um einen Faktor $(1 + \epsilon)$ schlechter ist als dasselbe Clustering auf der Ursprungsmenge P. Neben der Approximation des k-means-Problems können Kernmengen auch ein effektives Mittel im Umgang mit großen Datenmengen sein. Wenn die Größe der Kernmenge S nur ein kleiner Anteil an der Größe von P ist, können Clustering-Algorithmen auf S deutlich schneller ausgeführt werden als auf P.

Schließlich betrachten wir in dieser Arbeit noch eine weitere Modellierung von

Clusteringproblemen, die mit der geometrischen Auffassung verwandt ist. Für einige Anwendungen ist es sinnvoll, die zu clusternden Objekte als Knoten eines gewichteten Graphen zu betrachten. Das Ähnlichkeitsmaß ist dann nicht mehr die euklidische Distanz, sondern das Gewicht der Kanten zwischen den Objekten beziehungsweise Knoten. Stellt man sich den gezeichneten Graphen vor, kommt das Kantengewicht der „Distanz“ zwischen den Knoten gleich. Bei der Clusteranalyse von Graphen können wir unter anderem auf diverse Ergebnisse aus der Graphentheorie zurückgreifen und zudem Techniken aus der linearen Algebra anwenden, wie wir später aufzeigen werden.

Anwendungen. Die Clusteranalyse war in der Vergangenheit bereits Gegenstand intensiver Forschung, die bis heute regelmäßig neue theoretische Erkenntnisse liefert. Es handelt sich aber um ein Gebiet mit hoher praktischer Relevanz, da Clustering in zahlreichen Anwendungen entweder selbst durchgeführt werden muss, oder als Teilproblem in einem übergeordneten Kontext auftritt. Wir wollen an dieser Stelle einen kurzen und sicherlich unvollständigen Überblick über einige wichtige Anwendungen der bisher vorgestellten Clusteringmodelle geben.

Eine gegenwärtige Anwendung für k-means-Clustering haben wir mit der Analyse sozialer Netzwerke bereits erwähnt. Eine weitere Anwendung, die gerade heute im Bereich der Online-Shops relevant ist, findet das k-means-Clustering in der Marktforschung. Hier sollen Informationen über das Einkaufsverhalten von Kunden genutzt werden, um diese beispielsweise den Marktsegmenten nach zu unterteilen. Je nachdem, wie detailliert die gesammelten Informationen sind, können diese auch genutzt werden, um Produktplatzierung zu optimieren, die Nachfrage nach neuen Produkten abzuschätzen, oder potenzielle neue Märkte zu erkennen. Neben den wirtschafts- und sozialwissenschaftlichen Anwendungsfällen, kommt die Clusteranalyse auch in geographischen und geologischen Zusammenhängen zum Einsatz. Konkret werden hier zum Beispiel chemische Eigenschaften von gesammelten Proben geclustert, um geographische Profile der untersuchten Substanzen zu erstellen.

Kernmengen werden in drei Szenarien eingesetzt. Wie bereits erwähnt bieten viele Kernmengenkonstruktionen eine garantierte Approximationsgüte, das heißt, das anhand der Kernmenge berechnete Clustering weicht für jede beliebige Wahl der Zentren aus $\mathbb{R}^d$ um beispielsweise einen Faktor von höchstens $1 + \epsilon$ von einem Clustering mit denselben Zentren auf der Ursprungsmenge ab. Man spricht in diesem Zusammenhang auch von *starken* Kernmengen. Ist diese Eigenschaft erfüllt, können wir mit einer Kernmenge, die eine deutlich kleinere Größe hat als die Ursprungsmenge, für Approximationsalgorithmen für das k-means-Problem einen

signifikanten *Speedup*, also eine Beschleunigung der Ausführungszeiten, erzielen. Dabei muss selbstverständlich beachtet werden, dass die Konstruktion der Kernmenge selbst ebenfalls Zeit in Anspruch nimmt und nicht zu aufwändig sein darf, wenn ein Speedup beabsichtigt wird.

Ein relativ neues Modell für Algorithmen sind sogenannte *Datenströme*, bei denen die Eingabe nicht initial vollständig vorliegt, sondern kontinuierlich in einem Datenstrom geliefert wird. Algorithmen für Datenströme, also *Datenstromalgorithmen* dürfen die Daten des Stroms nur genau einmal einlesen und sind zusätzlich polylogarithmisch bezüglich der Eingabegröße platzbeschränkt, das heißt, es ist nicht möglich, die gesamte Eingabe zu protokollieren. Beim Clustering in Datenströmen nutzt man aus, dass die Vereinigung einer $(1+\epsilon_1)$-Kernmenge und einer $(1+\epsilon_2)$-Kernmenge eine $(1+\epsilon_1)(1+\epsilon_2)$-Kernmenge ergibt. Die Eingabe wird in kleinere Teile partitioniert, für die jeweils eine Kernmenge berechnet wird. Die Kernmengen werden dann zu einer einzelnen Kernmenge vereinigt. Da jede Vereinigung einen zusätzlichen Fehler verursacht, darf dies nicht zu oft erfolgen. Man setzt dazu die Merge-and-Reduce-Technik ein, die in einer Baum-Struktur bottom-up die Kernmengen einer Partition miteinander vereinigt. Mit der Technik wird insbesondere sichergestellt, dass jeder einzelne Punkt der Eingabe nur in $\log |P|$ vielen Reduktionsschritten verwendet wird, wobei $|P|$ die Anzahl an bisher betrachteten Eingabepunkten ist. Die Anwendung von Kernmengen im Bereich von Datenstromalgorithmen ist Gegenstand gegenwärtiger Forschung und hat bereits gute Ergebnisse erbracht, wie wir später noch sehen werden.

Schließlich können wir Kernmengen mit einer ähnlichen Idee einsetzen, um Clusteringalgorithmen parallel verteilt auszuführen. Wenn die Eingabepunktmenge verteilt vorliegt, berechnen wir auf den Partitionen verteilt oder parallel Kernmengen, übermitteln die berechneten Kernmengen an eine zentrale Instanz und vereinen diese dort zu einer gesamten Kernmenge. Dadurch vermeiden wir es, die gesamte Punktmenge über das Netzwerk übermitteln zu müssen.

Die Clusteranalyse von Graphen findet ihre vielleicht wichtigste Anwendung in der Bildsegmentierung. Sie wird in diesem Kontext eingesetzt, um bei digitalen Bildern Kanten und insbesondere Objekte zu erkennen. Die Kantenerkennung ist eine fundamentale Technik in der digitalen Bildverarbeitung, die als Vorverarbeitung für speziellere Verfahren nötig ist. Das Erkennen von Objekten macht beispielsweise die Erkennung von Gesichtern, Personen oder Orten auf Fotos möglich.

Darüber hinaus ergibt sich eine natürliche Anwendung in der Identifikation relevanter Strukturen und insbesondere der Konnektivitätsanalyse in Netzwerken. Auch die Entdeckung von häufigen Anrufmustern in der Telekommunikation

kann hier von Interesse sein. Schließlich kann es in Netzwerken mit dynamischer Topologie, wie beispielsweise *ad-hoc-Netzwerken*, die in intelligenten Umgebungen mit vielen wechselnden mobilen Geräten vorkommen, hilfreich sein, zu clustern. Lokales Clustering wird hier unter anderem eingesetzt, um Routing-Algorithmen zu verbessern. Das Netzwerk wird dabei als Graph aufgefasst.

Zudem kann die Clusteranalyse von Graphen eingesetzt werden, um Caching-Mechanismen in Datenbanken zu optimieren. Konkret werden in Datenbanksystemen die einzelnen Einträge in Form von *Seiten* auf dem Speichermedium abgelegt, von denen einige im Hauptspeicher gecached werden. Ein gutes Clustering führt in diesem Kontext dazu, dass das Laden eines Clusters, dem ein angefragtes Datum zugeordnet ist, zudem relevante Daten liefert, die für ähnliche zukünftige Anfragen erforderlich sind.

Verwandte Arbeiten. Der Algorithmus von Lloyd [Llo82] ist eine schon in den Fünfzigerjahren entwickelte Heuristik für das k-means-Problem, die bis heute von praktischer Relevanz ist, da sie im Allgemeinen gute Clusterings produziert, obwohl der Algorithmus keine Approximationsgüte garantiert. Wir betrachten in dieser Arbeit insbesondere die Verbesserung von Lloyds Algorithmus durch Arthur und Vassilvitskii [AV07], die eine $\mathcal{O}(\log k)$-Approximation für das k-means-Problem ist.

Das heutige Konzept von *starken* Kernmengen wurde in [HM04] vorgestellt. Wir beschäftigen uns in dieser Arbeit hauptsächlich mit der Konstruktion aus [FSS13, Sch14].

Das Clusteringmodell in Graphen kann bis zu [KL70] zurückverfolgt werden. Die erste Anwendung von sogenannten *spektralen* Methoden für die Clusteranalyse von Graphen und insbesondere die Anwendung im Bereich der Bildsegmentierung stammt von [SM00]. Eine wichtige Weiterentwicklung des Ansatzes geht zurück auf [NJW01]. Die unmittelbare Äquivalenz zum geometrischen k-means-Problem wurde in [DGK04, DGK07] gezeigt. Wir werden sehen, dass diese für unsere Zwecke besonders wichtig ist.

In [Swi08] wird ebenfalls unter Einsatz der Kernel Methode anhand einer Kernmengenkonstruktion das Graphpartitionierungsproblem untersucht. Diese Arbeit ist von [Swi08] folgendermaßen abzugrenzen: zunächst wird in dieser Arbeit explizit die Kernel Methode auf den k-means++-Algorithmus angewandt. Wir zeigen insbesondere formal, dass die darin umfassten Distanzberechnungen im Kernel-Raum möglich sind. Zudem beweisen wir, dass auch die Kernel-Variante Kernel-k-means++ eine $\mathcal{O}(\log k)$-Approximation für das k-means-Problem ist. Weiterhin nutzen wir den Kernel-k-means++-Algorithmus, um das Graclus-Framework zu erweitern. In [Swi08] wurde die Kernmengenkonstruktion von Stream-KM++ für die Clusteranalyse von

Graphen verwendet. Wir setzen hingegen die Konstruktion von [FSS13] ein und wenden zudem die Kernel Methode auf diese an. Zudem haben wir die Konstruktion von [FSS13] implementiert und experimentell evaluiert.

Die Arbeit ist folgendermaßen gegliedert. In Kapitel 2 definieren wir die wichtigsten Begriffe und Konzepte, die wir im Rahmen dieser Arbeit benötigen. Außerdem stellen wir die Algorithmen vor, die wir als Grundlage verwenden oder weiterentwickeln beziehungsweise verbessern wollen. In Kapitel 3 entwickeln wir zum einen eine Verbesserung des Algorithmus von Dhillon, Guan und Kulis [DGK04, DGK07] und zeigen zum anderen, mit welchen Modifikationen die Kernmengenkonstruktion von Feldman, Sohler und Schmidt [FSS13, Sch14] praktisch umsetzbar wird. Die entwickelten Algorithmen wollen wir dann in Kapitel 4 anhand einer Reihe von Experimenten empirisch bewerten. Unsere Ergebnisse fassen wir schließlich in Kapitel 5 zusammen.

2 Grundlegende Definitionen und Algorithmen

In diesem Kapitel definieren wir die für unsere Zwecke relevanten Begriffe im Kontext der Clusteranalyse und führen die wichtigen grundlegenden Algorithmen ein, deren Ideen für uns im Folgenden noch von Bedeutung sein werden. Wir gehen dabei nach Themengebieten geordnet vor: In Abschnitt 2.1 skizzieren wir kurz das Themengebiet der Clusteranalyse, definieren die üblichen Zielfunktionen und stellen zwei bedeutende Algorithmen vor. Abschnitt 2.2 führt kurz in die Graphentheorie sowie die Clusteranalyse von Graphen ein. In diesem Abschnitt werden wir zudem verschiedene Optimierungskritierien für die Clusteranalyse von Graphen herausstellen. Schließlich fassen wir in Abschnitt 2.3 die wichtigsten Methoden und Algorithmen aus dem Bereich der spektralen Clusteranalyse zusammen und stellen zudem die wesentlichen Konzepte von Kernel-Methoden vor.

2.1 Clustering und k-means

Clusteranalyse oder „Clustering“ beschäftigt sich mit der Einteilung von Objekten in Gruppen („Cluster“), sodass sich die Objekte innerhalb eines Clusters gemäß eines bestimmten Optimierungskriteriums ähnlich sind und von Objekten eines anderen Clusters unterscheiden. Es existieren zahlreiche grundsätzlich verschiedene Ansätze, um Clusteringprobleme zu lösen. Wir beschränken uns in dieser Arbeit auf *partitionierende* Clusteringprobleme und -verfahren. Bei diesen soll eine Menge von Objekten, welche der erste Teil der Eingabe ist, gemäß einer Cluster-Zielfunktion möglichst optimal in genau k Cluster unterteilt werden. Dabei ist k der ganzzahlige zweite Teil der Eingabe, das heißt, die Anzahl an Clustern muss bei diesen Verfahren vorab festgelegt werden. Die initial gewählten k Zentren werden dann iterativ verschoben, bis ein gewisses Abbruchkriterium erfüllt ist.

Für die Zielfunktion, welche die Nähe oder Ferne von Punkten zueinander quantifiziert, sind bei Eingabepunkten aus $\mathbb{R}^d$ Metriken naheliegend. Intuitiv ist dabei die euklidische Distanz, auf der die beiden bekanntesten Clustering-Problemstellungen basieren. Wir notieren die euklidische Distanz zwischen zwei Punkten $x, y \in \mathbb{R}^d$ mit $\|x - y\|$.

Definition 2.1.1 (k-median und k-means). Sei $P \subset \mathbb{R}^d$ und $k \in \mathbb{N}^+$. Das k-median-Problem besteht darin, eine Menge von k (Cluster-)*Zentren* $C = \{c_1, \ldots, c_k\}$ mit $c_i \in \mathbb{R}^d$ zu finden, sodass der folgende Term minimal wird:

$$\sum_{p \in P} \min_{c \in C} \|p - c\|$$

Das k-means-Problem unterscheidet sich nur darin, dass bei diesem die Summe der *quadrierten* euklidischen Distanzen zum jeweils nächstgelegenen Zentrum minimiert werden soll, das heißt, dass der folgende Term minimiert werden soll:

$$\sum_{p \in P} \min_{c \in C} \|p - c\|^2$$

Beim *gewichteten* k-means-Problem werden den Eingabepunkten zusätzlich mit einer Funktion $w : P \to \mathbb{R}$ Gewichte zugewiesen. Die zu minimierende Zielfunktion lautet dann entsprechend

$$\sum_{p \in P} \min_{c \in C} w(p)\|p - c\|^2$$

Sowohl das k-Median-Problem [MS84] als auch das k-means-Problem [ADHP09] sind optimal NP-schwer lösbar. Typischerweise werden zur Lösung daher approximative oder heuristische Algorithmen eingesetzt. Die bekannteste und bis heute sehr erfolgreiche Heuristik für das k-means-Problem ist der Algorithmus von Lloyd [Llo82]. Wegen seiner großen Popularität wird der Algorithmus häufig auch als „k-means-Algorithmus“ bezeichnet. Der Algorithmus wählt initial k zufällige Punkte aus der Eingabemenge als initiale Clusterzentren. Darüber hinaus existieren zahlreiche weitere Methoden zur initialen Wahl der Zentren, siehe dazu beispielsweise [CKV13]. Anschließend wird jedem Punkt das am nächsten gelegene Zentrum zugewiesen. Dadurch entstehen die initialen Cluster mit ihren jeweiligen Zentren. Im zweiten Schritt wird das neue Zentrum eines jeden Clusters als der geometrische Zentroid des Clusters gewählt. Die Wahl des geometrischen Zentroiden als neues Zentrum ist für das 1-means-Problem optimal [ORSS12]. Die Zuweisung von Punkten zum nächstgelegenen Clusterzentrum und die Berechnung der neuen Zentroiden werden solange alterniert, bis die Lösung konvergiert, also wenn sich die Zuordnungen der Punkte nicht mehr ändern. In der Praxis wird gelegentlich auch nach einer festen Anzahl von Iterationen terminiert.

Algorithmus 1: Algorithmus von Lloyd

Eingabe: $: P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+$

Ausgabe: $: k$-means-Clustering von P

Wähle zufällig k Zentren $c_1^{(0)}, \ldots, c_k^{(0)}$ aus P

$S_i^{(0)} \leftarrow \{p \in P : \|p - c_i^{(0)}\|^2 \leq \|p - c_{i'}^{(0)}\|^2 \, \forall \, i' \in \{1, \ldots, k\}\}$

repeat

  $c_i^{(t)} \leftarrow \frac{1}{|S_i^{(t-1)}|} \sum_{p_j \in S_i^{(t-1)}} p_j$

  $S_i^{(t)} \leftarrow \{p \in P : \|p - c_i^{(t)}\|^2 \leq \|p - c_{i'}^{(t)}\|^2 \, \forall \, i' \in \{1, \ldots, k\}\}$

until $S_i^{(t)} = S_i^{(t-1)}$

Die asymptotische Laufzeit des Algorithmus beträgt $\mathcal{O}(nkdi)$, wobei i die Anzahl an durchgeführten Iterationen ist. Wenn der Algorithmus konvergiert und nicht durch eine feste Anzahl von Iterationen terminiert wird, wurde ein lokales Optimum gefunden, welches jedoch im Allgemeinen kein globales Optimum oder eine Approximation eines globalen Optimums ist.

Der Algorithmus k-means++ [AV07] sieht eine alternative Wahl der initialen Clusterzentren in Lloyds Algorithmus vor: er wählt diese auf einfache, aber dennoch geschickte Art und Weise und führt anschließend die übrigen Schritte von Lloyds Algorithmus aus. Dazu wird zunächst ein einzelnes Clusterzentrum c_1 zufällig gleichverteilt aus der Eingabe-Punktmenge P gewählt. Alle weiteren Clusterzentren werden sukzessive nach der folgenden Vorschrift gewählt, bis insgesamt k Zentren gewählt wurden. Im Weiteren bezeichnen wir mit $D(x)$ für einen Punkt x aus der Eingabe-Punktmenge P die geringste Distanz von x zum nächstgelegenen bereits gewählten Zentrum. In jeder Iteration wird als nächstes Zentrum c_i der Punkt $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$ mit Wahrscheinlichkeit $\frac{D(x')^2}{\sum_{x \in P} D(x)^2}$ gewählt.

Algorithmus 2: k-means++

Eingabe: $: P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+$

Ausgabe: $: k$ initiale Clusterzentren für P

1 Wähle c_1 zufällig gleichverteilt aus P
2 **for** $i \leftarrow 2$ **to** k **do**
3 Wähle den Punkt $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$ als Zentrum c_i mit Wahrscheinlichkeit $\frac{D(x')^2}{\sum_{x \in P} D(x)^2}$
4 Führe Lloyds Algorithmus mit den initialen Clusterzentren $c_1, \ldots, c_k$ aus.

Die k Zentren, die von k-means++ ausgewählt werden, sind eine $\mathcal{O}(\log k)$-Approximation für das k-means-Problem, die durch die anschließende Ausführung von Lloyds Algorithmus noch zu einem lokalen Optimum verbessert werden.

In der Einleitung haben wir bereits das Konzept der (starken) Kernmenge erwähnt. Diese ist formal folgendermaßen definiert.

Definition 2.1.2 (Starke Kernmenge [HM04, Sch14]). Sei $P \subset \mathbb{R}^d$ eine endliche Punktmenge und sei $w_1 : P \to \mathbb{R}$ eine Gewichtsfunktion, die jedem Punkt in P ein Gewicht zuordnet. Sei weiterhin $\epsilon \in (0, 1)$. Eine endliche Menge $S \subset \mathbb{R}^d$ und eine Gewichtsfunktion $w_2 : S \to \mathbb{R}$ bilden eine *starke* $(1 - \epsilon)$-Kernmenge für

P, wenn für alle Mengen von k Zentren $C \subset P$ gilt:

$$(1-\epsilon)\sum_{x\in P} w_1(x)\min_{c\in C}\|x-c\|^2 \leq \sum_{y\in S} w_2(y)\min_{c\in C}\|y-c\|^2 \leq (1+\epsilon)\sum_{x\in P} w_1(x)\min_{c\in C}\|x-c\|^2$$

Im nächsten Abschnitt betrachten wir das Clusteringproblem auf Graphen, das wir später noch genauer untersuchen wollen.

2.2 Graphen und Clusteranalyse von Graphen

Der *Graph* ist in der Informatik eine vielseitig einsetzbare und gut erforschte Datenstruktur. Wir können diverse Clusteringprobleme nicht nur für eine Eingabepunktmenge formulieren, sondern auch für eine Eingabe, die aus einem Graphen besteht. Wir beginnen mit der folgenden Definition des Graphen.

Definition 2.2.1 (Graph). Sei V eine endliche Menge und $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. Dann heißt das Tupel $G = (V, E)$ ein (endlicher) *Graph* mit *Knotenmenge* V und *Kantenmenge* E. Ist $e = \{u, v\} \in E$, dann sagen wir, dass die Kante e des Graphen G die Knoten u und v *verbindet*. In diesem Fall sind u und v die *Endknoten* von e. Ein Knoten $u \in V$ und eine Kante $e \in V$ heißen *inzident* genau dann, wenn $u \in e$. Wir sagen, dass u und ein weiterer Knoten $v \in V$ *adjazent* sind genau dann, wenn es eine Kante $e' = \{u, v\}$ in E gibt. Typischerweise bezeichnet $n = |V|$ die *Knotenzahl* von G und $m = |E|$ die *Kantenzahl* von G.

Bei einem 3-Tupel $G' = (V', E', w')$ mit $w' : E \to \mathbb{N}$ spricht man von einem *gewichteten* Graphen, dessen Kanten über ein Gewicht verfügen, das von der Gewichtsfunktion w' abgebildet wird.

Es gibt zwei grundlegende Datenstrukturen für Graphen: Bei den sogenannten *Adjazenzlisten* wird für jeden Knoten v im Graphen eine Liste Adj_v gehalten, in der für jeden zu v inzidenten Knoten ein Eintrag in Adj_v enthalten ist, der den entsprechenden adjazenten Knoten referenziert, sowie gegebenenfalls das Gewicht der Kante zwischen den beiden Knoten. Alternativ wird in einer *Adjazenzmatrix* Adj^G der Größe $|V| \times |V|$ an der Stelle $Adj^G_{u,v}$ das Gewicht der Kante zwischen den Knoten u und v eingetragen, sofern die beiden Knoten durch eine Kante miteinander verbunden sind. Anderenfalls wird zumeist -1 oder 0 eingetragen. Bei ungewichteten Graphen wird dementsprechend lediglich 0 oder 1 eingetragen.

Wenn die Eingabe keine Punktmenge, sondern ein (gewichteter) Graph ist, können wir die Ideen der bereits vorgestellten Verfahren weiterhin anwenden. Wir interessieren uns in diesem Fall für die „Ähnlichkeit“ von Knotenmengen im Graphen.

Definition 2.2.2 (Graph-Schnitt). Sei $G = (V, E, w)$ ein gewichteter Graph. Ein *Schnitt* $C = (S, T)$ von G ist eine Partitionierung von V in die beiden Mengen S und T, das heißt, dass $V = S \cup T$ und gleichzeitig $S \cap T = \emptyset$. Die *Schnittmenge* von C sind die Kanten in E, die einen Endpunkt in S und den anderen Endpunkt in T haben, das heißt, formal ist die Schnittmenge definiert als

$$\{(u, v) \in E \mid u \in S, v \in T\}.$$

Das Gewicht oder der Wert eines Schnittes ist die Summe der Kantengewichte der Schnittmenge. Wir verwenden die folgende Notation:

$$w_{cut}(S, T) = \sum_{u \in S, v \in T} w((u, v))$$

Falls der Graph in Form einer Adjazenzmatrix Adj^G vorliegt, lautet die Berechnungsvorschrift entsprechend:

$$w_{cut}(S, T) = \sum_{u \in S, v \in T} Adj^G_{u,v}$$

Für eine direkte Analogie zum k-means-Problem wäre ein Schnitt- beziehungsweise Partitionierungs-Begriff wünschenswert, der eine k-fache Partitionierung der Knotenmenge erlaubt. Diese lautet wie folgt.

Definition 2.2.3 (k-Graphpartitionierung). Sei $G = (V, E, w)$ ein gewichteter Graph. Für zwei Mengen $A, B \subseteq V$ definieren wir:

$$w(A, B) = \sum_{u \in A, v \in B} w((u, v))$$

Das k-Graphpartitionierungsproblem besteht darin, eine Partitionierung der Knotenmenge V in k disjunkte Teilmengen $V_1, \ldots, V_k$ mit $\bigcup_{i \in \{1,\ldots,k\}} V_i = V$ zu ermitteln, sodass sich die Knoten innerhalb einer Partition bezüglich einer Ähnlichkeitsrelation möglichst ähnlich sind und sich die Knoten unterschiedlicher Partitionen bezüglich der Ähnlichkeitsrelation möglichst stark voneinander unterscheiden [KL70].

Beim Clustering von Punktmengen lagen für die Unähnlichkeitsrelation Metriken nahe, im Falle der Graphpartitionierung existiert eine Reihe von Optimierungskriterien, von denen wir im Folgenden die verbreitetsten einführen.

1. **Ratio Association**. Bei der Ratio Association sollen die Intra-Clusterabstände

relativ zur jeweiligen Clustergröße maximiert werden:

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|}$$

2. **Ratio Cut**. Beim Ratio Cut wird das Gewicht des Schnitts zwischen jeweils einem Cluster und allen anderen Punkten im Graphen minimiert:

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{|V_c|}$$

3. **Kernighan-Lin**. Bei dem in [KL70] vorgestellten Optimierungskriterium werden die Intra-Clusterabstände ähnlich der Ratio Association minimiert, allerdings müssen alle Partitionen hier zusätzlich dieselbe Größe haben. Die hier vorgestellte Zielfunktion ist von 2 auf k Partitionen verallgemeinert:

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{|V_c|} \text{ s.t. } |V_c| = \frac{|V|}{k} \, \forall \, c \in \{1, \ldots, k\}$$

4. **Normalized Cut**. Ziel des Normalized Cut ist wie beim Ratio Cut die Minimierung des Schnitts von einem Cluster mit den übrigen Punkten im Graphen, jedoch relativ zum Schnitt des Clusters mit *allen* Knoten im Graphen. Der Normalized Cut oder kurz NCut ist in der Bild-Segmentierung recht verbreitet [SM00]. Wir betrachten hier ebenfalls die auf k Partitionen verallgemeinerte Zielfunktion.

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{w(V_c, V)}$$

Im nächsten Abschnitt betrachten wir zwei Techniken, mit denen Limitierungen hinsichtlich der Qualität der erzielten Clusterings sowie der Komplexität der Berechnungen verbessert werden können.

2.3 Kernel-Methoden und spektrales Clustering

Legt man die k-means Zielfunktion zu Grunde, können im Ursprungsraum $\mathbb{R}^d$ nur Cluster berechnet werden, die *linear separierbar* sind. Ein illustratives Beispiel im $\mathbb{R}^2$ sind zwei konzentrisch angeordnete ringförmige Punktmengen. Die intuitiv optimalen Cluster sind die beiden jeweiligen Ringe, es ist jedoch im $\mathbb{R}^2$ nicht

möglich, Clusterzentren anzugeben, die den zweidimensionalen Raum in Ringe partitionieren. Die Beschränkung liegt darin, dass wir die Partitionierung oder Separierung durch Hyperebenen vornehmen. Im Ursprungsraum können wir daher nur linear separierbare Cluster bestimmen. Wir betrachten nun ein Verfahren, mit dem sich diese Beschränkung umgehen lässt.

2.3.1 Kernel-k-means

Unsere fundamentale Motivation für diesen Abschnitt ist Covers Theorem [Cov65], welches informell zusammengefasst besagt, dass eine zufällige Eingabepunktmenge im d-dimensionalen Raum mit hoher Wahrscheinlichkeit in einen höherdimensionalen Raum (zum Beispiel der Dimension $D \gg d$) abgebildet linear separierbar ist.

Für Klassifizierungs- und insbesondere Clusteringprobleme benötigen wir sowohl im Ursprungsraum als auch im abgebildeten Raum ein algebraisches Maß für Unähnlichkeit. Üblicherweise wird dabei eine auf *Skalarprodukten* basierende Metrik eingesetzt. Wir beziehen uns im Weiteren auf die folgende Definition von *euklidischen Vektorräumen*, also Vektorräumen mit einem reellen Skalarprodukt.

Definition 2.3.1 (Euklidischer Vektorraum). Sei V ein Vektorraum über $\mathbb{R}$. Ein reelles Skalarprodukt auf V ist eine Abbildung $\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$, die den folgenden Eigenschaften genügt:

1. $\langle a, b \rangle = \langle b, a \rangle \, \forall \, a, b \in V$
2. $\langle \lambda a + \mu b, c \rangle = \lambda \langle a, c \rangle + \mu \langle b, c \rangle \, \forall \, a, b, c \in V$ und $\lambda, \mu \in \mathbb{R}$
3. $\langle a, a \rangle \geq 0$ und $\langle a, a \rangle = 0 \Leftrightarrow a = 0 \, \forall \, a \in V$

Ein $\mathbb{R}$-Vektorraum mit einem reellen Skalarprodukt heißt *euklidischer Vektorraum*.

Die k-means-Zielfunktion kann auf euklidische Vektorräume erweitert werden, indem wir für einen euklidischen Vektorraum $\mathcal{V}$ eine Norm $\|\cdot\| : \mathcal{V} \to \mathbb{R}$ über das Skalarprodukt gemäß der folgenden Abbildungsvorschrift für jedes $\chi \in \mathcal{V}$ definieren: $\|\chi\| = \sqrt{\langle \chi, \chi \rangle}$. Das auf Skalarprodukten basierte Analog zu Definition 2.1.1 des k-means-Problems lautet dann wie folgt:

Definition 2.3.2 (k-means in euklidischen Vektorräumen [Sch14]). Sei $\mathcal{V}$ ein euklidischer Vektorraum mit der Norm $\|\cdot\| : \mathcal{V} \to \mathbb{R}$ und sei $\mathcal{P} \subset \mathcal{V}$ sowie $k \in \mathbb{N}^+$. Das k-means-Problem in $\mathcal{V}$ besteht darin, eine Menge von k Clusterzentren $\mathcal{C} = \{c_1, \ldots, c_k\}$ mit $c_i \in V$ zu finden, sodass der folgende Term minimal wird:

$$\sum_{p \in \mathcal{P}} \min_{c \in \mathcal{C}} \|p - c\|^2$$

Die gewichtete Variante des k-means-Problems in euklidischen Vektorräumen lautet entsprechend wie folgt:

$$\sum_{p \in \mathcal{P}} \min_{c \in \mathcal{C}} w(p) \|p - c\|^2$$

Damit ist es nun möglich, ein intuitives Verfahren zum Bestimmen von Clusterings mit nicht-linearen Separatoren anzugeben. Wir bilden zunächst mit einer Abbildung $\phi : \mathcal{X} \to \mathcal{V}$ unsere Eingabepunkte aus ihrem Ursprungsraum $\mathcal{X}$ (also beispielsweise $\mathbb{R}^d$) auf einen D-dimensionalen (üblicherweise $D \gg d$) euklidischen Vektorraum $\mathcal{V}$ ab, und berechnen in diesem mit Lloyds Algorithmus oder einem anderen Clusteringverfahren ein Clustering, welches im Ursprungsraum $\mathcal{X}$ im Allgemeinen nicht-linear separierten Clustern entspricht. Außerdem ist es mit einem „Trick“, den wir im Folgenden erläutern, möglich, implizit auf einen unendlich dimensionalen Vektorraum (einen Funktionenraum), abzubilden. Bei diesem Verfahren wird entsprechend der folgende Term minimiert:

$$\sum_{p \in P} \min_{c \in C} \|\phi(p) - c\|^2$$

Die Zentroid-Berechnung zur Bestimmung eines Clusterzentrums c_i im Sinne von Lloyds Algorithmus lautet dann

$$c_i = \frac{\sum_{p \in S_i} \phi(p)}{|S_i|}$$

wobei S_i die Menge aller dem Clusterzentrum c_i zugeordneten Punkte ist. Zunächst haben wir uns damit nicht-linear separierte Cluster „erkauft“, der Preis, den wir dafür zahlen, besteht jedoch in höherem Rechenaufwand, da wir einerseits die Punkte in den jeweils höherdimensionalen Raum abbilden müssen und andererseits die Berechnung der (euklidischen) Distanzen aufwändiger wird; diese benötigt im Ursprungsraum Zeit $\mathcal{O}(d)$ und im höherdimensionalen Raum $\mathcal{O}(D)$, sofern dieser endlich dimensional ist. Der Mehraufwand für die Distanzberechnung fällt dabei stärker ins Gewicht, da wir die Distanzberechnung beispielsweise bei Lloyds Algorithmus in jeder Schleifeniteration durchführen.

Dieses Problem wurde erstmals im Zusammenhang mit *Support Vector Machines* (kurz SVMs) gelöst [BGV92]. Die Lösung mittels des sogenannten *Kernel-Tricks* kann in den Clustering-Kontext übertragen werden [Sch14]. Wir skizzieren dies kurz. Zunächst halten wir eine wichtige Beobachtung hinsichtlich der (quadrierten) euklidischen Distanz $\|\phi(p) - c\|^2$ zwischen dem abgebildeten Punkt und einem Clusterzentrum fest. Diese lässt sich wie folgt ausschließlich über Skalarprodukte berechnen (wobei auch hier wieder S_c die dem Clusterzentrum c zugeordnete

Punktmenge ist):

$$\begin{aligned}
\|\phi(p) - c\|^2 &= \left\|\phi(p) - \frac{1}{|S_c|}\sum_{x\in S_c}\phi(x)\right\|^2 \\
&= \left\langle \phi(p) - \frac{1}{|S_c|}\sum_{x\in S_c}\phi(x), \phi(p) - \frac{1}{|S_c|}\sum_{x\in S_c}\phi(x)\right\rangle \\
&= \langle\phi(p),\phi(p)\rangle - \frac{1}{|S_c|}\sum_{x\in S_c}\langle\phi(p),\phi(x)\rangle - \frac{1}{|S_c|}\sum_{x\in S_c}\langle\phi(x),\phi(p)\rangle \\
&\quad + \frac{1}{|S_c|^2}\sum_{x,y\in S_c}\langle\phi(x),\phi(y)\rangle \qquad (2.1)
\end{aligned}$$

Diese Beobachtung allein hilft uns noch nicht weiter, da auch die Berechnung des Skalarproduktes im höherdimensionalen Raum einen asymptotischen Aufwand von $\mathcal{O}(D)$ hätte. An dieser Stelle kommt nun der in [BGV92] vorgestellte *Kernel-Trick* zum Tragen. Durch den Einsatz von sogenannten *Kernelfunktionen* lässt sich dieser Mehraufwand umgehen. Dabei handelt es sich um positiv definite Abbildungen $\kappa : X \times X \to \mathbb{R}$, die bei Eingabe von zwei Punkten aus dem Ursprungsraum X das Skalarprodukt der in den höherdimensionalen Raum abgebildeten Punkte berechnen:

$$\kappa(x,y) = \langle\phi(x),\phi(y)\rangle$$

Der Kernel-Trick besteht also darin, dass sich die Distanzberechnung auf Skalarprodukte von Eingabepunkten reduzieren lässt, für die keine explizite Kenntnis der Abbildung ϕ oder der abgebildeten Punkte $\phi(p)$ nötig ist. Dies können wir umsetzen, indem wir insgesamt $\mathcal{O}(n^2)$ Skalarprodukte berechnen oder direkt eine Kernelfunktion κ verwenden. Mit der Kernelfunktion κ können wir die Distanzberechnung aus 2.1 weiter vereinfachen:

$$\|\phi(p) - c\|^2 = \kappa(p,p) - \frac{1}{|S_c|}\sum_{x\in S_c}\kappa(p,x) - \frac{1}{|S_c|}\sum_{x\in S_c}\kappa(x,p) + \frac{1}{|S_c|^2}\sum_{x,y\in S_c}\kappa(x,y) \qquad (2.2)$$

Tabelle 1 bietet eine Übersicht über häufig eingesetzte Kernelfunktionen im Maschinellen Lernen [HSS08]. Insbesondere die Gauss-Kernelfunktion und die Polynom-Kernelfunktion werden auch bei Support-Vektor-Maschinen gerne verwendet.

Eine auf Kernelfunktionen basierende Variante von (Lloyds) Algorithmus 1 folgt mit 2.2 unmittelbar. Der Algorithmus wird üblicherweise kurz „Kernel-k-means-

Linear-Kernel	$\kappa(x_i, x_j) = x_i \cdot x_j + \gamma$
Polynom-Kernel	$\kappa(x_i, x_j) = (x_i \cdot x_j + \gamma)^{\delta}$
Gauss-/RBF-Kernel	$\kappa(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$
Sigmoid-Kernel	$\kappa(x_i, x_j) = \tanh(\alpha(x_i \cdot x_j) + \theta)$

Tabelle 1: Beispiele für häufig eingesetzte Kernelfunktionen im Maschinellen Lernen [HSS08].

Algorithmus 3: Kernel-k-means

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+, \kappa : X \times X \to \mathbb{R}$, initiales Clustering $C_1, \ldots, C_k$
Ausgabe: : k-means-Clustering von P

```
1 t ← 0
2 repeat
3     Berechne ‖φ(p) − c_i^(t)‖² für alle p ∈ P und i ∈ {1, ..., k} mit 2.2
4     foreach p ∈ P do
5         c*(p) ← arg min_i ( ‖φ(p) − c_i^(t)‖ )
6     for i ← 1 to k do
7         C_i^t ← {p | c*(p) = i}
8     t ← t + 1
9 until Konvergenz oder t > t_max
```

Algorithmus“ genannt. Der Vorfaktor für gewichtetes k-means hat auf die Berechnungsvorschriften keine weiteren Auswirkungen, Algorithmus 3 kann dementsprechend für gewichtetes k-means erweitert werden. Der Algorithmus lässt sich mit einer asymptotischen Laufzeit von $\mathcal{O}(n^2(\tau + d))$ implementieren [DGK04, DGK07], wobei τ die Anzahl an Iterationen der äußeren Schleife ist.

Im nächsten Unterabschnitt führen wir Verfahren ein, die mit Techniken aus der linearen Algebra das Graphclusteringproblem lösen und in Kombination mit dem hier vorgestellten Algorithmus eine robuste Grundlage für unsere Zwecke bilden.

2.3.2 Spektrale Clustering Methoden

Beim spektralen Clustering besteht die Eingabe aus einem Graphen, dessen Knoten geclustert werden sollen. Informationen über die „Ähnlichkeit“ von Objekten werden in Form eines Kantengewichts der Kante zwischen den jeweiligen Objekten realisiert. Wir werden im Weiteren insbesondere Eigenvektoren und zugehörige Eigenwerte von Matrizen betrachten und rufen uns daher kurz die Definition dieser in Erinnerung.

Definition 2.3.3 (Eigenvektor und Eigenwert). Sei A eine quadratische Matrix. Das *Eigenwertproblem* besteht darin, eine Zahl λ und einen dazugehörigen Vektor

$\vec{x} \neq \vec{0}$ zu finden, für die gilt:

$$A\vec{x} = \lambda\vec{x}$$

Die reelle oder komplexe Zahl λ heißt *Eigenwert* von A, der Vektor $\vec{x}$ heißt Eigenvektor von A.

Wenn G der Graph ist, durch den die Eingabemenge repräsentiert wird, dann ist die Adjazenzmatrix Adj^G des Graphen der Ausgangspunkt der spektralen Clusteranalyse. Bevor wir mit konkreten Verfahren der spektralen Clusteranalyse beginnen können, benötigen wir einige grundlegende Definitionen der (spektralen) Graphentheorie.

Definition 2.3.4 (Grad-Matrix). Sei $G = (V, E)$ ein Graph mit $|V| = n$ Knoten. Die *Grad-Matrix* von G ist die $n \times n$-Diagonalmatrix D, die folgendermaßen definiert ist:

$$D_{i,j} = \begin{cases} \deg(v_i) & \text{falls } i = j, \\ 0 & \text{sonst} \end{cases}$$

Dabei ist $\deg(v_i)$ die Anzahl an Kanten, die zum Knoten v_i inzident sind. Diese Anzahl wird auch *Grad* des Knotens v_i genannt.

Mit Hilfe der Grad-Matrix eines Graphen lässt sich die sogenannte *Laplace-Matrix* des Graphen berechnen, welche eine Matrix-Repräsentation von Graphen ist, die aufschlussreiche Informationen über die Struktur und Beschaffenheit des Graphen erlaubt. Sie ist wie folgt definiert:

Definition 2.3.5 (Laplace-Matrix [Lux07]). Sei $G = (V, E)$ ein Graph ohne Schleifen mit Grad-Matrix D und (gewichteter) Adjazenzmatrix Adj^G. Die (nicht-normalisierte) *Laplace-Matrix* L von G ist definiert als:

$$L = D - Adj^G$$

Die Einträge der Matrix sind dementsprechend folgendermaßen definiert:

$$L_{i,j} = \begin{cases} \deg(v_i) & \text{falls } i = j, \\ -1 & \text{falls } i \neq j \text{ und } v_i \text{ inzident zu } v_j \text{ ist}, \\ 0 & \text{sonst} \end{cases}$$

Es existieren zwei normalisierte Formen der Laplace-Matrix, die *normalisierte*

Laplace-Matrizen genannt werden. Sie sind wie folgt definiert:

$$\mathcal{L}_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$
$$\mathcal{L}_{rw} = D^{-1} L$$

Die Eigenwerte der Laplace-Matrizen eines Graphen geben beispielsweise Aufschluss über die *Zusammenhangskomponenten* eines Graphen. Eine Zusammenhangskomponente definiert ist als Teilgraph, in dem jedes Paar von Knoten über einen Pfad miteinander verbunden ist. Das folgende Lemma illustriert die Zusammenhangsinformationen, die sich aus der Laplace-Matrix ablesen lassen:

Lemma 2.3.1 (Zusammenhang und Laplace-Matrix [Lux07]). Gegeben sei $G = (V, E)$ ein schleifenloser Graph mit nicht-negativen Gewichten und Laplace-Matrix L. Die Anzahl an Zusammenhangskomponenten von G ist gleich der Anzahl an Eigenwerten von L, die gleich Null sind.

Da die Eigenwerte einer Matrix auch „Spektrum“ genannt werden und man bei den hier vorgestellten Methoden die Eigenwerte der Laplace-Matrix nutzt, wird diese Technik „spektrale Clusteranalyse“ genannt.

Die wichtigsten Algorithmen, die auf dem Spektrum der Laplace-Matrix der Eingabe basieren, wollen wir im Folgenden vorstellen. Wir beginnen mit den Algorithmen 4 und 5, welche die normalisierten Laplace-Matrizen verwenden und in den jeweils zitierten Papieren vorgestellt wurden.

Algorithmus 4: Nicht-normalisiertes spektrales Clustering [Lux07]

Eingabe: : Gewichtete Adjazenzmatrix $Adj^G \in \mathbb{R}^{n \times n}, k \in \mathbb{N}^+$
Ausgabe: : k Cluster $C_1, \ldots, C_k$

1 Berechne die nicht-normalisierte Laplace-Matrix L des Graphen G mit Adj^G
2 Bestimme Eigenvektoren $u_1, \ldots, u_k$ zu den k kleinsten Eigenwerten von L
3 Sei $U \in \mathbb{R}^{n \times k}$ die Matrix mit $u_1, \ldots u_k$ als Spaltenvektoren
4 Für $i \leftarrow \{1, \ldots, n\}$ sei $y_i \in \mathbb{R}^k$ der Vektor der i-ten Reihe von U
5 Clustere die Punkte $(y_i)_{i \in \{1,\ldots,n\}}$ mit einem k-means-Algorithmus in $C_1, \ldots, C_k$

Der in [SM00] vorgestellte Algorithmus sieht in seiner ursprünglichen Form zunächst eine Bipartitionierung anhand des Eigenvektors mit dem zweitkleinsten Eigenwert vor, die anschließend gegebenenfalls weiter partitioniert wird. Dieses

Algorithmus 5: Normalisiertes spektrales Clustering [SM00]

Eingabe: : Gewichtete Adjazenzmatrix $Adj^G \in \mathbb{R}^{n \times n}, k \in \mathbb{N}^+$
Ausgabe: : k Cluster $C_1, \ldots, C_k$

1 Berechne die normalisierte Laplace-Matrix $\mathcal{L}_{rw}$ des Graphen G mit Adj^G
2 Bestimme Eigenvektoren $u_1, \ldots, u_k$ zu den k kleinsten Eigenwerten von $\mathcal{L}_{rw}$
3 Sei $U \in \mathbb{R}^{n \times k}$ die Matrix mit $u_1, \ldots u_k$ als Spaltenvektoren
4 Für $i \leftarrow \{1, \ldots, n\}$ sei $y_i \in \mathbb{R}^k$ der Vektor der i-ten Reihe von U
5 Clustere die Punkte $(y_i)_{i \in \{1,\ldots,n\}}$ mit einem k-means-Algorithmus in $C_1, \ldots, C_k$

Vorgehen ist jedoch analog zu dem hier vor uns angegebenen Algorithmus.

Algorithmus 6: Normalisiertes spektrales Clustering [NJW01]

Eingabe: : Gewichtete Adjazenzmatrix $Adj^G \in \mathbb{R}^{n \times n}, k \in \mathbb{N}^+$
Ausgabe: : k Cluster $C_1, \ldots, C_k$

1 Berechne die normalisierte Laplace-Matrix $\mathcal{L}_{sym}$ des Graphen G mit Adj^G
2 Bestimme Eigenvektoren $u_1, \ldots, u_k$ zu den k kleinsten Eigenwerten von $\mathcal{L}_{sym}$
3 Sei $U \in \mathbb{R}^{n \times k}$ die Matrix mit $u_1, \ldots u_k$ als Spaltenvektoren
4 Sei $T \in \mathbb{R}^{n \times k}$ die Matrix mit den Einträgen $T_{i,j} = \frac{U_{i,j}}{\left(\sum_k U_{i,k}^2\right)^{\frac{1}{2}}}$
5 Für $i \leftarrow \{1, \ldots, n\}$ sei $y_i \in \mathbb{R}^k$ der Vektor der i-ten Reihe von T
6 Clustere die Punkte $(y_i)_{i \in \{1,\ldots,n\}}$ mit einem k-means-Algorithmus in $C_1, \ldots, C_k$

Der Algorithmus aus [NJW01] ist dem von [SM00] sehr ähnlich, verwendet jedoch die normalisierte Laplace-Matrix $\mathcal{L}_{sym}$. Außerdem ist für diesen Algorithmus eine zusätzliche Normalisierung nötig, deren Details in [Lux07] erläutert werden.

Das Berechnen aller Eigenvektoren benötigt asymptotischen Rechenaufwand von $\mathcal{O}(n^3)$, was für viele Anwendungen deutlich zu viel Rechenzeit ist. Beschränkt man sich auf die Berechnung von wenigen oder sogar nur einem einzelnen Eigenvektor, lässt sich die Komplexität signifikant verringern, beispielsweise unter Einsatz von Lanczos Algorithmus [Lan50]. Da für viele spektrale Clusteringalgorithmen die Berechnung der Eigenvektoren jedoch nach wie vor ein großer Flaschenhals ist, betrachten wir im nächsten Kapitel einen Ansatz, mit dem die Berechnung von Eigenvektoren für Graphclusterings prinzipiell vermieden werden kann.

3 Kernel-k-means Methoden für spektrales Graphclustering

In diesem Kapitel wollen wir Verfahren zur spektralen Clusteranalyse von Graphen vorstellen und verbessern. Als Grundlage wollen wir den Kernel-k-means Algorithmus verwenden. Lloyds Algorithmus ist nach wie vor eine Heuristik, die in der Praxis mit geringen Ausführungszeiten Clusterings von akzeptabler Qualität berechnet. Unter Einsatz der Kernel Methode können wir die Qualität der berechneten Clusterings noch einmal steigern, ohne dabei signifikante Performanzeinbußen in Kauf nehmen zu müssen. Der Algorithmus ist zudem leicht zu implementieren. Gerade im Bereich der Clusteranalyse sollte dies nicht unterschätzt werden, da eine praktische Evaluation der Algorithmen in diesem anwendungsreichen Gebiet unerlässlich ist.

Das Kapitel ist folgendermaßen gegliedert: In Abschnitt 3.1 diskutieren wir, wie wir mit dem Kernel-k-means Algorithmus Graphclusteringprobleme (und umgekehrt) lösen können. In Abschnitt 3.2 zeigen wir, wie sich die Kernel Methode auch auf den Algorithmus k-means++ anwenden lässt und wie wir damit den Kernel-k-means Algorithmus für Graphclustering verbessern können.

3.1 Graphschnitte und Graphclustering mit gewichtetem Kernel-k-means

Um die Graphclusteringprobleme aus Definition 2.2.3 mit dem Kernel-k-means Algorithmus zu lösen, müssen wir uns zunächst überlegen, wie wir diese in eine Eingabe für den Algorithmus transformieren können. Dass eine solche Transformation (bidirektional) möglich ist, haben bereits Dhillon, Guan und Kulis gezeigt [DGK04, DGK07]. Wir fassen daher zunächst den für uns relevanten Teil ihrer Ergebnisse zusammen.

Die grundlegende Idee besteht darin, zu zeigen, dass sowohl die Zielfunktion für Kernel-k-means als auch die (diversen) Zielfunktionen für Graphpartitionierungsprobleme als *Spurmaximierungsprobleme* formuliert werden können. Die *Spur* (engl. *trace*) einer quadratischen $n \times n$-Matrix M ist dabei die Summe der Elemente auf der Hauptdiagonalen von M:

$$\operatorname{trace}(M) = \sum_{i=1}^{n} M_{i,i}$$

Weiterhin gilt für zwei quadratische Matrizen M, M':

1. $\text{trace}(MM') = \text{trace}(M'M)$
2. $\text{trace}(M^T M) = \|M\|_F^2$
3. $\text{trace}(M + M') = \text{trace}(M) + \text{trace}(M')$

Wir formen zunächst die Kernel-k-means Zielfunktion um und widmen uns dann den Graphpartitionierungsproblemen.

Gewichtetes Kernel-k-means. Für die Umformung des k-means-Problems legen wir das *gewichtete* Kernel-k-means Problem (engl. *weighted kernel k-means*, kurz WKKM) zu Grunde. Dieses lautet analog zu Definition 2.3.2 wie folgt. Unsere Eingabe besteht aus der gewichteten Punktmenge als $n \times d$-Matrix A, einer Gewichtsfunktion w, die einen Punkt-Zeilenvektor $\mathbf{p}$ aus A auf ein reelles Gewicht abbildet, und der Clusteranzahl k. Die Zielfunktion besteht wie gewohnt darin, die quadrierten euklidischen Distanzen der Punkte eines Clusters zu ihrem Clusterzentrum zu minimieren:

$$\min D(\{S_i\}_{i=1}^k) = \sum_{i=1}^{k} \sum_{\mathbf{p} \in S_i} w(\mathbf{p}) \|\phi(\mathbf{p}) - \mathbf{c}_i\|^2 \tag{3.1}$$

Dabei sind die $S_1, \ldots, S_k$ die Cluster, wobei ein Cluster S_i das Zentrum

$$\mathbf{c}_i = \frac{\sum_{\mathbf{p} \in S_i} w(\mathbf{p}) \phi(\mathbf{p})}{\sum_{\mathbf{p} \in S_i} w(\mathbf{p})}$$

hat. Wir setzen auch hier wieder den Kernel-Trick für die Distanzberechnungen ein. Dazu berechnen wir initial n^2 viele Skalarprodukte $\langle \phi(\mathbf{p}_i), \phi(\mathbf{p}_j) \rangle$ und speichern diese in einer $n \times n$-Kernelmatrix K mit $K_{i,j} = \kappa(\mathbf{p}_i, \mathbf{p}_j) = \langle \phi(\mathbf{p}_i), \phi(\mathbf{p}_j) \rangle$. Die Distanzberechnung von einem Punkt zu seinem Clusterzentrum können wir ebenfalls analog zu 2.2 mit Hilfe der Kernelmatrix umformulieren. Sie unterscheidet sich von dieser nur darin, dass die Punkte gewichtet sind. Mit $|S_i| = \sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j)$ gilt:

$$\|\phi(\mathbf{p}_j) - \mathbf{c}_i\|^2 = K_{i,i} - \frac{2 \sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j) K_{i,j}}{\sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j)} + \frac{\sum_{\mathbf{p}_j, \mathbf{p}_l \in S_i} w(\mathbf{p}_j) w(\mathbf{p}_l) K_{j,l}}{\left(\sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j) \right)^2} \tag{3.2}$$

Die Erweiterung von Algorithmus 3 zu Algorithmus 7 ergibt sich ebenfalls unmittelbar.

Algorithmus 7: Weighted Kernel-k-means

Eingabe: : Kernel-Matrix K, $k \in \mathbb{N}^+$, Punktgewichte w
Ausgabe: : k-means-Clustering der Eingabepunktmenge

1 Wähle k initiale Cluster $S_1^{(0)}, \ldots, S_k^{(0)}$; $t \leftarrow 0$
2 **foreach p do**
3 Bestimme $c^*(\mathbf{p}) \leftarrow \arg\min_{i=1}^{k} \|\phi(\mathbf{p}) - \mathbf{c_i}\|^2$ mit 3.2
4 **for** $i \leftarrow 1$ **to** k **do**
5 $S_i^{(t+1)} \leftarrow \{\mathbf{p} : c^*(\mathbf{p}) = i\}$
6 $t \leftarrow t + 1$
7 **if** $t < t_{max}$ ***and*** *nicht konvergiert* **then**
8 **go to** 2
9 **else**
10 **return** $S_1^{(t)}, \ldots, S_k^{(t)}$

Gewichtetes Kernel-k-means als Spurmaximierung. Wir formen als nächstes die Zielfunktion des gewichteten Kernel-k-means Problems in ein Spurmaximierungsproblem um. Dazu gehen wir zunächst davon aus, dass die Punktgewichte in einer $n \times n$-Diagonalmatrix W gespeichert sind, sodass $W_{j,j} = w(\mathbf{p}_j)$ für alle Punkte $\mathbf{p}_j$ und die Punktgewichte des Clusters S_i in der $|S_i| \times |S_i|$-Diagonalmatrix W^i mit $W^j_{m,m} = w(\mathbf{p}_m)$ für alle Punkte $\mathbf{p}_m \in S_i$ gespeichert sind.

Sei nun s_i die Summe der Gewichte der Punkte in Cluster i, also

$$s_i = \sum_{\mathbf{p}_j \in S_i} w(\mathbf{p}_j) = \sum_{\mathbf{p}_j \in S_i} W^i_{j,j}$$

Wir betrachten die $n \times k$-Matrix Z mit den Einträgen

$$Z_{j,i} = \begin{cases} \frac{1}{\sqrt{s_i}}, & \text{falls } \mathbf{p}_j \in S_i, \\ 0 & \text{sonst} \end{cases}$$

sowie die (nicht explizit berechnete) Matrix $\Phi_i = [\phi(\mathbf{p}_1), \ldots, \phi(\mathbf{p}_m)]$ von abgebildeten Punkten des Clusters $S_i = \{\mathbf{p}_1, \ldots, \mathbf{p}_m\}$. Mit diesen können wir die Zielfunktion des gewichteten Kernel-k-means Problems (3.1) umformen. Wir erinnern uns, dass diese die Minimierung der Summe der Intraclusterabstände ist. Wir schreiben für den Intraclusterabstand eines Clusters S_i: $d(S_i) = \sum_{\mathbf{p} \in S_i} w(\mathbf{p}) \|\phi(\mathbf{p}) - \mathbf{c}_i\|^2$. Damit können wir den Zielfunktionswert $D(\{S_i\}_{i=1}^k)$ folgendermaßen vereinfachen:

$$D(\{S_i\}_{i=1}^k) = \sum_{i=1}^{k} d(S_i)$$

Der Zentroidvektor c_i des Clusters S_i berechnet sich in unserer ursprünglichen Problemformulierung durch

$$\mathbf{c}_i = \frac{\sum_{\mathbf{p}\in S_i} w(\mathbf{p})\phi(\mathbf{p})}{\sum_{\mathbf{p}\in S_i} w(\mathbf{p})}$$

Diese Berechnungsvorschrift ist wegen der Definition von Φ_i äquivalent zu

$$\mathbf{c}_i = \Phi_i \frac{W^j \mathbf{e}}{s_j},$$

wobei $\mathbf{e}$ der Einsvektor der passenden Dimension ist. Wir können alternativ unter Einsatz der Matrix Z auch direkt eine Matrix C aller Clusterzentren bestimmen, wenn wir für die gesamte Eingabepunktmenge die (ebenfalls nicht explizit berechnete) Abbildungsmatrix $\Phi = [\Phi_1, \ldots, \Phi_k] = [\phi(\mathbf{p}_1), \ldots, \phi(\mathbf{p}_n)]$ verwenden:

$$C = \Phi W Z Z^T$$

In diesem Fall entspricht das Zentrum eines einzelnen Clusters S_i

$$\mathbf{c}_i = \left(\Phi W Z Z^T\right)_{\cdot i}$$

wobei wir für eine Matrix M mit $M_{\cdot i}$ die i-te Spalte von M meinen. Damit ist es uns möglich, den Zielfunktionswert so zu formulieren, dass wir später eine Äquivalenz zur Graphpartitionierung sehen werden. Mit $Y = W^{\frac{1}{2}} Z = U D^{\frac{1}{2}} U^{-1} Z$ gilt:

$$\begin{aligned}
D(\{S_i\}_{i=1}^k) &= \sum_{i=1}^{k} \sum_{\mathbf{p}\in S_i} w(\mathbf{p}) \|\phi(\mathbf{p}) - \mathbf{c}_i\|^2 \\
&= \sum_{j=1}^{n} W_{j,j} \left\| \Phi_{\cdot j} - \left(\Phi W Z Z^T\right)_{\cdot j} \right\|^2 \\
&= \sum_{j=1}^{n} W_{j,j} \left\| \Phi_{\cdot j} - \left(\Phi W^{\frac{1}{2}} Y Y^T W^{-\frac{1}{2}}\right)_{\cdot j} \right\|^2 \\
&= \sum_{j=1}^{n} \left\| \Phi_{\cdot j} \left(W_{j,j}\right)^{\frac{1}{2}} - \left(\Phi W^{\frac{1}{2}} Y Y^T\right)_{\cdot j} \right\|^2 \\
&= \left\| \Phi W^{\frac{1}{2}} - \Phi W^{\frac{1}{2}} Y Y^T \right\|_F^2
\end{aligned}$$

Der ursprüngliche Zielfunktionswert ist damit auf die (quadrierte) Frobenius-Norm von $\Phi W^{\frac{1}{2}} - \Phi W^{\frac{1}{2}} Y Y^T$ transformiert. Wir können nun die oben genannte Verbindung zwischen der Frobenius-Norm und der Matrix-Spur ausnutzen, um zur

beabsichtigten Spur-Maximierung zu gelangen. Wir erinnern uns, dass für eine Matrix M gilt, dass $\text{trace}(M^T M) = \|M\|_F^2$. Wir können somit weiter umformen:

$$\begin{aligned} D(\{S_i\}_{i=1}^k) &= \text{trace}(W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}} - W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}YY^T \\ &\quad - YY^T W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}} + YY^T W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}YY^T) \\ &= \text{trace}(W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}) - \text{trace}(Y^T W^{\frac{1}{2}}\Phi^T\Phi W^{\frac{1}{2}}Y) \end{aligned}$$

In der Kernel-Matrix K speichern wir sämtliche Skalarprodukte der projizierten Punkte, demnach ist per Definition $K = \Phi^T\Phi$. Die Kernel-Matrix ist genau wie die Gewichts-Matrix W unabhängig von der Lösung. Dementsprechend ist auch $\text{trace}(W^{\frac{1}{2}}KW^{\frac{1}{2}})$ unabhängig von der Lösung. Die Minimierungszielfunktion 3.1 ist daher gleichzusetzen mit der Maximierungszielfunktion des folgenden Spurmaximierungsproblems:

$$\max_Y \text{trace}(Y^T W^{\frac{1}{2}}KW^{\frac{1}{2}}Y) \tag{3.3}$$

Die Spur der Matrix Y hat offensichtlich hat offensichtlich einen wichtigen Anteil an der Qualität der Lösung. Wir wollen daher abschließend noch kurz einen genaueren Blick auf Y werfen. Die Matrix $Y = W^{\frac{1}{2}}Z$ ist eine orthonormale Diagonalmatrix, das heißt $Y^TY = I$. Ihre Einträge sehen wie folgt aus:

$$Y = \begin{pmatrix} \frac{W_1^{\frac{1}{2}}\mathbf{e}}{\sqrt{s_1}} & & & \\ & \frac{W_2^{\frac{1}{2}}\mathbf{e}}{\sqrt{s_2}} & & \\ & & \cdots & \\ & & & \frac{W_k^{\frac{1}{2}}\mathbf{e}}{\sqrt{s_k}} \end{pmatrix}$$

Graphpartitionierung als Spurmaximierung. Es ist möglich, für alle in Definition 2.2.3 vorgestellten Problemstellungen ein äquivalentes Spurmaximierungsproblem anzugeben. Wir beschränken uns hier zunächst auf die beiden Zielfunktionen mit praktischer Relevanz für das Evaluations-Kapitel dieser Arbeit, nämlich die Ratio Association und den Normalized Cut. Wir beginnen mit der Ratio Association. Das hier verwendete w entspricht dem aus Definition 2.2.3. Wir führen zunächst einen Indikator-Vektor $\mathbf{x}^c$ der Dimension n für jede Partition beziehungsweise jedes Cluster c ein, sodass $\mathbf{x}^c(i) = 1$ genau dann, wenn das Cluster c den Knoten i enthält. Die Zielfunktion bei der Ratio Association lautet

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|}$$

Wir nehmen an, dass A die (gewichtete) Adjazenzmatrix des Eingabegraphen ist und definieren zusätzlich

$$\tilde{\mathbf{x}}^c = \frac{\mathbf{x}^c}{((\mathbf{x}^c)^T\mathbf{x}^c)^{\frac{1}{2}}}$$

Das Produkt $(\mathbf{x}^c)^T\mathbf{x}^c$ entspricht der Größe der Partition c:

$$|V_c| = (\mathbf{x}^c)^T\mathbf{x}^c$$

Die Summe der Kantengewichte der Partition $w(V_c, V_c)$ können wir über folgendes Produkt abbilden:

$$w(V_c, V_c) = (\mathbf{x}^c)^T A\mathbf{x}^c$$

Damit können wir eine erste Transformation der Zielfunktion vornehmen:

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{|V_c|} \Leftrightarrow \max \sum_{c=1}^{k} \frac{(\mathbf{x}^c)^T A\mathbf{x}^c}{(\mathbf{x}^c)^T\mathbf{x}^c} \Leftrightarrow \max \sum_{c=1}^{k} (\tilde{\mathbf{x}}^c)^T A\tilde{\mathbf{x}}^c$$

Betrachten wir $\tilde{X}$ als die Matrix, deren c-te Spalte der Vektor $\tilde{\mathbf{x}}^c$ ist, können wir die Zielfunktion umformen zu

$$\max_{\tilde{X}} \operatorname{trace}(\tilde{X}^T A\tilde{X}) \tag{3.4}$$

Wenn wir in 3.3 für die Gewichtsmatrix W die Einheitsmatrix wählen, also $W = \mathbb{1}_n$ und die Adjazenzmatrix als Kernel-Matrix setzen, sodass $K = A$, dann sind 3.4 und 3.3 äquivalent. Auf diese Weise können wir das Ratio Association Problem mit unserem Algorithmus für gewichtetest Kernel-k-means lösen.

Für den Normalized Cut gehen wir ähnlich vor. Unsere ursprüngliche Zielfunktion lautet

$$\min \sum_{c=1}^{k} \frac{w(V_c, V \setminus V_c)}{w(V_c, V)}$$

Zunächst wollen wir die Funktion in ein Maximierungsproblem umwandeln. Wir beobachten, dass die Zielfunktion äquivalent ist zu

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{w(V_c, V)}$$

Wir benötigen zur Umformung des Normalized Cut die diagonale Gradmatrix D des Graphen, welcher der Adjazenzmatrix A zu Grunde liegt. Wir verwenden wieder

die Indikator-Vektoren und definieren zusätzlich

$$\hat{\mathbf{x}}^c = \frac{\mathbf{x}^c}{((\mathbf{x}^c)^T D \mathbf{x}^c)^{\frac{1}{2}}}$$

Unter Verwendung der Gradmatrix können wir die Variante des Normalized Cut als Maximierungsproblem wie folgt umformen:

$$\max \sum_{c=1}^{k} \frac{w(V_c, V_c)}{w(V_c, V)} \Leftrightarrow \max \sum_{c=1}^{k} \frac{(\mathbf{x}^c)^T A \mathbf{x}^c}{(\mathbf{x}^c)^T D \mathbf{x}^c} \Leftrightarrow \max \sum_{c=1}^{k} (\hat{\mathbf{x}}^c)^T A \hat{\mathbf{x}}^c$$

Im mittleren Term zählen wir die Kanten im Zähler und Nenner jeweils doppelt. Dies gleich sich dann bei der Division aus. Wir nehmen analog zur Ratio Association an, dass die Matrix $\hat{X}$ die Spaltenmatrix der Vektoren $\hat{\mathbf{x}}^c$ ist. Mit $\tilde{Y} = D^{\frac{1}{2}} \hat{X}$ lautet das Spurmaximierungsproblem für den Normalized Cut folgendermaßen:

$$\max_{\tilde{Y}} \operatorname{trace}(\tilde{Y}^T D^{-\frac{1}{2}} A D^{-\frac{1}{2}} \tilde{Y}) \tag{3.5}$$

Das Problem in 3.5 ist äquivalent zu 3.3, wenn wir als Gewichtsmatrix $W = D$ und als Kernel-Matrix $K = D^{-1} A D^{-1}$ setzen.

Wir wollen abschließend noch anmerken, dass die Adjazenzmatrix A positiv definit sein muss, damit sie als Kernel-Matrix in Algorithmus 7 garantiert zu einer iterativen Verringerung des gewichteten Kernel-k-means Zielfunktionswertes führt. Kernel-Matrizen müssen zudem generell positiv definit sein, damit sie Skalarprodukte in einem anderen Vektorraum berechnen [SC04]. Um dies in der Praxis sicherzustellen, ist bei der Wahl der Kernel-Matrix gegebenenfalls das Aufaddieren einer „Verschiebung“ auf die ursprüngliche Kernel-Matrix nötig. Wir gehen auf dieses Detail im Experimente-Kapitel noch genauer ein.

3.2 Die Kernel Methode für k-means++

Ein wichtiges Merkmal von Algorithmus 7 ist, dass er grundsätzlich ohne Methoden der spektralen Clusteranalyse auskommt. Wir müssen uns jedoch wie auch bei Lloyds Algorithmus ohne Kernel Methode mit der geeigneten initialen Wahl der Clusterzentren beschäftigen. Zunächst einmal wählen wir diese wie schon zuvor zufällig aus der Eingabepunktmenge. Diese Vorgehensweise ist durchaus praktikabel, die experimentelle Evaluation aus [DGK04, DGK07] zeigt jedoch, dass sich mit einer geschickteren Wahl die Qualität des letztlich berechneten Clusterings noch signifikant steigern lässt. Die Autoren verwenden für die Initialisierung die spektrale

Clusteranalyse, was wiederum die Laufzeit der Eigenvektor-Berechnung mit sich bringt.

Konkret wird mit einem Verfahren aus [GL96] eine Problemrelaxion von 3.3/ 3.4 beziehungsweise von 3.3/ 3.5 gelöst. Relaxieren wir die Wahl von Y beziehungsweise $\tilde{Y}$ auf eine beliebige orthonormale Matrix, dann sind diese von der Form $Y = V_k Q$ beziehungsweise $\tilde{Y} = \tilde{V}_k Q$. Dabei ist Q eine beliebige orthogonale $k \times k$-Matrix, V_k die Matrix der k größten Eigenvektoren von $W^{\frac{1}{2}} K W^{\frac{1}{2}}$ und $\tilde{V}_k$ die Matrix der k größten Eigenvektoren von $D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$.

Die Berechnung der jeweiligen k größten Eigenvektoren wird mit wachsender Clusteranzahl k aufwändig, wie wir schon in Abschnitt 2.3 beobachtet haben. Intuitiv ist daher die Fragestellung, ob es möglich ist, eine Initialisierung ohne Eigenvektor-Berechnungen vorzunehmen, die dennoch deutlich verbesserte Clusterings mit dem Kernel-k-means-Algorithmus ermöglicht.

In Abschnitt 2.1 haben wir bereits den Algorithmus k-means++ von Arthur und Vassilvitskii [AV07] kennengelernt. Die initiale Wahl der Cluster erfolgt hier gemäß einer Wahrscheinlichkeitsverteilung der Punkte als mögliche Clusterzentren. Diese ist proportional zur quadrierten euklidischen Distanz zum jeweils nächstgelegenen bereits gewählten Zentrum ist. Wir zeigen nun, dass die Kernel Methode auch auf den Algorithmus k-means++ angewendet werden kann. Damit ist es uns möglich, Algorithmus 3 (und damit auch Algorithmus 7) in eine Kernel-basierte Variante von k-means++ umzuwandeln, die wir Kernel-k-means++ nennen.

Wir erinnern uns, dass wir im Zusammenhang mit k-means++ für einen Punkt x aus der Eingabe-Punktmenge P mit $D(x)$ die geringste Distanz von x zum nächstgelegenen bereits gewählten Zentrum bezeichnen. Der Punkt x wird als nächstes Zentrum mit der Wahrscheinlichkeit

$$\frac{D(x)^2}{\sum_{y \in P} D(y)^2}$$

gewählt. Wir können demnach die Kernel Methode auf den Algorithmus k-means++ anwenden, wenn wir für jeden Eingabepunkt x den Wert $D(\phi(x))$ beziehungsweise $D(\phi(x))^2$ berechnen können. Dazu zeigen wir nun, dass wir diese Werte wie schon zuvor beim Kernel-k-means-Algorithmus ausschließlich über die Kernelfunktion κ ohne explizite Kenntnis der abgebildeten Punkte oder der Abbildung ϕ selbst berechnen können.

Wir nehmen zunächst an, dass die Zentren $c_1, \ldots, c_l$ mit $l \in \{1, \ldots, k-1\}$ bereits gewählt worden sind. Dann können wir $D(x)$ formal beschreiben als das Minimum

über alle quadrierten euklidischen Distanzen von x zu jedem der bisher gewählten Zentren c_i:

$$D(x) = \min_{i=1\ldots l} \|x - c_i\|^2$$

Wenden wir die Kernel Methode an, berechnet sich $D(\phi(x))$ dementsprechend folgendermaßen:

$$D(\phi(x)) = \min_{i=1\ldots l} \|\phi(x) - \phi(c_i)\|^2$$

Wir können hier die $\phi(c_i)$ berechnen, da diese im Kernel-k-means++-Algorithmus stets Eingabepunkt sind. Im Allgemeinen ist dies nicht möglich. Es genügt, zu zeigen, dass wir $\|\phi(x) - \phi(c_i)\|^2$ ausschließlich mit der Kernelfunktion berechnen können. Es gilt:

$$\|\phi(x) - \phi(c_i)\|^2 = \kappa(x, x) - 2\kappa(x, c_i) + \kappa(c_i, c_i) \tag{3.6}$$

Mit denselben Überlegungen können wir auch zeigen, dass sich der Wert von $D(\phi(x))^2$ mit der Kernel Methode berechnen lässt. Zunächst lässt sich auch $D(\phi(x))^2$ als Minimum formulieren:

$$D(\phi(x))^2 = \left(\min_{i=1\ldots l} \|\phi(x) - \phi(c_i)\|^2\right)^2$$

Dazu müssen wir effektiv $\left(\|\phi(x) - \phi(c_i)\|^2\right)^2$ berechnen. Auch dies lässt sich mit der Kernelfunktion umsetzen:

$$\begin{aligned}\left(\|\phi(x) - \phi(c_i)\|^2\right)^2 &= \left(\kappa(x, x) - 2\kappa(x, c_i) + \kappa(c_i, c_i)\right)^2 \\ &= \left(\kappa(x, x)\right)^2 - 4\kappa(x, x)\kappa(x, c_i) + 2\kappa(x, x)\kappa(c_i, c_i) \\ &\quad + 4\kappa(x, c_i)\kappa(x, c_i) - 4\kappa(x, c_i)\kappa(c_i, c_i) + \left(\kappa(c_i, c_i)\right)^2 \end{aligned} \tag{3.7}$$

Der Algorithmus Kernel-k-means++ wird damit die logische Weiterentwicklung von k-means++ auf Basis der Kernel Methode.

Wir zeigen nun, dass auch Kernel-k-means++ eine $\mathcal{O}(\log k)$-Approximation für das k-means-Problem ist. Strukturell gehen wir dabei analog zu dem Papier von Arthur und Vassilvitskii [AV07] vor, da sich durch die Kernel Methode keine wesentlichen Änderungen ergeben. Für eine Zentrenmenge $C = \{c_1, \ldots, c_k\}$ bezeichnen wir mit dem *Potenzial* θ den Zielfunktionswert des Clusterings bezüglich der Kernel-k-means-Zielfunktion, das heißt

$$\theta(C) = \sum_{p \in P} \min_{c \in C} \|\phi(p) - c\|^2$$

Algorithmus 8: Kernel-k-means++

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+$, Kernelfunktion κ
Ausgabe: : k initiale Clusterzentren für P

1 Wähle c_1 zufällig gleichverteilt aus P
2 **for** $i \leftarrow 2$ **to** k **do**
3 Berechne $D(\phi(x'))^2$ mit 3.7 für alle $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$
4 Wähle den Punkt $x' \in P \setminus \{c_1, \ldots, c_{i-1}\}$ als Zentrum c_i mit Wahrscheinlichkeit $\frac{D(\phi(x'))^2}{\sum_{x \in P} D(\phi(x))^2}$
5 Führe den Kernel-k-means-Algorithmus mit den initialen Clusterzentren $c_1, \ldots, c_k$ aus.

Das Potenzial des optimalen Clusterings C_{OPT} notieren wir mit θ_{OPT}. Allgemeiner bezeichnen wir mit dem Potenzial für eine Punktmenge $A \subset P$: $\theta(A) = \sum_{p \in A} \min_{c \in C} \|\phi(p) - c\|^2$. Bevor wir mit der Analyse beginnen, benötigen wir folgendes Lemma, das einen beliebigen Punkt mit einer Punktmenge und ihrem Zentroiden in Relation setzt:

Lemma 3.2.1. Sei V ein euklidischer Vektorraum und $S \subset V$ eine Vektormenge mit dem geometrischen Zentroiden $z(S) = \frac{1}{|S|} \sum_{\mathbf{x} \in S} \mathbf{x}$. Weiterhin sei $\mathbf{q} \in V$ ein beliebiger Vektor aus V. Es gilt:

$$\sum_{\mathbf{x} \in S} \|\mathbf{x} - \mathbf{q}\|^2 - \sum_{\mathbf{x} \in S} \|\mathbf{x} - z(S)\|^2 = |S| \cdot \|z(S) - \mathbf{q}\|^2$$

Ein Beweis von Lemma 3.2.1 findet sich beispielsweise in [Sch14, Lemma 2.4.1, S. 25]. Wir zeigen nun das eigentliche Ergebnis.

Satz 3.2.1 (Kernel-k-means++ ist $\mathcal{O}(\log k)$-kompetitiv). Wenn das Clustering C mit Kernel-k-means++ berechnet wird, dann gilt für das Potenzial θ von C:

$$E[\theta(C)] \leq 8(\ln k + 2)\theta_{\text{OPT}}$$

Der Beweis erfolgt in drei Schritten. Wir zeigen zunächst, dass das erste zufällig gleichverteilt gewählte Clusterzentrum die Approximationseigenschaft erfüllt. Dieselbe Eigenschaft zeigen wir im zweiten Schritt für die übrigen Zentren, die gemäß der D^2-Gewichtung gewählt werden. Im dritten und letzten Schritt beweisen wir mittels Induktion das Gesamtergebnis.

Lemma 3.2.2 (Das erste Zentrum ist kompetitiv). Sei A ein beliebiges Cluster von C_{OPT} und sei C ein Clustering mit genau einem Zentrum, das zufällig

gleichverteilt aus A gewählt wird. Dann gilt:

$$E[\theta(A)] = 2\theta_{\mathrm{OPT}}(A)$$

Beweis. Sei $z(A)$ der geometrische Zentroid der Punkte in A, das heißt

$$z(A) = \frac{1}{|A|} \sum_{\mathbf{p} \in A} \phi(\mathbf{p})$$

Mit Lemma 3.2.1 gilt:

$$\begin{aligned}
&\sum_{\mathbf{p} \in A} \frac{1}{|A|} \cdot \left(\sum_{\mathbf{p}' \in A} \|\phi(\mathbf{p}) - \phi(\mathbf{p}')\|^2 \right) \\
&= \frac{1}{|A|} \sum_{\mathbf{p} \in A} \left(\sum_{\mathbf{p}' \in A} \|\phi(\mathbf{p}') - z(A)\|^2 + |A| \cdot \|\phi(\mathbf{p}) - z(A)\|^2 \right) \\
&= 2 \sum_{\mathbf{p} \in A} \|\phi(\mathbf{p}) - z(A)\|^2 \\
&= 2E[\theta(A)]
\end{aligned}$$

□

Wir zeigen nun, dass die Approximationseigenschaft auch für die übrigen Zentren, die gemäß der D^2-Gewichtung gewählt werden, gilt.

Lemma 3.2.3 (Die übrigen Zentren sind kompetitiv). Sei A ein beliebiges Cluster von C_{OPT} und sei C ein beliebiges Clustering. Wenn ein Zentrum aus A zufällig nach der D^2-Gewichtung zu C hinzugefügt wird, dann gilt:

$$E[\theta(A)] \leq 8\theta_{\mathrm{OPT}}(A)$$

Beweis. Ein Punkt $\mathbf{p} \in A$ wird mit der Wahrscheinlichkeit

$$\frac{D(\phi(\mathbf{p}))^2}{\sum_{\mathbf{p}' \in A} D(\phi(\mathbf{p}'))^2}$$

als Zentrum gewählt. Wenn der Punkt $\mathbf{p}$ als Zentrum gewählt wird, beträgt der Potenzial-Wert eines Punktes $\mathbf{p}'$ der dem Zentrum $\mathbf{p}$ zugeordnet ist, vor dem Hinzufügen von p genau

$$\min\{D(\phi(\mathbf{p}')), \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2\}$$

Der Erwartungswert für das Potenzial des gesamten Clusters A ist dementsprechend:

$$E[\theta(A)] = \sum_{\mathbf{p}\in A} \frac{D(\phi(\mathbf{p}))^2}{\sum_{\mathbf{p}'\in A} D(\phi(\mathbf{p}'))^2} \sum_{\mathbf{p}'\in A} \min\{D(\phi(\mathbf{p}')), \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2\}$$

Für die Berechnung von $D(\phi(\mathbf{p}'))$ und der Distanzen $\|\phi(\mathbf{p}') - \phi(\mathbf{p})\|$ verwenden wir im Kernel-k-means++-Algorithmus die Kernelfunktion κ. Die Kernelfunktion beziehungsweise die zugehörige Kernelmatrix ist *positiv definit* (für Details siehe beispielsweise [SC04]), daher gilt für diese die Cauchy-Schwarz-Ungleichung, also $|\kappa(x,y)|^2 \leq \kappa(x,x) \cdot \kappa(y,y)$. Aus der Cauchy-Schwarz-Ungleichung folgen die Dreiecks-Ungleichung, sowie die Power-Mean-Ungleichung bezüglich der Kernelfunktion. Letztere besagt für reelle Zahlen $a_1, \ldots, a_m$: $\sum a_i^2 \geq \frac{1}{m}\left(\sum_{a_i}\right)^2$. Wir können daher an dieser Stelle die Dreiecks-Ungleichung anwenden und erhalten für alle $\mathbf{p}, \mathbf{p}'$:

$$D(\phi(\mathbf{p})) \leq D(\phi(\mathbf{p}')) + \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|$$

Für die D^2-Gewichtung gilt zudem mit der Power-Mean-Ungleichung:

$$D(\phi(\mathbf{p}))^2 \leq 2D(\phi(\mathbf{p}'))^2 + 2\|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2$$

Summiert über alle Punkte $\mathbf{p} \in A$ erhalten wir somit:

$$D(\phi(\mathbf{p}))^2 \leq \frac{2}{|A|} \sum_{\mathbf{p}'\in A} D(\phi(\mathbf{p}'))^2 + \frac{2}{|A|} \sum_{\mathbf{p}'\in A} \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2$$

Wir schätzen nun $E[\theta(A)]$ nach oben ab:

$$\begin{aligned} E[\theta(A)] \leq{}& \frac{2}{|A|} \cdot \sum_{\mathbf{p}\in A} \frac{\sum_{\mathbf{p}'\in A} D(\phi(\mathbf{p}'))^2}{\sum_{\mathbf{p}'\in A} D(\phi(\mathbf{p}'))^2} \cdot \sum_{\mathbf{p}'\in A} \min\{D(\phi(\mathbf{p}')), \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2\} \\ &+ \frac{2}{|A|} \cdot \sum_{\mathbf{p}\in A} \frac{\sum_{\mathbf{p}'\in A} \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2}{\sum_{\mathbf{p}'\in A} D(\phi(\mathbf{p}'))^2} \\ &\cdot \sum_{\mathbf{p}'\in A} \min\{D(\phi(\mathbf{p}')), \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2\} \end{aligned}$$

Im ersten Summanden schätzen wir

$$\min\{D(\phi(\mathbf{p}')), \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2\} \leq \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2$$

nach oben ab, und im zweiten Summanden

$$\min\{D(\phi(\mathbf{p}')), \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2\} \leq D(\phi(\mathbf{p}'))$$

Damit erhalten wir vereinfacht:

$$E[\theta(A)] \leq \frac{4}{|A|} \sum_{\mathbf{p} \in A} \sum_{\mathbf{p}' \in A} \|\phi(\mathbf{p}') - \phi(\mathbf{p})\|^2$$

Nun können wir Lemma 3.2.2 anwenden und erhalten:

$$E[\theta(A)] \leq 8\theta_{\mathrm{OPT}}(A)$$ □

Aus den Lemmas 3.2.2 und 3.2.3 folgt, dass die Wahl der Zentren von Kernel-k-means++ der Approximationseigenschaft genügt, solange der Algorithmus Zentren aus jedem Cluster des optimalen Clusterings C_{OPT} auswählt. Im dritten und letzten Schritt unserer Analyse zeigen wir nun, dass der entstehende Fehler im anderen Fall im Erwartungswert nach oben insgesamt durch $\mathcal{O}(\log k)$ beschränkt ist.

Lemma 3.2.4. Sei C ein beliebiges Clustering. Gegeben seien $u > 0$ viele freie Cluster (die übrigen Cluster nennen wir abgedeckt) von C_{OPT}. P_u seien die Punkte in diesen Clustern. Weiterhin sei $P_c = P - P_u$. Wenn $t \leq u$ zufällig gemäß der D^2-Gewichtung gewählte Zentren zu C hinzugefügt werden, dann gilt für das erwartete neue Potenzial $E[\theta'(C)]$ von C:

$$E[\theta'(C)] \leq (\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot (1 + H_t) + \frac{u - t}{u} \cdot \theta(P_u)$$

Dabei ist H_t die t-te harmonische Zahl $H_t = 1 + \frac{1}{2} + \cdots + \frac{1}{t}$.

Beweis. Wir beweisen die Aussage durch vollständige Induktion für $(t-1, u)$ und $(t-1, u-1)$.

Induktionsanfang. Falls $t = 0$ und $u > 0$, gilt:

$$1 + H_t = \frac{u - t}{u} = 1$$

Wir betrachten nun den Fall, dass $t = u = 1$. Wir wählen genau ein neues Zentrum aus der Menge von genau einem freien Cluster mit der Wahrscheinlichkeit $\frac{\theta(P_u)}{\theta(C)}$. In diesem Fall folgt aus Lemma 3.2.3, dass

$$E[\theta'(C)] \leq \theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)$$

Auch wenn wir ein Zentrum aus einem abgedeckten Cluster wählen, gilt $\theta'(C) \leq \theta(C)$. Somit können wir das erwartete neue Potenzial in diesem Fall nach oben

abschätzen:

$$\begin{aligned} E[\theta'(C)] &\leq \frac{\theta(P_u)}{\theta(C)} \cdot (\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) + \frac{\theta(P_u)}{\theta(C)} \cdot \theta(C) \\ &\leq 2\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u) \end{aligned}$$

Bezüglich der Behauptung gilt in diesem Fall, dass $1 + H_t = 1 + 1 = 2$ (und $\frac{u-t}{u} = \frac{1-1}{1} = 0$). Daher gilt die Behauptung auch in diesem Fall.

Induktionsschritt. Wir nehmen eine Fallunterscheidung bezüglich des Clusters vor, aus dem wir unser erstes Zentrum wählen.

(a) Wir nehmen an, das erste gewählte Zentrum stammt aus einem abgedeckten Cluster. Analog zum Induktionsanfang ist die Wahrscheinlichkeit dafür $\frac{\theta(P_c)}{\theta(C)}$. Da das neue Zentrum aus einem abgedeckten Cluster stammt, kann es $\theta(C)$ nicht vergrößern. Wir verwenden die Induktionsannahme mit $t - 1$. Folglich tragen wir erwartet höchstens den folgenden Betrag zu $E[\theta'(C)]$ bei:

$$\frac{\theta(P_c)}{\theta(C)} \cdot \left((\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot (1 + H_{t-1}) + \frac{u - (t-1)}{u} \cdot \theta(P_u) \right)$$

(b) Wir nehmen an, das erste gewählte Zentrum stammt aus einem freien Cluster A. Die Wahrscheinlichkeit für dieses Ereignis ist $\frac{\theta(A)}{\theta(C)}$. Für alle $a \in A$ sei p_a die Wahrscheinlichkeit, dass wir a als Zentrum wählen. Weiterhin sei θ_a das Potenzial $\theta(A)$, nachdem a als Zentrum gewählt wurde. Wir wenden wieder die Induktionsannahme an, setzen jedoch $t = t-1$, $u = u-1$ und fügen A zur Menge der abgedeckten Cluster hinzu. Dann tragen wir erwartet höchstens den folgenden Betrag zu $E[\theta'(C)]$ bei:

$$\begin{aligned} &\frac{\theta(A)}{\theta(C)} \cdot \sum_{a \in A} p_a \left((\theta(P_c) + \theta_a + 8\theta_{\mathrm{OPT}}(P_u) - 8\theta_{\mathrm{OPT}}(A)) \right) \\ &\quad \cdot (1 + H_{t-1}) + \frac{u - (t-1)}{u - 1} \cdot (\theta(P_u) - \theta(A)) \\ &\leq \frac{\theta(A)}{\theta(C)} \cdot \Bigg((\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot (1 + H_{t-1}) \\ &\quad + \frac{u - t}{u - 1} \cdot (\theta(P_u) - \theta(A)) \Bigg) \end{aligned}$$

Die zweite Abschätzung gilt, da $\sum_{a \in A} p_a \leq 8\theta_{\mathrm{OPT}}(A)$ direkt aus Lemma 3.2.3 folgt. An dieser Stelle müssten wir über alle freien Cluster summieren, um den gesamten Potenzial-Beitrag nach oben abzuschätzen. Dazu setzen wir

die Power-Mean-Ungleichung ein. Mit dieser gilt: $\sum_{A \subset P_u} \theta(A^2) \geq \frac{1}{u} \cdot \theta(P_u)^2$. Damit ist es uns nun möglich, den über alle freien Cluster aufsummierten Beitrag zum Potenzial nach oben zu beschränken:

$$\begin{aligned} &\frac{\theta(P_u)}{\theta(C)} \cdot (\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot (1 + H_{t-1}) \\ &\quad + \frac{1}{\theta(C)} \cdot \frac{u-t}{u-1} \cdot \left(\theta(P_u)^2 - \frac{1}{u} \cdot \theta(P_u)^2 \right) \end{aligned}$$

Dieser Term lässt sich etwas vereinfachen zu:

$$\frac{\theta(P_u)}{\theta(C)} \cdot (\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot (1 + H_{t-1}) + \frac{u-t}{u} \cdot \theta(P_u)$$

Wir kombinieren nun den erwarteten Potenzial-Beitrag zu $\theta'(C)$ aus den Fällen (a) und (b) und erhalten die folgende obere Schranke:

$$\begin{aligned} E[\theta'(C)] &\leq (\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot (1 + H_{t-1}) + \frac{u-t}{u} \cdot \theta(P_u) + \frac{\theta(P_c)}{\theta(C)} \cdot \frac{\theta(P_u)}{u} \\ &\leq (\theta(P_c) + 8\theta_{\mathrm{OPT}}(P_u)) \cdot \left(1 + H_{t-1} + \frac{1}{u} \right) + \frac{u-t}{u} \cdot \theta(P_u) \end{aligned}$$

In der Aussage von Lemma 3.2.4 ist $t \leq u$ vorausgesetzt. Daher gilt $\frac{1}{u} \leq \frac{1}{t}$. Damit folgt der Induktionsschluss. □

Wir können Satz 3.2.1 als Spezialfall von Lemma 3.2.4 beweisen. Dazu betrachten wir das Clustering C, welches wir erhalten, nachdem die Zentren-Wahl von Kernel-k-means++ abgeschlossen ist und bevor der Kernel-k-means Algorithmus ausgeführt wird. Sei A das Cluster aus C_{OPT}, aus dem wir das erste Zentrum ausgewählt haben. Wir wenden Lemma 3.2.4 mit $t = u = k - 1$ an und legen dazu A als einziges abgedecktes Cluster fest. Dann gilt:

$$E[\theta_{\mathrm{OPT}}] \leq (\theta(A) + 8\theta_{\mathrm{OPT}} - 8\theta_{\mathrm{OPT}}(A)) \cdot (1 + H_{k-1})$$

Es ist allgemein bekannt, dass für $t \geq 2$ die t-te harmonische Zahl nach oben beschränkt ist durch $H_t < \ln t + 1$. Damit gilt insbesondere $H_{k-1} \leq \ln k + 1$. Somit gilt:

$$E[\theta_{\mathrm{OPT}}] \leq (\theta(A) + 8\theta_{\mathrm{OPT}} - 8\theta_{\mathrm{OPT}}(A)) \cdot (\ln k + 2)$$

Satz 3.2.1 folgt mit Lemma 3.2.2.

3.3 Eine praktische Kernmengenkonstruktion

In diesem Abschnitt wollen wir die Kernmengenkonstruktion von Feldman, Schmidt und Sohler [FSS13, Sch14] auf ihre Implementierbarkeit untersuchen und gegebenenfalls für eine praktische Umsetzung modifizieren.

Bevor wir die Konstruktion erläutern, benötigen wir zwei Konzepte, die zum Verständnis nötig sind. Zunächst wird mit der Konstruktion streng genommen eine Kernmenge *mit Verschiebung* Δ berechnet. Diese unterscheidet sich von einer starken Kernmenge lediglich in der zusätzlichen Addition der Konstante Δ in den Kosten der Kernmenge. Formal ist eine Kernmenge mit Verschiebung folgendermaßen definiert.

Definition 3.3.1 (Kernmenge mit Verschiebung [FSS13]). Sei $P \subset \mathbb{R}^d$ eine endliche Punktmenge und sei $w_1 : P \to \mathbb{R}$ eine Gewichtsfunktion, die jedem Punkt in P ein Gewicht zuordnet. Sei weiterhin $\epsilon \in (0, 1)$ und $\Delta \in \mathbb{R}$. Eine endliche Menge $S \subset \mathbb{R}^d$ und eine Gewichtsfunktion $w_2 : S \to \mathbb{R}$ bilden eine Kernmenge mit Verschiebung Δ für P, wenn für alle Mengen von k Zentren $C \subset P$ gilt:

$$(1-\epsilon)\sum_{x\in P} w_1(x)\min_{c\in C}\|x-c\|^2 \leq \sum_{y\in S} w_2(y)\min_{c\in C}\|y-c\|^2+\Delta \leq (1+\epsilon)\sum_{x\in P} w_1(x)\min_{c\in C}\|x-c\|^2$$

Weiterhin liegt der Kernmengenkonstruktion die bereits erwähnte Erkenntnis zu Grunde, dass die optimale Lösung eines 1-means-Problems für eine Punktmenge P der geometrische Zentroid $z(P)$ ist. Mit dem folgenden Lemma lassen sich die 1-means-Kosten einer Punktmenge P mit einem gegebenen Zentrum c aufteilen in die optimalen 1-means-Kosten von P plus das $|P|$-fache der Distanz von c zum optimalen Zentrum, also dem Zentroiden.

Lemma 3.3.1. Sei $P \subset \mathbb{R}^d$ eine endliche Punktmenge mit dem geometrischen Zentroiden $z(P)$. Für jeden Punkt $q \in \mathbb{R}^d$ gilt:

$$\sum_{x\in P}\|x-q\|^2 = \sum_{x\in P}\|x-z(P)\|^2 + |P|\cdot\|q-z(P)\|^2$$

Wenn P eine gewichtete Punktmenge mit Gewichtsfunktion $w : P \to \mathbb{R}^+$ ist, dann gilt:

$$\sum_{x\in P} w(x)\cdot\|x-q\|^2 = \sum_{x\in P} w(x)\cdot\|x-z_w(P)\|^2 + W\cdot\|q-z_w(P)\|^2$$

Dabei ist $W = \sum_{x\in P} w(x)$ und $z_w(P) = \frac{1}{W}\sum_{x\in P} w(x)\cdot x$.

Die Kernmengenkonstruktion nutzt aus, dass der Anteil der optimalen 1-means-Kosten konstant in allen möglichen Clusterings ist. Die Berechnung des zweiten Summanden gestaltet sich als schwierig. Die Idee der Konstruktion besteht darin, diese Erkenntnisse auf das k-means-Problem zu verallgemeinern. Gegeben ein Clustering C, lässt sich die Punktmenge P in k Teile partitionieren, sodass der Zentroid und die 1-means-Kosten jeder Partition ausreichen, um die Gesamtkosten von C zu berechnen. C ist jedoch zuvor nicht bekannt.

Der Algorithmus partitioniert die ursprüngliche Punktmenge P rekursiv in Teilmengen, bis es für eine Teilmenge P' ausreicht, eine 1-means-Kernmenge zu verwenden, um die k-means-Kosten von P' zu approximieren. Sobald dies für alle Teilmengen möglich ist, wird nicht länger partitioniert und die Teilmengen werden durch ihre Zentroiden ersetzt. Die Zentroiden bilden dann die Punkte der Kernmenge, deren Gewicht die Kardinalität der jeweiligen Teilmenge ist. Wir unterscheiden bei den Teilmengen zwischen *k-unstrukturierten* Teilmengen und *k-strukturierten* Teilmengen. Eine k-unstrukturierte Teilmenge muss nicht weiter partitioniert werden. Für sie gilt, dass ihre 1-means-Kosten höchstens um einen Faktor $(1 + \epsilon')$ größer sind als die k-means-Kosten. Die k-strukturierten Teilmengen werden jeweils solange weiter partitioniert, bis nur noch k-unstrukturierte Teilmengen übrig sind. In der ursprünglichen Version des Algorithmus werden die k-strukturierten Teilmengen anhand der optimalen k-means-Lösung partitioniert. Alternativ wird die rekursive Partitionierung abgebrochen, wenn eine maximale Rekursionstiefe ν erreicht ist, um sicherzustellen, dass nicht zu lange partitioniert wird.

Algorithmus 9: `Coreset` [FSS13, Sch14]

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+, \epsilon$
Ausgabe: : Kernmenge S mit Gewichten w und Verschiebung Δ

$\epsilon' \leftarrow \epsilon^2/50$
$S \leftarrow \emptyset$
$\Delta \leftarrow 0$
$M \leftarrow$ `Partition`$(P, k, 0, \lceil \log_{1+\epsilon'} 1/\epsilon' \rceil, \epsilon')$
for *jede Teilmenge* $P' \in M$ **do**
    $S \leftarrow S \cup z(P')$
    $w(z(P')) \leftarrow |P'|$
    $\Delta \leftarrow \Delta + \sum_{x \in P'} \|x - z(P')\|^2$
return S, w, Δ

Algorithmus 9 ist eine Pseudocode-Beschreibung der Kernmengenkonstruktion. Die Punkte der Kernmenge werden in S gespeichert, die Gewichtung in w

und die Verschiebung in Δ. Wie zuvor beschrieben ruft `Coreset` die rekursive Partitionierungs-Routine `Partition` auf und ersetzt am Ende der Partitionierung die Teilmengen durch ihren Zentroiden, der dann als Punkt in die Kernmenge aufgenommen wird.

Die Partitionierung nimmt der Algorithmus 10 `Partition` vor. In der Menge M speichert diese alle berechneten Teilmengen. Die Zeilen 1-3 berechnen dabei eine neue Partitionierung, was in der ursprünglichen Variante des Algorithmus anhand einer optimalen k-means-Lösung erfolgt. Die Unterscheidung nach k-strukturierten und k-unstrukturierten Teilmengen erfolgt in Zeile 4. Der rekursive Aufruf der Partitionierung erfolgt in den Zeilen 7-8.

Algorithmus 10: `Partition` [FSS13, Sch14]

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+, t, \nu, \epsilon'$
Ausgabe: : Partitionierung M

1 Berechne optimale Menge von Zentren $C^* = \{c_1, \ldots, c_k\}$ für P
2 Sei $P_1, \ldots, P_k$ die durch C^* induzierte Partitionierung von P
3 Sei $z(P)$ der Zentroid von P
4 **if** $t = \nu$ **or** $\sum_{x \in P} \|x - z(P)\|^2 \leq (1 + \epsilon') \sum_{i=1}^{k} \sum_{y \in P_i} \|y - z(P_i)\|^2$ **then**
5 $M \leftarrow M \cup P$
6 **else**
7 **for** $i \leftarrow 1$ **to** k **do**
8 $M \leftarrow M \cup$ `Partition`$(P_i, k, t + 1, \nu, \epsilon')$
9 **return** M

In [Sch14] wird bereits thematisiert, dass die wiederholte Partitionierung anhand einer optimalen k-means-Lösung für die Praxis zu aufwändig wäre und zu unpraktikablen Laufzeiten führen würde. Schmidt schlägt daher vor, eine α-approximative Lösung zu verwenden. Dementsprechend wird dann in Zeile 1 von Algorithmus 10 eine Lösung C' berechnet, deren Kosten höchstens um einen Faktor α größer sind als die Kosten einer optimalen k-means-Lösung für P. Um zu entscheiden, ob die Teilmengen k-strukturiert sind, wird dann geprüft, ob

$$\sum_{x \in P'} \|x - z(P')\|^2 \leq ((1 + \epsilon')/\alpha) \sum_{i=1}^{k} \sum_{y \in P_i} \|y - z(P_i)\|^2$$

gilt. Die Analyse in [Sch14] zeigt, dass auch bei Verwendung der Approximation noch Garantien über Qualität und Laufzeit bewiesen werden können. Insbesondere berechnet der entsprechend modifizierte Algorithmus weiterhin eine $(1 + \epsilon)$-Kernmenge mit Verschiebung Δ, sofern α klein genug ist.

Wir wollen noch einen Schritt weiter gehen und die Partitionierung konstant approximativ vornehmen. Die Schwierigkeit besteht weiterhin darin, eine Lösung für das k-means-Problem im Partitionierungsschritt zu berechnen, die nur geringen Aufwand hat und dennoch in vielen Fällen Partitionierungen mit handhabbaren Kosten erzeugt. Wie schon zuvor sehen wir von einer vollkommen zufälligen Auswahl der Zentren ab und setzen stattdessen die D^2-Gewichtung von k-means++ ein. Dabei gehen wir folgendermaßen vor. Zunächst wählen wir initial $c \cdot k$ Zentren gemäß der D^2-Gewichtung aus der gesamten Punktmenge. Diese induzieren eine initiale Partitionierung $P_1, \ldots, P_{ck}$, die wir bestimmen, indem wir jeden Punkt dem nächstgelegenen Zentrum zuweisen. Anschließend führen wir rekursiv die folgende modifizierte, weitere Partitionierung durch. Bis zu einer maximalen Tiefe T (entspricht ν aus der ursprünglichen Konstruktion) prüfen wir für jede Teilmenge P', ob eine Partitionierung in weitere $d \cdot k$ Teilmengen um einen Faktor f billiger ist, als die 1-means-Kosten mit dem Zentroiden $z(P')$. Die $d \cdot k$ Teilmengen in der rekursiven Partitionierung wählen wir dabei wieder gemäß der D^2-Gewichtung in der jeweiligen Teilmenge aus. Wenn eine weitere Partitionierung anhand der $d \cdot k$ Zentren vorzuziehen ist, berechnen wir die neuen Teilmengen wieder über Zuweisung der Punkte zu den nächstgelegenen Zentren.

Algorithmus 11: `Coreset2`

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+, c, d, f, T$
Ausgabe: : Kernmenge S mit Gewichten w und Verschiebung Δ

$l \leftarrow c \cdot k$
$S \leftarrow \emptyset$
$M \leftarrow \emptyset$
$\Delta \leftarrow 0$
Wähle l Zentren $C^0 = \{c_1, \ldots, c_l\}$ gemäß D^2-Gewichtung aus P
Sei $P_1, \ldots, P_l$ die durch C^0 induzierte Partitionierung von P
for $i \leftarrow 1$ **to** l **do**
    $M_i \leftarrow$ `Partition2`$(P_i, k, d, f, 1, T)$
    $M \leftarrow M \cup M_i$
for *jede Teilmenge* $P' \in M$ **do**
    $S \leftarrow S \cup z(P')$
    $w(z(P')) \leftarrow |P'|$
    $\Delta \leftarrow \Delta + \sum_{x \in P'} \|x - z(P')\|^2$
return S, w, Δ

Sobald bei allen Teilmengen die Partitionierung abgebrochen ist, ersetzen wir jede Teilmenge durch ihren Zentroiden. Die Vereinigung aller Zentroiden entspricht dann wieder den Punkten der Kernmenge. Die Gewichtung w und die Verschiebung

Δ werden ebenfalls analog berechnet.

Algorithmus 11 ist die Pseudocode-Beschreibung unserer Variante der Kernmengenkonstruktion. In den Zeilen 5-6 wird die initiale Partitionierung gemäß D^2-Gewichtung vorgenommen. Anschließend führen wir in den Zeilen 7-9 die rekursive Partitionierung für jede der Teilmengen solange durch, bis entweder die maximale Tiefe erreicht wurde oder die Abbruchbedingung über die Kosten einer weiteren Partitionierung erreicht ist. Wir vereinigen alle berechneten Teilmengen zur Menge M. In den Zeilen 10-14 gehen wir analog zur ursprünglichen Konstruktion vor. Alle Teilmengen werden wieder durch ihren Zentroiden ersetzt.

Algorithmus 12: `Partition2`

Eingabe: : $P \subseteq \mathbb{R}^d, k \in \mathbb{N}^+, d, f, t, T$
Ausgabe: : Partitionierung M

1 $m \leftarrow d \cdot k$
2 $M \leftarrow \emptyset$
3 Wähle m Zentren $C^t = \{c_1, \ldots, c_m\}$ gemäß D^2-Gewichtung aus P
4 Sei $P_1, \ldots, P_m$ die durch C^t induzierte Partitionierung von P
5 Sei $z(P)$ der Zentroid von P
6 **if** $t = T$ **or** $\sum_{x \in P} \|x - z(P)\|^2 \leq \frac{1}{f} \sum_{i=1}^{m} \sum_{y \in P_i} \|y - z(P_i)\|^2$ **then**
7 $\quad M \leftarrow M \cup P$
8 **else**
9 $\quad$ **for** $i \leftarrow 1$ **to** m **do**
10 $\quad\quad M \leftarrow M \cup$ `Partition2`$(P_i, k, d, f, t+1, T)$
11 **return** M

Algorithmus 12 ist unsere Variante der Partitionierung. In den Zeilen 3-5 wählen wir $d \cdot k$ Zentren gemäß D^2-Gewichtung und prüfen in Zeile 6 die modifizierte Abbruchbedingung. Ansonsten ist die Partitionierung unverändert.

Bezüglich der Laufzeit müssen wir jeden Punkt der Dimension d mindestens einmal betrachten. Damit erhalten wir eine Mindestlaufzeit von $\mathcal{O}(nd)$. Hinzu kommt ein Faktor für die Anzahl an Punkten der Kernmenge, die wir anhand der D^2-Gewichtung aus der Eingabepunktmenge beziehungsweise deren Partitionen auswählen. Dies sind in der initialen Partitionierung $c \cdot k$ viele. In der rekursiven Partitionierung kommen maximal $\mathcal{O}((dk)^T)$ viele hinzu, wenn bis zur maximalen Rekursionstiefe T in jeder Ebene und jeder Teilmenge weiter partitioniert wird. Insgesamt erhalten wir damit eine obere Laufzeit-Schranke von $\mathcal{O}\left(nd \cdot \left(ck + (dk)^T\right)\right)$.

Wir können bei unserer Variante der Kernmengenkonstruktion keine Garantien für

die Qualität der berechneten Kernmenge mehr geben. Es ist daher unerlässlich, die Konstruktion empirisch auszuwerten. Diese Auswertung wollen wir in Abschnitt 4.4 des nächsten Kapitels vornehmen.

4 Experimentelle Evaluation

In diesem Kapitel wollen wir empirisch evaluieren, wie performant unsere Algorithmen sind.

Wir gehen folgendermaßen vor: wir beschreiben zunächst in Abschnitt 4.1 die Umgebung, unter der wir unsere Experimente durchgeführt haben. Damit wollen wir die Reproduzierbarkeit unserer Ergebnisse garantieren und eine grobe Vorstellung von der Leistungsfähigkeit der eingesetzten Hardware ermöglichen. Wir stellen in diesem Abschnitt auch die eingesetzten Eingabedaten vor.

In Abschnitt 4.2 untersuchen wir einleitend zunächst, inwieweit der Einsatz der Kernel Methode tatsächlich die Qualität von Clusterings verbessern kann. Außerdem untersuchen wir insbesondere, ob der Overhead, der durch die Verwendung eines Kernels entsteht, in der Praxis tatsächlich so gering ist, wie wir zuvor bei der Einführung des Kernel-Tricks argumentiert haben.

In Abschnitt 4.3 diskutieren wir, ob wir mit dem Kernel-k-means++-Algorithmus den gewünschten Trade-off im Bereich der spektralen Clusteranalyse von Graphen erzielen konnten. Dazu betrachten wir einerseits Experimente mit den klassischen Graphclustering-Zielfunktionen und führen zusätzlich die vielleicht wichtigste Anwendung in der Bildsegmentierung durch.

4.1 Experimentelle Rahmenbedingungen

Um Vergleichbarkeit der Testläufe sicherzustellen, wurden sämtliche hier vorgestellten Ausführungen der Algorithmen auf derselben Hardware durchgeführt. Der Rechner wurde unter 64 Bit Linux (Arch Linux) betrieben und verfügte über einen Intel i7 3,5 GHz Vierkern-Prozessor sowie 16 GB Arbeitsspeicher. Die Laufzeiten der Algorithmen wurden jeweils über die reine Ausführung des jeweiligen Algorithmus ermittelt, das heißt, Vorverarbeitung, wie beispielsweise das Einlesen von Daten aus einer Datei, wird in den angegebenen Laufzeiten nicht berücksichtigt.

Die Algorithmen wurden mit einer Ausnahme in C++ implementiert. Als Compiler wurde die GCC in der Version 4.9.2 verwendet. Der Code wurde jeweils für 64-Bit Architekturen übersetzt, das heißt, es wurde das GCC-Flag `-m64` gesetzt. Außerdem wurden für die Optimierung die Flags `-O3` und `-fPIC` gesetzt. Lediglich für die Bildsegmentierung wurde MATLAB verwendet.

Für die Graphclustering-Experimente in Abschnitt 4.3 wurden Graphen aus dem „Graph Partitioning Archive“ [SWC04] eingesetzt. Die Daten stammen aus Anwendungsfällen, in denen Graphpartitionierungen berechnet werden müssen,

Datensatz	Anzahl Punkte	Clusteranzahl k
BENSAID	49	3
DUNN	90	2
IRIS	150	3
ECOLI	336	7
CIRCLE	108	2
BLE-3	320	3
BLE-2	312	2
UE-4	262	4
UE-3	377	3
ULE-4	298	4

Tabelle 2: Eigenschaften der Datensätze aus [BPKK99].

und wurden von diversen Forschern im Laufe der Zeit bereitgestellt. Das Archiv ist unter der folgenden URL abrufbar: `http://staffweb.cms.gre.ac.uk/~wc06/partition/` (Stand: 02.02.2015).

Für die Experimente in den Abschnitten 4.2 und 4.4 wurden Clusteringeingaben aus dem „UC Irvine Machine Learning Repository“ [Lic13] verwendet. Das UCI Machine Learning Repository ist unter der folgenden URL erreichbar: `http://archive.ics.uci.edu/ml/` (Stand: 02.02.2015). Es enthält über 300 Datensätze aus Anwendungen der Community für Maschinelles Lernen, von denen einige Clusteringprobleme sind. Das jeweilige zu lösende Problem kann in der durchsuchbaren Übersicht unter `http://archive.ics.uci.edu/ml/datasets.html` dem Attribut „Default Task“ entnommen werden. Die meisten Datensätze sind Klassifizierungsprobleme. Wir haben uns auf Datensätze beschränkt, die als Clustering-Probleme ausgezeichnet sind, da nicht bei allen Datensätzen klar wird, aus welchem Kontext beziehungsweise welcher Anwendung die Daten stammen.

4.2 Die Kernel Methode

Wir wollen einleitend zunächst untersuchen, ob die Kernel Methode tatsächlich signifikante Verbesserungen der Clusteringqualität erbringen kann. Wir betrachten zunächst die Güte der berechneten Clusterings der Algorithmen k-means (Lloyds Algorithmus) und Kernel-k-means++. Wir bereiten dazu Ergebnisse des Papiers [KLLL05] auf, welches sich mit exakt dieser Fragestellung beschäftigt. Die Datensätze, mit denen diese Experimente nachgestellt wurden, stammen aus diversen Papieren, deren Ergebnisse in [BPKK99] zusammengefasst sind. Tabelle 2 zeigt, dass es sich um kleine Datensätze handelt. Für diese Datensätze sind jedoch die k-means-Zielfunktionswerte der jeweils optimalen Lösungen bekannt [BPKK99].

Diese konnten vermutlich gerade aus dem Grund, dass die Datensätze sehr klein sind, mit überschaubarem Rechenaufwand ermittelt werden. Zudem decken die Datensätze ein großes Spektrum an geometrischen Clustern ab, insbesondere sind hyperspherische und hyperellipsoide Cluster enthalten. Graphische Plots der Cluster sind in [KLLL05] enthalten.

Um den klassischen k-means-Algorithmus mit dem Kernel-k-means-Algorithmus zu vergleichen, haben wir die beiden Implementierungen der Algorithmen in der Dlib-Bibliothek[1] auf die Datensätze angewandt. Für den Kernel-k-means-Algorithmus wurde wie in [KLLL05] der Gauss-/RBF-Kernel verwendet. Wir haben diesen für unsere Experimente mit $\sigma = 0.1$ parametrisiert. In [KLLL05] finden sich keine Angaben über die Wahl des Parameters, die geringfügigen Abweichungen unserer Ergebnisse stammen daher vermutlich von der Wahl von σ. Zudem wurde die maximale Anzahl an Iterationen auf 100 beschränkt.

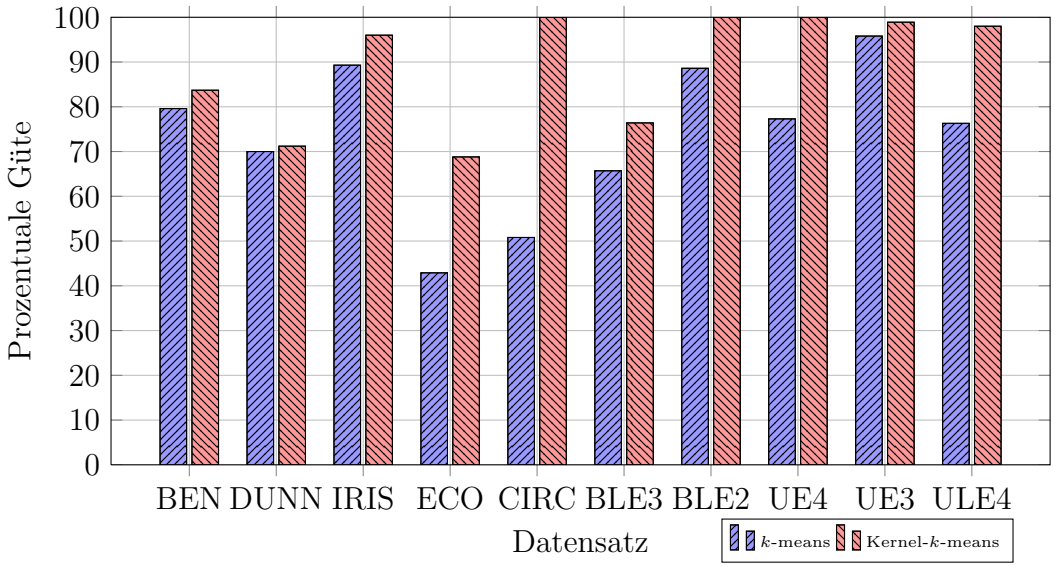


Abbildung 1: Die Clusteringqualität von k-means und Kernel-k-means.

Wir haben die prozentuale Güte ermittelt, indem wir jeweils die optimalen Kosten durch die Kosten der ermittelten Lösung dividiert haben. Abbildung 1 und Tabelle 3 zeigen, dass die Kernel Methode bei geometrischem Clustering auf allen Instanzen zu einem besseren Clustering führt. Auf den schwer zu clusternden Instanzen BENSAID und DUNN beträgt die Verbesserung schon einige Prozentpunkte, während das Potenzial der Kernel Methode auf den Instanzen ECOLI und CIRCLE besonders gut zu erkennen ist. Dabei handelt es sich um Instanzen, auf denen die Cluster eine hyperellipsoide beziehungsweise hyperspherische Form haben. Auf diesen Instanzen clustert der k-means-Algorithmus wegen der Hyperebenen-Beschränkung nur sehr

[1] Version 18.12, `http://dlib.net/` (Stand: 01.02.2015)

Datensatz	k**-means**	**Kernel-**k**-means**
BENSAID	79.6	83.7
DUNN	70	71.2
IRIS	89.3	96
ECOLI	42.9	68.8
CIRCLE	50.8	100
BLE-3	65.7	76.4
BLE-2	88.6	100
UE-4	77.3	100
UE-3	95.8	98.9
ULE-4	76.3	98

Tabelle 3: Die Clusteringqualität von k-means und Kernel-k-means.

schlecht, während mit der Kernel Methode auch nicht linear separierte Cluster möglich sind. Bemerkenswert ist auch, dass der Kernel-k-means-Algorithmus sogar auf Instanzen mit mehreren hundert Punkten noch die optimale Lösung berechnet, wie man an den Instanzen CIRCLE, BLE-2 und UE-4 erkennen kann.

Die Datensätze sind nicht geeignet, um Laufzeiteinbußen bei der Kernel Methode zu ermitteln, da beide Algorithmen auf Instanzen dieser Größe nahezu unmittelbar terminieren. Wir verwenden daher größere Datensätze aus dem UCI Machine Learning Repository. Tabelle 4 gibt einen Überblick über die Beschaffenheit der Datensätze.

Datensatz	**Anzahl Punkte**	**Dimension** d
NYSK	10241	7
CENSUS	2458285	68
BAG-OF-WORDS	8000000	100000
PLANTS	22632	70
130-US	100000	55
GFE	27965	100
YOUTUBE	120000	1000000

Tabelle 4: Eigenschaften der Datensätze aus [Lic13].

In Abbildung 2 und Tabelle 5 ist gut zu erkennen, dass der Overhead, der durch die Kernel Methode entsteht, bei den kleinen Instanzen (bis zu 100.000 Punkten) vernachlässigbar gering ist. Auf den größeren Instanzen sind Laufzeiteinbußen in der Größenordnung von rund 15% zu verzeichnen. Diese sind nicht zu vernachlässigen. In Relation zur potenziellen Qualitätssteigerung aus den vorhergegangenen Expe-

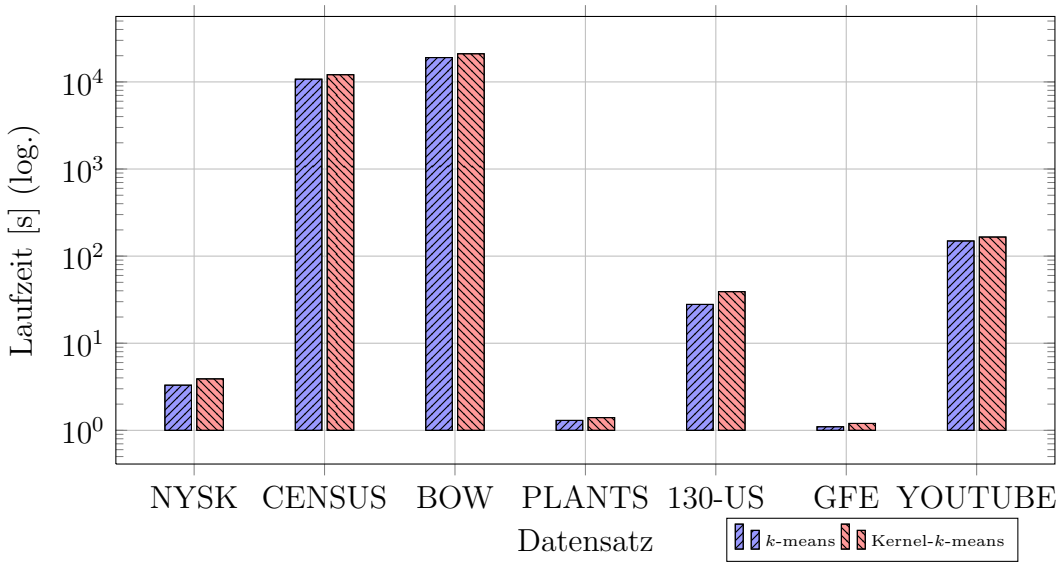


Abbildung 2: Die Laufzeiten von k-means und Kernel-k-means.

rimenten betrachtet, handelt es sich jedoch um einen Trade-off, der sich in den allermeisten Fällen lohnt.

Datensatz	**k-means**	**Kernel-k-means**
NYSK	3.3	3.9
CENSUS	10750.4	12097.2
BOW	19033.3	21048.1
PLANTS	1.3	1.4
130-US	27.9	39.1
GFE	1.1	1.2
YOUTUBE	149.3	165.7

Tabelle 5: Die Laufzeiten von k-means und Kernel-k-means in Sekunden.

Wir diskutieren im Folgenden die experimentellen Ergebnisse im Zusammenhang mit dem Kernel-k-means++-Algorithmus.

4.3 Der Kernel-k-means++-Algorithmus

Für den Kernel-k-means++-Algorithmus haben wir das Framework „Graclus“[2] von Kulis und Guan in der Version 1.2 erweitert. Dabei handelt es sich um eine Implementierung des gewichteten Kernel-k-means Algorithmus 7 für die spektrale Clusteranalyse beziehungsweise Graphpartitionierung. Eine detaillierte Beschreibung, die recht nah an der Implementierung orientiert ist, haben die Autoren

[2] `http://www.cs.utexas.edu/users/dml/Software/graclus.html` (Stand: 27.01.2015)

Zielfunktion	Knotengewichte	Kernelmatrix
Ratio Association	1 für alle Knoten	$K = \sigma I + A$
Ratio Cut	1 für alle Knoten	$K = \sigma I - L$
Kernighan-Lin	1 für alle Knoten	$K = \sigma I - K$
Normalized Cut	Grad des Knoten	$K = \sigma D^{-1} + D^{-1}AD^{-1}$

Tabelle 6: Die σ-verschobenen Kernelmatrizen für die Graphschnitt-Zielfunktionen [DGK04].

in [DGK07] gegeben. Graclus ist im Bereich quelloffener Graphpartitionierungs-Software gegenwärtig das vielleicht schnellste Framework, wie ein Vergleich im „Berkeley Segmentation Dataset and Benchmark“[3] vermuten lässt. Graclus wird in einer Reihe von quelloffener Bildverarbeitungs-Software verwendet. Das prominenteste Beispiel, das Graclus verwendet, ist vermutlich „Clustering Views for Multi-view Stereo (CMVS)“[4].

Bevor wir unsere Experimente vorstellen, wollen wir kurz auf ein bislang offen gebliebenes Problem aus Abschnitt 3.1 eingehen. Wir hatten am Ende des Abschnitts angemerkt, dass wir für eine konkrete Implementierung des Algorithmus sicherstellen müssen, dass die verwendeten Kernelmatrizen positiv definit sind. In [DGK07] wird detailliert diskutiert, dass für alle Graphpartitionierungs-Zielfunktionen gezeigt werden kann, dass eine einfache diagonale Verschiebung der ursprünglichen Kernelmatrix K um einen Wert σ die Analogie über die Formulierungen als Spurmaximierungsprobleme intakt bleibt. Dabei muss σ muss groß genug gewählt werden, damit der entsprechend verschobene Kernel K' eine positiv definite Matrix ist. Für manche Eingaben kann es vorkommen, dass die Gewichtsmatrix W so beschaffen ist, dass die Kernelmatrix auch ohne eine diagonale Verschiebung schon positiv definit ist. Die Experimente in [DGK04, DGK07] legen nahe, dass es in solchen Fällen sogar vorteilhaft sein kann, eine negative σ-Verschiebung vorzunehmen. Empirisch wird σ idealerweise so gewählt, dass die Spur der Kernelmatrix möglichst nah bei Null liegt. Tabelle 6 zeigt die σ-verschobenen Kernelmatrizen für die diversen Graphschnitt-Zielfunktionen. Wie bereits erwähnt sind für unsere Zwecke nur die Ratio Association und der Normalized Cut relevant.

Graphpartitionierung. Graclus selbst bietet drei Möglichkeiten der Initialisierung an: zufällige Initialisierung (analog zum klassischen Algorithmus von Lloyd),

[3] `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/` (Stand: 29.01.2015)

[4] `http://www.di.ens.fr/cmvs/` (Stand: 29.01.2015)

Datensatz	Anzahl Knoten	Anzahl Kanten
BCSSTK31	35588	572914
T60K	60005	89440
598A	110971	741934
144	144649	1074393
M14B	214765	1679018
AUTO	448695	3314611

Tabelle 7: Eigenschaften der Datensätze aus dem Graph Partitioning Archive [SWC04].

spektrale Initialisierung (entspricht der Initialisierung über Eigenwert-Berechnung, die wir zu Beginn von Abschnitt 3.2 diskutiert haben) und Initialisierung durch METIS. METIS [KK98] ist ein state-of-the-art Graphpartitionierungs-Framework, das in einem mehrschrittigen Verfahren Cluster gleicher Größe in Graphen berechnet. Es diente bereits in [DGK04, DGK07] als Benchmark-Vergleichsinstanz. Wir werden METIS auch in unseren Experimenten für Vergleiche heranziehen. Die gemessenen Werte beziehen sich jeweils auf die *gesamte* Ausführung des Algorithmus, also nicht nur auf die Initialisierung.

Wegen der Zufallskomponente im Kernel-k-means++-Algorithmus haben wir diesen sowie die vollkommen zufällige Initialisierung pro Datensatz und pro Clusterzahl k insgesamt 100 mal ausgeführt. In unseren Ergebnissen geben wir für die zufällige Initialisierung jeweils das arithmetische Mittel über alle 100 Läufe an. Für den Kernel-k-means++-Algorithmus geben wir zusätzlich die gemessenen Minima und Maxima über alle Durchläufe an, um eine Vorstellung der Größenordnung der Schwankungen zu ermöglichen. Wir betrachten insbesondere auch die Laufzeiten der Algorithmen, um den Trade-off zwischen Clusteringqualität und Ausführungszeit beurteilen zu können. Wir untersuchen die erzielten Werte für Ratio Association (je *größer* der Wert, desto besser ist die Qualität des berechneten Clusterings) und Normalized Cut (je *kleiner* der Wert, desto besser die Qualität des berechneten Clusterings). Für jeden Datensatz und jede Zielfunktion berechnen wir mit den Algorithmen jeweils Clusterings mit 128 und 512 Clustern. Damit erreichen wir eine höhere Anforderung an die spektralen Algorithmen. Wir haben bereits thematisiert, dass bei der Performanz für diese die Anzahl der Cluster kritisch ist, da sie die Anzahl der zu berechnenden Eigenwerte erhöht. Schließlich sei bereits vorab erwähnt, dass die von uns gemessenen Werte bezüglich der Laufzeit teilweise stark von denen aus [DGK07] (nach unten) abweichen. Dies ist mit großer Wahrscheinlichkeit auf die deutlich leistungsfähigere Hardware zurückzuführen:

in [DGK07] wurde ein Rechner mit einem 2.4 GHz Pentium IV-Prozessor und nur 1 GB Arbeitsspeicher eingesetzt. Es ist zudem naheliegend, dass der Prozessor über einen 32 Bit Chipsatz verfügte und Graclus dementsprechend auch für 32 Bit-Architekturen kompiliert wurde.

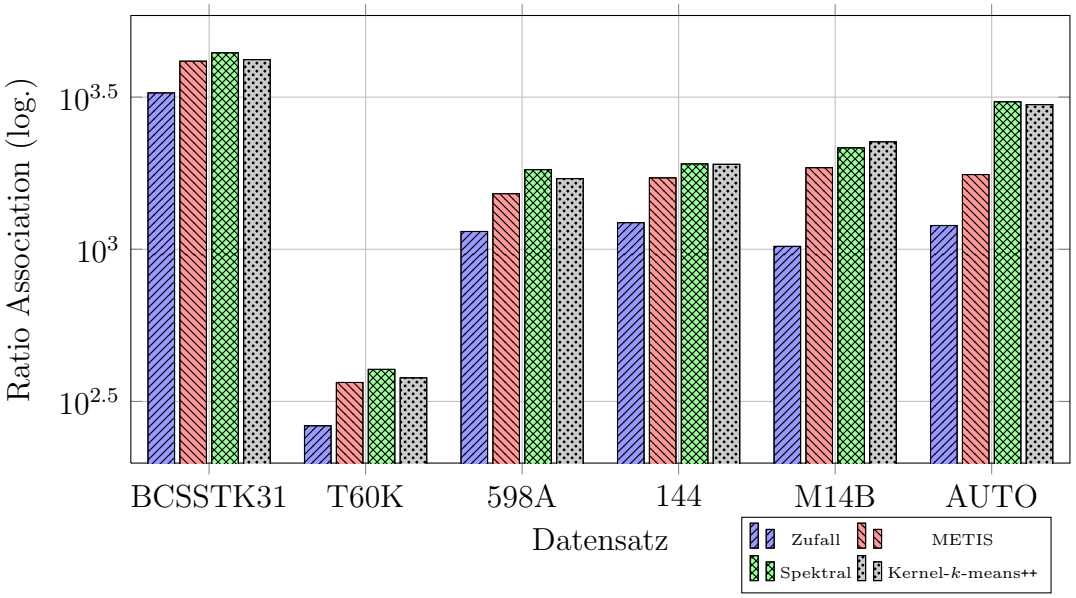


Abbildung 3: Ratio Association Werte der Algorithmen für $k = 128$ (je größer, desto besser).

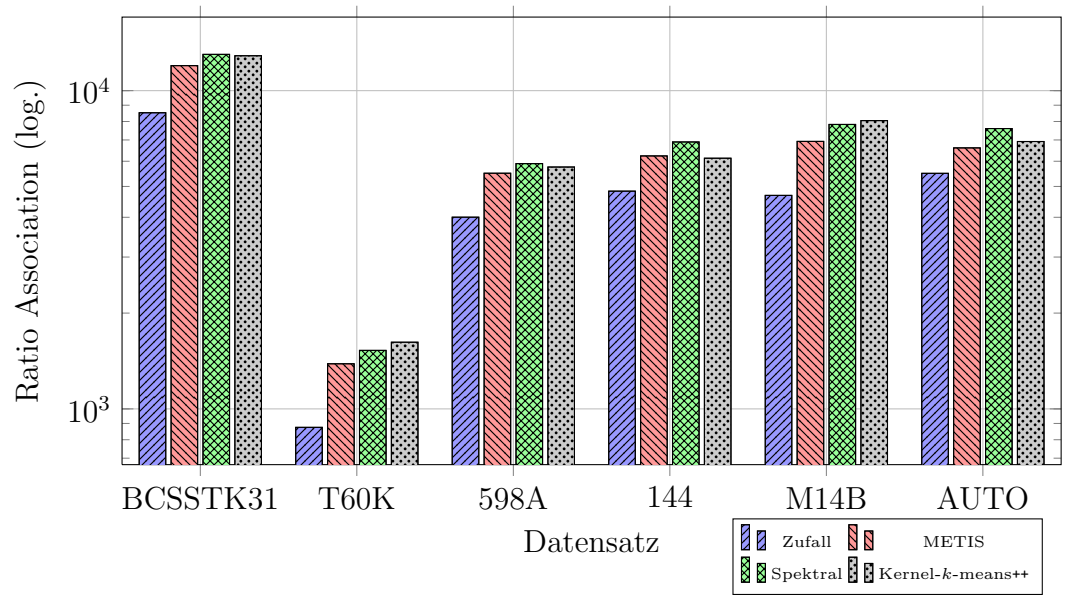


Abbildung 4: Ratio Association Werte der Algorithmen für $k = 512$ (je größer, desto besser).

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	3265	4150	4422	4199	3955	4322
T60K	263	365	403	378	345	385
598A	1144	1523	1827	1707	1622	1754
144	1223	1717	1908	1902	1521	2334
M14B	1022	1854	2155	2255	2133	2552
AUTO	1197	1760	3052	2988	2852	3128

Tabelle 8: Ratio Association Werte der Algorithmen für $k = 128$ (je größer, desto besser).

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	8522	11984	13002	12873	3955	4322
T60K	875	1386	1527	1620	345	385
598A	4003	5502	5899	5755	1622	1754
144	4833	6236	6899	6133	1521	2334
M14B	4687	6927	7832	8052	2133	2552
AUTO	5498	6610	7603	6922	2852	3128

Tabelle 9: Ratio Association Werte der Algorithmen für $k = 512$ (je größer, desto besser).

Die Abbildungen 3 und 4 beziehungsweise die Tabellen 8 und 9 zeigen die Güte der berechneten Clusterings für die Ratio Association. Es ist zunächst gut zu erkennen, dass wir die Ergebnisse von [DGK04] in etwa bestätigt haben: sowohl die Initialisierung mit METIS, als auch die spektrale Initialisierung liefern deutlich bessere Clusterings als die rein zufällige Initialisierung. Der Algorithmus Kernel-k-means++ ist auf allen Instanzen mindestens kompetitiv und auf einigen sogar noch besser. Bemerkenswert ist, dass die Verbesserung des Zielfunktionswertes mit geringerer Anzahl an Clustern prozentual größer ist als bei einer größeren Anzahl an Clustern. So werden beispielsweise bei 128 Clustern im Schnitt um 50% bessere Ergebnisse erzielt, während bei 512 Clustern eher rund 20%-30% Verbesserung erzielt werden. Die maximal erzielten Werte für Kernel-k-means++ zeigen, dass in fast allen Testläufen mit der D^2-Gewichtung eine zufällige Initialisierung stattgefunden hat, welche die spektrale Initialisierung verbessert. Wie bereits erwähnt gilt dies im arithmetischen Mittel jedoch nicht bei allen Instanzen.

Wir betrachten als nächstes die Laufzeiten der Algorithmen. Wir haben gerade dabei von Clusterings mit kleinerer Clusterzahl abgesehen, da die Laufzeiten für kleinere Werte von k so gering sind, dass sich Unterschiede kaum bemerkbar machen.

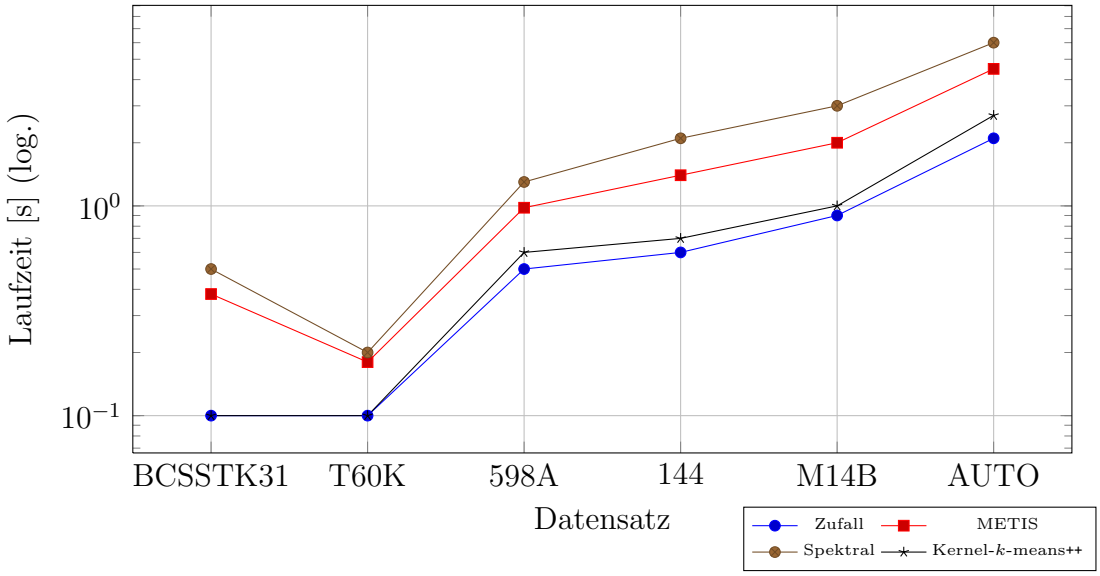


Abbildung 5: Laufzeiten der Algorithmen für Ratio Association mit $k = 128$.

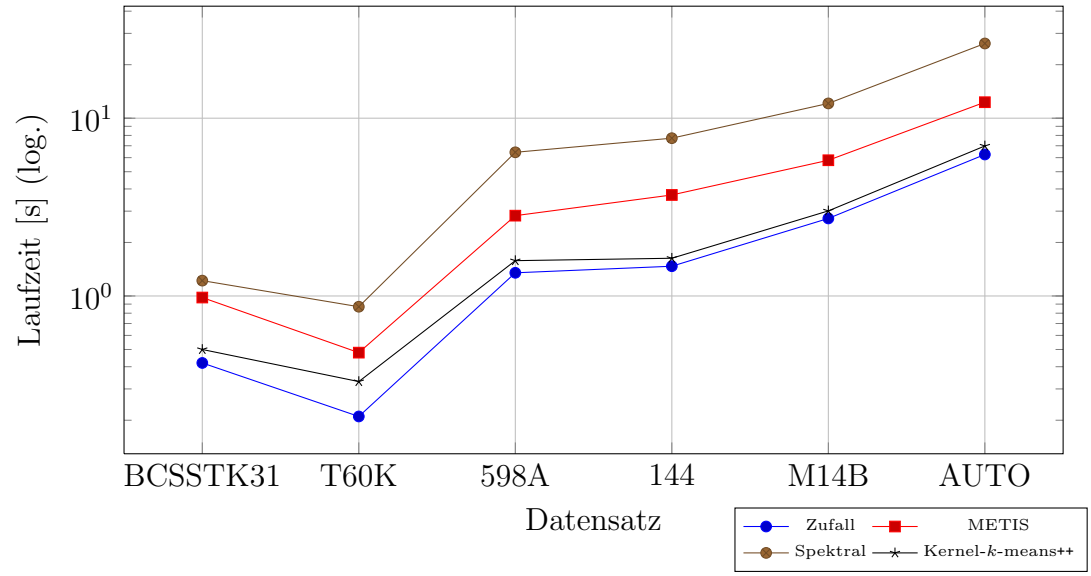


Abbildung 6: Laufzeiten der Algorithmen für Ratio Association mit $k = 512$.

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	0.1	0.38	0.51	0.1	0.1	0.1
T60K	0.1	0.18	0.25	0.1	0.1	0.1
598A	0.51	0.98	1.35	0.63	0.5	0.66
144	0.62	1.43	2.16	0.74	0.6	0.91
M14B	0.95	2.0	3	1.0	1.0	1.19
AUTO	2.13	4.5	6	2.72	2.43	3.27

Tabelle 10: Laufzeiten der Algorithmen für Ratio Association mit $k = 128$.

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	0.42	0.98	1.22	0.5	0.47	0.71
T60K	0.21	0.48	0.87	0.33	0.29	0.39
598A	1.35	2.83	6.43	1.58	1.42	1.66
144	1.47	3.7	7.72	1.63	1.55	1.74
M14B	2.73	5.8	12.11	3.01	2.91	3.34
AUTO	6.25	12.3	26.32	6.98	6.55	7.88

Tabelle 11: Laufzeiten der Algorithmen für Ratio Association mit $k = 512$.

Sowohl für $k = 128$ als auch für $k = 512$ erfolgt die Ausführung mit zufälliger Initialisierung um mehr als einen Faktor von 2 schneller als mit METIS oder spektralen Methoden. Die verbesserte Clusteringqualität erhalten wir bei beiden Verfahren unter Einsatz von Rechenzeit, die nicht zu unterschätzen ist. Grundsätzlich ist zwar anzumerken, dass die erzielten Laufzeiten generell sehr gering sind, wenn man bedenkt, dass die größten verwendeten Instanzen mehrere hunderttausend Knoten und einige Millionen Kanten haben. Für die massenhafte Verarbeitung von Graphen dieser Größenordnung sind die Laufzeiteinbußen dennoch beachtlich. Beim Algorithmus Kernel-k-means++ ist dagegen gut zu beobachten, dass die Laufzeiten kompetitiv zur rein zufälligen Initialisierung sind. Der zusätzliche Overhead für die Kernel Methode entspricht relativ in etwa dem, was wir in den Experimenten in Abschnitt 4.2 beobachten konnten. Insgesamt haben wir eine deutliche Verbesserung erreicht, wenn man bedenkt, dass auch die Qualität der von Kernel-k-means++ berechneten Clusterings kompetitiv ist.

Zudem kann Abbildung 6 und Tabelle 11 entnommen werden, dass der Abstand zwischen dem spektralen Verfahren und den anderen Methoden gerade bei großer Clusteranzahl k mehrere Größenordnungen annimmt. Dies bestätigt unsere zuvor formulierte Annahme, dass der Aufwand für die Berechnung der Eigenwerte für große Clusterzahlen sehr hoch wird. Wir wollen am Ende dieses Abschnitts kurz die Verwendung von Kernel-k-means++ für die Bildsegmentierung thematisieren.

Für die Bildsegmentierung hat sich der Normalized Cut als geeignete Zielfunktion in der Graphpartitionierung erwiesen [SM00]. Wir haben daher die Experimente auch für den Normalized Cut durchgeführt und stellen die Ergebnisse im Folgenden vor, bevor wir uns der direkten Anwendung in der Bildsegmentierung widmen.

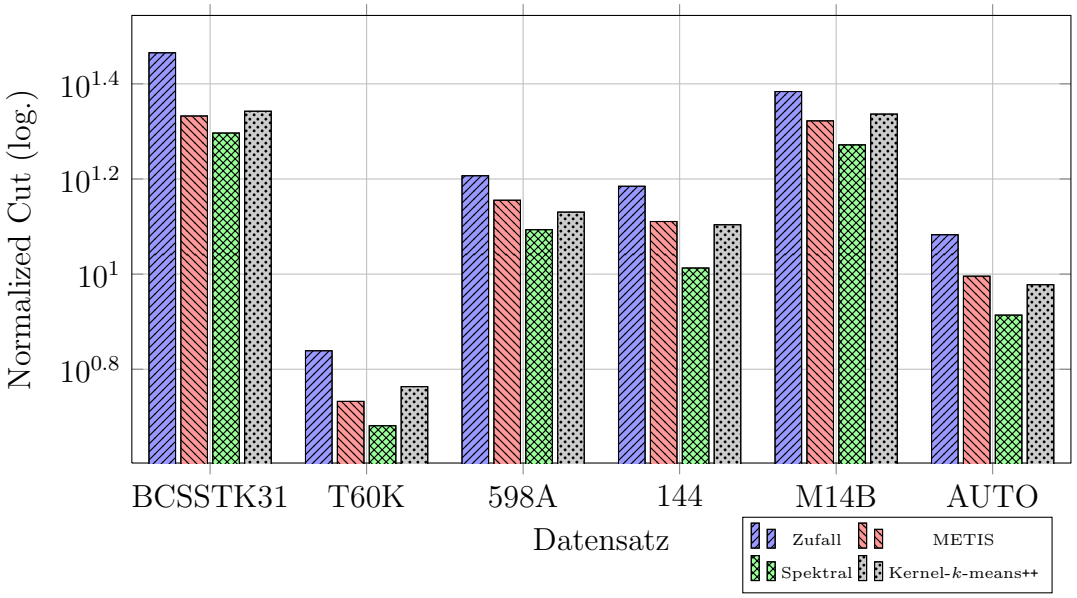


Abbildung 7: Normalized Cut Werte der Algorithmen für $k = 128$ (je kleiner, desto besser).

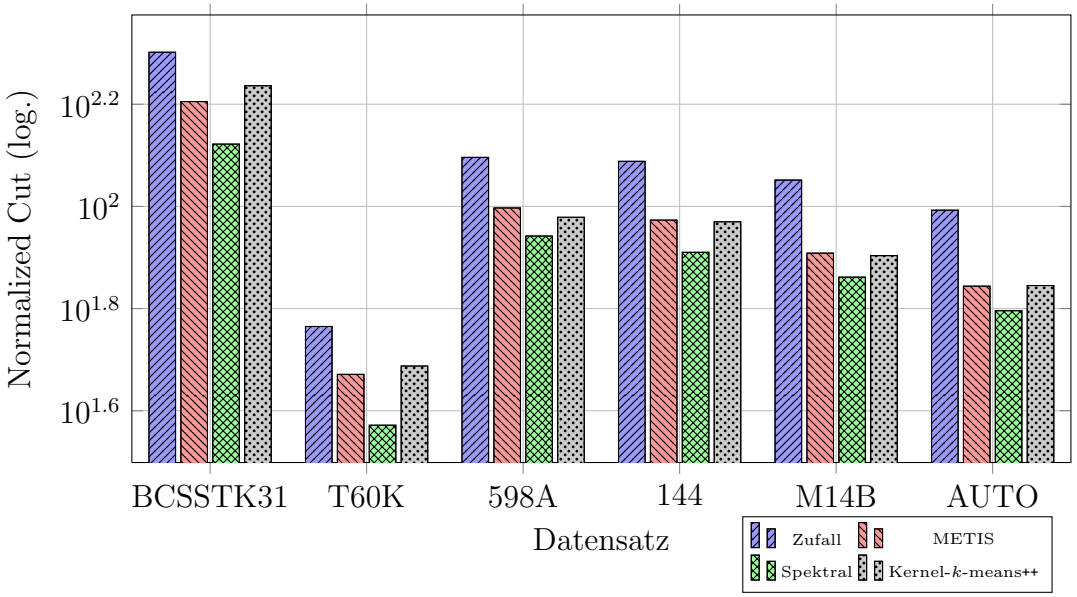


Abbildung 8: Normalized Cut Werte der Algorithmen für $k = 512$ (je kleiner, desto besser).

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	29.2	21.5	19.8	22	19.3	23.2
T60K	6.9	5.4	4.8	5.8	5.1	6.3
598A	16.1	14.3	12.4	13.5	13.1	13.9
144	15.3	12.9	10.3	12.7	12.4	13.0
M14B	24.2	21	18.7	21.7	21.1	22.6
AUTO	12.1	9.9	8.2	9.5	8.9	10.1

Tabelle 12: Normalized Cut Werte der Algorithmen für $k = 128$ (je kleiner, desto besser).

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	200.3	160.4	132.4	172.3	161.1	180.6
T60K	58.2	46.9	37.3	48.7	45.2	51.3
598A	124.7	99.3	87.5	95.2	90.3	100.9
144	122.5	94	81.3	93.3	89.2	98.8
M14B	112.6	81	72.7	80.1	71.2	87.8
AUTO	98.3	69.8	62.5	70	64.3	76.2

Tabelle 13: Normalized Cut Werte der Algorithmen für $k = 512$ (je kleiner, desto besser).

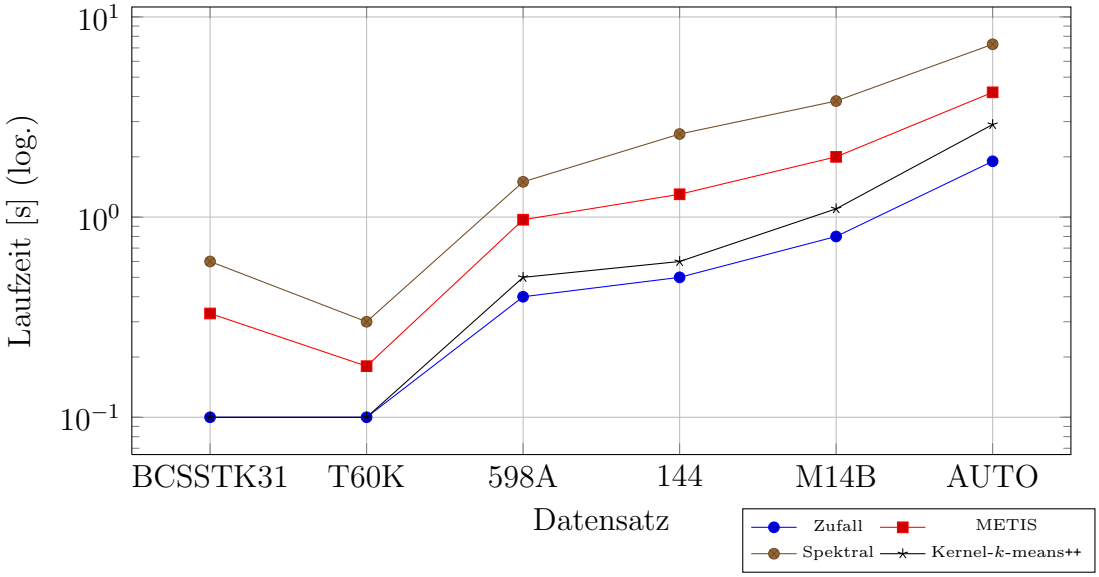


Abbildung 9: Laufzeiten der Algorithmen für Normalized Cut mit $k = 128$.

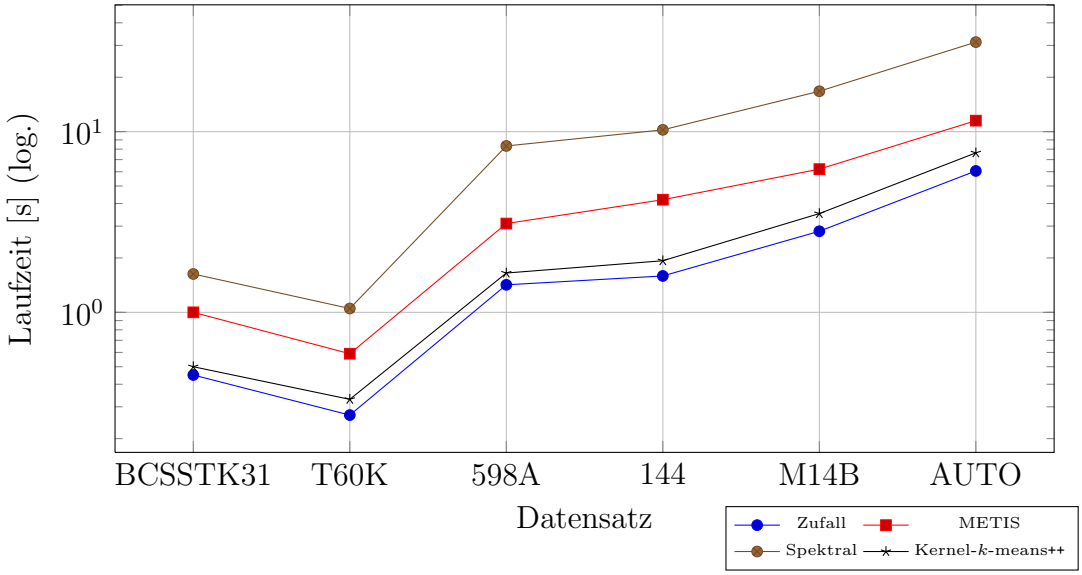


Abbildung 10: Laufzeiten der Algorithmen für Normalized Cut mit $k = 512$.

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	0.1	0.33	0.6	0.1	0.1	0.1
T60K	0.1	0.18	0.3	0.1	0.1	0.1
598A	0.4	0.97	1.5	0.5	0.42	0.68
144	0.5	1.3	2.6	0.6	0.53	0.93
M14B	0.8	2.0	3.8	1.1	0.95	1.24
AUTO	1.9	4.2	7.3	2.9	2.37	3.39

Tabelle 14: Laufzeiten der Algorithmen für Normalized Cut mit $k = 128$.

Datensatz	Zufall	METIS	Spektral	Avg	Min	Max
BCSSTK31	0.45	1.0	1.63	0.5	0.48	0.77
T60K	0.27	0.59	1.05	0.33	0.28	0.48
598A	1.42	3.1	8.33	1.65	1.49	1.88
144	1.59	4.2	10.23	1.93	1.66	1.98
M14B	2.81	6.2	16.73	3.52	3.01	3.99
AUTO	6.06	11.5	31.29	7.63	7.12	8.48

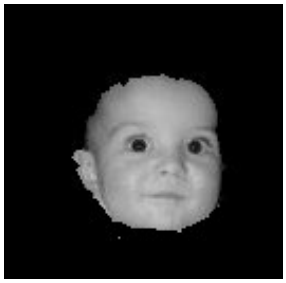
Tabelle 15: Laufzeiten der Algorithmen für Normalized Cut mit $k = 512$.

Die gemessenen Werte für den Normalized Cut sind in etwa in derselben relativen Größenordnung wie schon bei der Ratio Association. Qualitativ schneidet der Algorithmus Kernel-k-means++ nicht ganz so gut ab wie bei der Ratio Association, da er hier auf keiner Instanz die spektrale Initialisierung verbessert. Bei der Laufzeit zeigen sich dafür aber besonders deutliche Unterschiede. Insbesondere die spektrale Initialisierung ist bei den Experimenten mit großer Clusteranzahl sehr deutlich abgeschlagen.

Bildsegmentierung. Wir wollen abschließend wie angekündigt den Kernel-k-means++-Algorithmus zur Bildsegmentierung einsetzen. Dazu verwenden wir die Normalized Cut Zielfunktion und nutzen die MATLAB-Schnittstelle von Graclus, um die Cluster graphisch zu plotten. Wir haben zu diesem Zweck die Grafiken zunächst mit Code für „Image Segmentation with Normalized Cuts“ von Jianbo Shi[5] vorverarbeitet, wie in der Graclus-Dokumentation beschrieben. Für die Gesichtserkennung bietet sich eine Clusteranzahl von $k = 2$ an. Die beiden Cluster würden dann dem Gesicht und dem „Rest“ des Bildes entsprechen. Die Experimente haben schnell gezeigt, dass sich für die meisten Portrait-Fotos bessere Ergebnisse erzielen lassen, wenn $k = 3$ gewählt wird. In diesem Szenario entspricht wieder ein

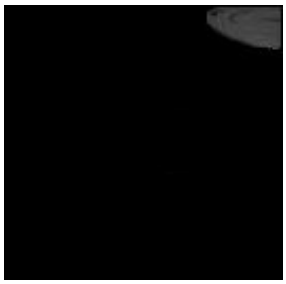
[5] `http://www.cis.upenn.edu/~jshi/software/` (Stand: 29.01.2015)

(a) Das vorverarbeitete, ursprüngliche Bild.



(b) Das erste Cluster (das Gesicht).

(c) Das zweite Cluster (der Torso).



(d) Das dritte Cluster (Fehler/Umgebung).

Abbildung 11: Mit Kernel-k-means++ segmentiertes Bild.

Cluster dem Gesicht, ein weiteres Cluster den irrelevanten Teilen des Bildes, und ein drittes Cluster wird für den Hals beziehungsweise den Torso verwendet.

Abbildung 11 zeigt den graphischen Plot des berechneten Normalized Cut Clusterings von Algorithmus Kernel-k-means++. Auf den drei Clustern ist gut zu erkennen, dass unsere Clusteraufteilung hier gut funktioniert hat. Es handelt sich um eine Beispielgrafik von der Graclus-Webseite[6].

Die Segmentierung einer einzelnen Grafik erfolgte nahezu unmittelbar. Um eine Vorstellung über die Leistungsfähigkeit des Algorithmus in der Bildsegmentierung zu ermöglichen, haben wir ihn auf die Extended Yale Face Database B (B+)[7] [GBK01] angewandt. Dabei handelt es sich um eine Sammlung von 16128 Portrait-Fotos von 28 Menschen in 9 unterschiedlichen Positionen und 64 verschiedenen Belichtungsszenarien. Der gesamte Datensatz ist etwa 2 GB groß. Der Datensatz konnte mit Kernel-k-means++ in rund 50 Minuten verarbeitet werden. Unter Einsatz von METIS und mit der spektralen Initialisierung dauerte die gesamte Verarbeitung rund 2 Stunden und 10 Minuten, also mehr als doppelt so lange. Die „Korrektheit“ der Gesichtserkennung konnte wegen der Anzahl der Bilder nicht vollständig ermittelt werden, da diese nur durch menschlichen Augenschein verifiziert werden kann.

[6] `http://www.cs.utexas.edu/users/dml/Software/graclus.html` (Stand: 29.01.2015)
[7] `http://vision.ucsd.edu/content/extended-yale-face-database-b-b` (Stand: 29.01.2015)

Es ist nötig, bei einer Segmentierung zu prüfen, ob ein Cluster dem Gesicht auf dem Foto entspricht. In einer Stichprobe von 160 Bildern wurden 153 Bilder korrekt segmentiert, das entspricht einem Anteil von 96%. Bei der spektralen Methode wurden 157 Bilder korrekt segmentiert, was einem Anteil von 98% entspricht.

4.4 Die Kernmengenkonstruktion

Unsere Variante der Kernmengenkonstruktion nennen wir in diesem Abschnitt verkürzt `Cs2`.

4.4.1 k-means mit der Kernmengenkonstruktion

Die empirische Evaluation der „Güte“ einer (starken) Kernmenge im Sinne ihrer Definition 2.1.2 beziehungsweise 3.3.1 ist im Allgemeinen schwierig, da sich schlecht für alle möglichen Zentrenmengen die jeweiligen Kosten vergleichen lassen. Neben der Messung der Laufzeiten geht man bezüglich der Qualitätsanalyse von Kernmengen wie beispielsweise in [AMR+12, FGS+13] vor. Man berechnet einerseits die Kernmenge S für die Eingabepunktmenge P, wendet auf S einen k-means-Algorithmus an und erhält eine Menge von k Zentren $C^1 = \{c_1^1, \ldots, c_k^1\}$, für die sich k-means-Kosten c^1 in Bezug auf die Eingabepunktmenge P ergeben. Andererseits wendet man denselben k-means-Algorithmus direkt auf ganz P an und erhält ebenfalls eine Zentrenmenge $C^2 = \{c_1^2, \ldots, c_k^2\}$, für die sich k-means-Kosten c^2 für P ergeben. Die k-means-Kosten c^1 und c^2 lassen sich vergleichen. Die untersuchte Qualität der Kernmenge bezieht sich bei diesem Vorgehen dann auf die Lösung des k-means-Problems mit Hilfe der Kernmenge.

Auf diese Weise wollen wir in diesem Abschnitt vorgehen. Dazu haben wir auf die Ausgaben der Kernmengen-Algorithmen, die wir jeweils zehnmal ausgeführt haben, den k-means-Algorithmus und den k-means++-Algorithmus je zehnmal angewandt und die durchschnittlichen Kosten für den Vergleich genommen. Insgesamt wurden also 100 Ausführungen inklusive anschließendem k-means und weitere 100 Ausführungen mit anschließendem k-means++ gemessen. Diese Methode der Evaluation ist zwar nicht allumfassend, wir können jedoch den gesamten Algorithmus evaluieren und insbesondere die Leistungsfähigkeit der Kernmengenkonstruktion zur Lösung des k-means-Problems bewerten.

Algorithmen. Für einen Vergleich ziehen wir zunächst die Kernmengenkonstruktionen BICO [FGS+13] und Stream-KM++ [AMR+12] heran. Beide Algorithmen sind state-of-the-art Datenstromalgorithmen zur Berechnung von Kernmengen.

Datensatz	Anzahl Punkte	Dimension d
Spambase	4601	57
NYSK	10421	7
Intrusion / KDD Cup 1999 [AMR+12]	311078	34
Covertype	581012	54

Tabelle 16: Eigenschaften der Datensätze aus [Lic13] für die Evaluation von `Cs2`.

Außerdem wollen wir auch hier wieder die Dlib-Implementierung von k-means++ zum Vergleich einsetzen, da wir in unserer Konstruktion intensiven Gebrauch von der D^2-Gewichtung machen.

Datensätze. Sowohl k-means++ als auch `Cs2` benötigen im Vergleich zu den Datenstromalgorithmen bereits auf Instanzen mit mehreren hunderttausend Punkten vergleichsweise lange Ausführungszeiten. Wir beschränken uns daher in diesem Teil der Evaluation auf mittelgroße Instanzen mit weniger als 10^6 Punkten und insbesondere überschaubarer Dimensionsanzahl (bis ca. 50 Dimensionen). Tabelle 16 zeigt die Eigenschaften der ausgewählten Datensätze aus dem UCI Machine Learning Repository.

Kernmengen-Größe. Eine asymptotische Größe von $\mathcal{O}(k)$ ist eine natürliche untere Schranke für die Größe einer Kernmenge, die wir erreichen wollen. Wir orientieren uns in unseren Experimenten an [AMR+12, FGS+13]. In beiden Papieren wurden in den experimentellen Auswertungen Kernmengen der Größe $200k$ konstruiert. Für die beiden Datensätze Spambase und NYSK würden die Kernmengen bei einer Größe von $200k$ mehr Punkte enthalten als die Ursprungsmenge. Bei diesen beiden Datensätzen haben wir daher Kernmengen der Größe $20k$ berechnet, um mehr Testwerte für einen umfassenderen Vergleich zu generieren, sofern die Wahl der Parameter dies zulässt (insbesondere $c \leq 20$). Für die Evaluation unserer Konstruktion ergibt sich die Herausforderung, die diversen Parameter so zu wählen, dass die berechnete Kernmenge in etwa diese Größe annimmt. Wir können dies auf mehreren Wegen erreichen. Es ist beispielsweise möglich, schon bei der initialen Partitionierung ein großes c zu wählen (im Extremfall sogar schon hier $c = 200$) und durch die maximale Tiefe oder den Kostenfaktor f dafür zu sorgen, dass nur bis zu einer geringen Tiefe partitioniert wird (oder im Extremfall gar nicht weiter partitioniert wird). Alternativ können wir auch ein kleines c wählen und dafür größere Werte von d wählen, sodass wir initial nur wenige Teilmengen haben und in der rekursiven Partitionierung eine größere Anzahl von Teilmengen erzielen.

Clusteranzahl. Wir wollen auch hier mit verschiedenen Werten für die Clusteranzahl k arbeiten. Dabei orientieren wir uns an den Experimenten aus [AMR+12, FGS+13]. Wir führen die Algorithmen jeweils mit $k \in \{10, 30, 50\}$ aus.

Wir geben bei den Ergebnistabellen immer den durchschnittlichen Wert für Kosten und Laufzeit von zehn Ausführungen des k-means-Algorithmus beziehungsweise des k-means++-Algorithmus für je zehn Ausführungen der Kernmengen-Algorithmen an. Bei den Kernmengen-Algorithmen beziehen sich diese Werte auf die Ausführung von k-means beziehungsweise k-means++ unter Eingabe der Kernmenge und bei k-means++ selbst auf die 100 Ausführungen auf der gesamten Punktmenge P. Für `Cs2` geben wir die jeweiligen Parameter im Namen des Algorithmus an. Beispielsweise schreiben wir für $c = 200$, $d = /$, $f = /$ und $T = 1$: `Cs2`-c-200-d-/-f-/-T-1. Die Wahl von d und f ist hier irrelevant, da wegen T nur initial partitioniert wird. Dies deuten wir mit der Notation „/“ an.

Wir betrachten zunächst die Ergebnisse der Kernmengenkonstruktionen mit anschließender Ausführung des k-means-Algorithmus im Vergleich zu k-means++.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	$8.44 \cdot 10^7$	$5.93 \cdot 10^8$	$1.58 \cdot 10^{13}$	$3.77 \cdot 10^{11}$
	30	$1.33 \cdot 10^7$	$4.76 \cdot 10^8$	$4.63 \cdot 10^{11}$	$1.72 \cdot 10^{11}$
	50	$6.57 \cdot 10^6$	$6.63 \cdot 10^8$	$1.22 \cdot 10^{11}$	$1.29 \cdot 10^{11}$
BICO	10	$7.21 \cdot 10^7$	$5.71 \cdot 10^8$	$1.37 \cdot 10^{13}$	$3.56 \cdot 10^{11}$
	30	$1.29 \cdot 10^7$	$4.45 \cdot 10^8$	$4.48 \cdot 10^{11}$	$1.59 \cdot 10^{11}$
	50	$6.33 \cdot 10^6$	$6.32 \cdot 10^8$	$1.18 \cdot 10^{11}$	$1.15 \cdot 10^{11}$
k-means++	10	$8.71 \cdot 10^7$	$6.09 \cdot 10^8$	$1.75 \cdot 10^{13}$	$3.42 \cdot 10^{11}$
	30	$1.34 \cdot 10^7$	$4.98 \cdot 10^8$	$4.96 \cdot 10^{11}$	$1.54 \cdot 10^{11}$
	50	$6.68 \cdot 10^6$	$7.01 \cdot 10^8$	$1.29 \cdot 10^{11}$	$1.13 \cdot 10^{11}$
Cs2-c-200-d-/-f-/-T-1	10	$8.95 \cdot 10^7$	$6.42 \cdot 10^8$	$1.82 \cdot 10^{13}$	$3.55 \cdot 10^{11}$
	30	-	$5.33 \cdot 10^8$	$4.95 \cdot 10^{11}$	$1.70 \cdot 10^{11}$
	50	-	$7.67 \cdot 10^8$	$1.32 \cdot 10^{11}$	$1.18 \cdot 10^{11}$
Cs2-c-1-d-50-f-1.3-T-3	10	$8.61 \cdot 10^7$	$6.17 \cdot 10^8$	$1.82 \cdot 10^{13}$	$3.52 \cdot 10^{11}$
	30	$1.42 \cdot 10^7$	$5.06 \cdot 10^8$	$4.89 \cdot 10^{11}$	$1.63 \cdot 10^{11}$
	50	$6.71 \cdot 10^6$	$7.02 \cdot 10^8$	$1.31 \cdot 10^{11}$	$1.14 \cdot 10^{11}$
Cs2-c-5-d-25-f-1.2-T-15	10	$8.85 \cdot 10^7$	$6.32 \cdot 10^8$	$1.98 \cdot 10^{13}$	$3.62 \cdot 10^{11}$
	30	$1.49 \cdot 10^7$	$5.16 \cdot 10^8$	$5.11 \cdot 10^{11}$	$1.64 \cdot 10^{11}$
	50	$6.91 \cdot 10^6$	$7.18 \cdot 10^8$	$1.41 \cdot 10^{11}$	$1.19 \cdot 10^{11}$
Cs2-c-50-d-10-f-1.4-T-10	10	$8.60 \cdot 10^7$	$5.99 \cdot 10^8$	$1.70 \cdot 10^{13}$	$3.51 \cdot 10^{11}$
	30	$1.36 \cdot 10^7$	$5.02 \cdot 10^8$	$4.93 \cdot 10^{11}$	$1.61 \cdot 10^{11}$
	50	$6.74 \cdot 10^6$	$7.04 \cdot 10^8$	$1.32 \cdot 10^{11}$	$1.17 \cdot 10^{11}$
Cs2-c-100-d-5-f-1.1-T-4	10	$8.93 \cdot 10^7$	$6.40 \cdot 10^8$	$1.80 \cdot 10^{13}$	$3.51 \cdot 10^{11}$
	30	$1.48 \cdot 10^7$	$5.26 \cdot 10^8$	$5.01 \cdot 10^{11}$	$1.64 \cdot 10^{11}$
	50	-	$7.49 \cdot 10^8$	$1.31 \cdot 10^{11}$	$1.18 \cdot 10^{11}$

Tabelle 17: Durchschnittliche Kosten der Algorithmen für $k \in \{10, 30, 50\}$ (je kleiner, desto besser). Bei den Kernmengen-Algorithmen wurde anschließend der k-means-Algorithmus ausgeführt.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	3.2	4.1	71.6	241.8
	30	16.1	20.2	98.1	381.7
	50	44.1	67.3	230.3	529.0
BICO	10	0.1	1.5	15.2	25.2
	30	0.1	2.0	17.1	45.7
	50	0.1	2.4	20.5	42.3
k-means++	10	1.2	2.9	23.8	181.2
	30	5.7	10.2	897.3	2024.7
	50	13.3	28.3	644.0	6922.9
Cs2-c-200-d-/-f-/-T-1	10	3.1	7.2	48.4	381.2
	30	-	21.1	1574.1	4003.3
	50	-	58.6	1309.2	12372.3
Cs2-c-1-d-50-f-1.3-T-3	10	2.0	5.8	40.9	321.9
	30	11.4	19.7	1703.3	3321.6
	50	25.7	55.4	1122.1	11242.1
Cs2-c-5-d-25-f-1.2-T-15	10	1.9	5.1	40.3	299.7
	30	10.7	15.7	1603.8	2812.7
	50	24.3	44.2	1078.1	9659.8
Cs2-c-50-d-10-f-1.4-T-10	10	1.6	3.0	35.3	239.0
	30	8.2	13.5	1237.9	2044.4
	50	17.8	38.2	851.2	7831.1
Cs2-c-100-d-5-f-1.1-T-4	10	2.1	17.2	41.1	331.0
	30	9.8	16.9	1787.7	3912.7
	50	-	24.7	1129.5	10265.0

Tabelle 18: Durchschnittliche Laufzeiten der Algorithmen für $k \in \{10, 30, 50\}$ in Sekunden. Bei den Kernmengen-Algorithmen wurde anschließend der k-means-Algorithmus ausgeführt.

Der Kostenübersicht in Tabelle 17 können wir entnehmen, dass die Datenstromalgorithmen mit Ausnahme von Covertype geringere Kosten erzielen als k-means++. Auf Covertype sind die Kosten jedoch nur geringfügig größer. Im Schnitt erzielt Stream-KM++ etwa 2-3% geringere Kosten, während BICO bis zu 5% geringere Kosten erzielt. Cs2 hingegen erreicht in drei Parametrisierungen rund 3% höhere Kosten. Die besten Werte werden bei $c = 1, d = 50$ und $c = 50, d = 10$ erzielt. Bei letzteren ist Cs2 etwa 2% besser als k-means++ aber noch leicht über den Werten von Stream-KM++. Bei den Laufzeiten in Tabelle 18 ist die Ordnung der Algorithmen gleich. Wir können jedoch insbesondere bei den Datenstromalgorithmen einen

massiven Unterschied von einigen Größenordnungen in der Laufzeit ausmachen. Die Datenstromalgorithmen sind deutlich schneller als k-means++ und die Kernmengenkonstruktion. `Cs2` erzielt hier die geringsten Laufzeiten bei $c = 50, d = 10$, ist jedoch im Schnitt immer noch ca. 10-15% langsamer als k-means++.

Wir betrachten als nächstes die Ergebnisse der Kernmengen-Algorithmen bei anschließender Ausführung von k-means++.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	$7.85 \cdot 10^7$	$5.67 \cdot 10^8$	$1.27 \cdot 10^{13}$	$3.43 \cdot 10^{11}$
	30	$1.24 \cdot 10^7$	$4.22 \cdot 10^8$	$4.29 \cdot 10^{11}$	$1.57 \cdot 10^{11}$
	50	$6.29 \cdot 10^6$	$6.12 \cdot 10^8$	$1.11 \cdot 10^{11}$	$1.15 \cdot 10^{11}$
BICO	10	$7.81 \cdot 10^7$	$5.52 \cdot 10^8$	$1.19 \cdot 10^{13}$	$3.22 \cdot 10^{11}$
	30	$1.22 \cdot 10^7$	$4.13 \cdot 10^8$	$4.25 \cdot 10^{11}$	$1.48 \cdot 10^{11}$
	50	$6.15 \cdot 10^6$	$5.98 \cdot 10^8$	$1.03 \cdot 10^{11}$	$1.02 \cdot 10^{11}$
k-means++	10	$8.71 \cdot 10^7$	$6.09 \cdot 10^8$	$1.75 \cdot 10^{13}$	$3.42 \cdot 10^{11}$
	30	$1.34 \cdot 10^7$	$4.98 \cdot 10^8$	$4.96 \cdot 10^{11}$	$1.54 \cdot 10^{11}$
	50	$6.68 \cdot 10^6$	$7.01 \cdot 10^8$	$1.29 \cdot 10^{11}$	$1.13 \cdot 10^{11}$
`Cs2`-c-200-d-/-f-/-T-1	10	$8.71 \cdot 10^7$	$6.07 \cdot 10^8$	$1.63 \cdot 10^{13}$	$3.35 \cdot 10^{11}$
	30	-	$4.96 \cdot 10^8$	$4.82 \cdot 10^{11}$	$1.51 \cdot 10^{11}$
	50	-	$7.00 \cdot 10^8$	$1.23 \cdot 10^{11}$	$1.09 \cdot 10^{11}$
`Cs2`-c-1-d-50-f-1.3-T-3	10	$8.33 \cdot 10^7$	$5.92 \cdot 10^8$	$1.59 \cdot 10^{13}$	$3.31 \cdot 10^{11}$
	30	$1.30 \cdot 10^7$	$4.88 \cdot 10^8$	$4.75 \cdot 10^{11}$	$1.48 \cdot 10^{11}$
	50	$6.59 \cdot 10^6$	$6.89 \cdot 10^8$	$1.19 \cdot 10^{11}$	$1.06 \cdot 10^{11}$
`Cs2`-c-5-d-25-f-1.2-T-15	10	$8.63 \cdot 10^7$	$6.00 \cdot 10^8$	$1.71 \cdot 10^{13}$	$3.40 \cdot 10^{11}$
	30	$1.33 \cdot 10^7$	$4.94 \cdot 10^8$	$4.89 \cdot 10^{11}$	$1.51 \cdot 10^{11}$
	50	$6.65 \cdot 10^6$	$6.97 \cdot 10^8$	$1.25 \cdot 10^{11}$	$1.10 \cdot 10^{11}$
`Cs2`-c-50-d-10-f-1.4-T-10	10	$8.22 \cdot 10^7$	$5.85 \cdot 10^8$	$1.52 \cdot 10^{13}$	$3.30 \cdot 10^{11}$
	30	$1.28 \cdot 10^7$	$4.80 \cdot 10^8$	$4.70 \cdot 10^{11}$	$1.47 \cdot 10^{11}$
	50	$6.49 \cdot 10^6$	$6.81 \cdot 10^8$	$1.17 \cdot 10^{11}$	$1.09 \cdot 10^{11}$
`Cs2`-c-100-d-5-f-1.1-T-4	10	$8.70 \cdot 10^7$	$6.04 \cdot 10^8$	$1.73 \cdot 10^{13}$	$3.42 \cdot 10^{11}$
	30	$1.32 \cdot 10^7$	$4.95 \cdot 10^8$	$4.92 \cdot 10^{11}$	$1.54 \cdot 10^{11}$
	50	-	$6.97 \cdot 10^8$	$1.28 \cdot 10^{11}$	$1.12 \cdot 10^{11}$

Tabelle 19: Durchschnittliche Kosten der Algorithmen für $k \in \{10, 30, 50\}$ (je kleiner, desto besser). Bei den Kernmengen-Algorithmen wurde anschließend der k-means++-Algorithmus ausgeführt.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	1.4	2.7	28.7	115.2
	30	6.3	9.3	45.3	170.5
	50	18.2	25.2	100.1	290.2
BICO	10	0.1	0.8	7.6	13.7
	30	0.1	0.9	9.8	16.3
	50	0.1	1.0	12.1	19.2
k-means++	10	1.2	2.9	23.8	181.2
	30	5.7	10.2	897.3	2024.7
	50	13.3	28.3	644.0	6922.9
Cs2-c-200-d-/-f-/-T-1	10	1.2	2.9	22.6	175.3
	30	-	10.0	854.2	1889.1
	50	-	27.8	608.1	6122.3
Cs2-c-1-d-50-f-1.3-T-3	10	1.1	2.5	21.1	155.3
	30	5.5	9.1	802.3	1772.4
	50	12.9	26.5	577.2	5532.1
Cs2-c-5-d-25-f-1.2-T-15	10	1.0	2.3	19.9	149.2
	30	5.2	8.3	763.7	1566.3
	50	11.8	23.2	522.3	5138.4
Cs2-c-50-d-10-f-1.4-T-10	10	0.9	1.8	18.1	121.3
	30	4.3	7.6	653.2	1132.5
	50	9.9	20.4	434.1	4237.2
Cs2-c-100-d-5-f-1.1-T-4	10	1.0	8.5	20.6	162.4
	30	5.1	8.0	788.1	1802.4
	50	-	24.7	554.3	5322.1

Tabelle 20: Durchschnittliche Laufzeiten der Algorithmen für $k \in \{10, 30, 50\}$ in Sekunden. Bei den Kernmengen-Algorithmen wurde anschließend der k-means++-Algorithmus ausgeführt.

Mit der anschließenden Ausführung von k-means++ erzielen sämtliche Kernmengen-Algorithmen bezüglich k-means-Kosten und Laufzeiten bessere Ergebnisse. Die Verbesserung reicht bei den Kosten von 5-10% und bei den Laufzeiten bis zu einer Halbierung der Laufzeiten. Tabelle 19 ist zu entnehmen, dass unsere Konstruktion mit allen Parametrisierungen bessere k-means-Kosten erzielt als k-means++. In der Spitze sind Verbesserungen von bis zu 10% zu verzeichnen. Andererseits erzielen die Datenstromalgorithmen Stream-KM++und BICO durchweg noch geringere Kosten (in der Spitze bis rund 20% weniger im Vergleich zu k-means++). Bei den Laufzeiten in Tabelle 20 ist erneut deutlich zu erkennen, dass unsere Kernmengenkonstruktion,

die intensiven Gebrauch der D^2-Gewichtung macht, um einige Größenordnungen langsamer ist als die state-of-the-art Datenstromalgorithmen. Die Läufe mit den geringsten Kosten und Laufzeiten sind jeweils bei $c = 50$ und $d = 10$ und verbessern die Laufzeit von k-means++ um bis zu 40%. Dies ist eine beachtliche Verbesserung. Die dementsprechende Wahl der Parameter scheint empirisch sowohl für die Kosten als auch für die Laufzeit am besten geeignet zu sein.

Es ist abzusehen, dass wir mit unserer Konstruktion keine Laufzeiten in der Größenordnung der Datenstromalgorithmen erzielen können. Wir wollen stattdessen versuchen, auch für die Kernmengenkonstruktion die Kernel Methode für alle Distanzberechnungen anzuwenden, um die erzielten Kosten zu optimieren. Dazu setzen wir wieder einen Gauss-/RBF-Kernel mit $\sigma = 0.1$ ein. Für die Kernelvariante von `Cs2` schreiben wir `KCs2`. Wir beginnen wieder damit, im Anschluss an die Kernmengen-Algorithmen den k-means-Algorithmus auszuführen.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	$8.44 \cdot 10^7$	$5.93 \cdot 10^8$	$1.58 \cdot 10^{13}$	$3.77 \cdot 10^{11}$
	30	$1.33 \cdot 10^7$	$4.76 \cdot 10^8$	$4.63 \cdot 10^{11}$	$1.72 \cdot 10^{11}$
	50	$6.57 \cdot 10^6$	$6.63 \cdot 10^8$	$1.22 \cdot 10^{11}$	$1.29 \cdot 10^{11}$
BICO	10	$7.21 \cdot 10^7$	$5.71 \cdot 10^8$	$1.37 \cdot 10^{13}$	$3.56 \cdot 10^{11}$
	30	$1.29 \cdot 10^7$	$4.45 \cdot 10^8$	$4.48 \cdot 10^{11}$	$1.59 \cdot 10^{11}$
	50	$6.33 \cdot 10^6$	$6.32 \cdot 10^8$	$1.18 \cdot 10^{11}$	$1.15 \cdot 10^{11}$
k-means++	10	$8.71 \cdot 10^7$	$6.09 \cdot 10^8$	$1.75 \cdot 10^{13}$	$3.42 \cdot 10^{11}$
	30	$1.34 \cdot 10^7$	$4.98 \cdot 10^8$	$4.96 \cdot 10^{11}$	$1.54 \cdot 10^{11}$
	50	$6.68 \cdot 10^6$	$7.01 \cdot 10^8$	$1.29 \cdot 10^{11}$	$1.13 \cdot 10^{11}$
KCs2-c-200-d-/-f-/-T-1	10	$8.67 \cdot 10^7$	$6.02 \cdot 10^8$	$1.61 \cdot 10^{13}$	$3.35 \cdot 10^{11}$
	30	-	$4.89 \cdot 10^8$	$4.84 \cdot 10^{11}$	$1.56 \cdot 10^{11}$
	50	-	$6.93 \cdot 10^8$	$1.22 \cdot 10^{11}$	$1.10 \cdot 10^{11}$
KCs2-c-1-d-50-f-1.3-T-3	10	$8.33 \cdot 10^7$	$5.87 \cdot 10^8$	$1.53 \cdot 10^{13}$	$3.27 \cdot 10^{11}$
	30	$1.29 \cdot 10^7$	$4.81 \cdot 10^8$	$4.72 \cdot 10^{11}$	$1.45 \cdot 10^{11}$
	50	$6.55 \cdot 10^6$	$6.79 \cdot 10^8$	$1.18 \cdot 10^{11}$	$1.04 \cdot 10^{11}$
KCs2-c-5-d-25-f-1.2-T-15	10	$8.22 \cdot 10^7$	$5.79 \cdot 10^8$	$1.39 \cdot 10^{13}$	$3.41 \cdot 10^{11}$
	30	$1.30 \cdot 10^7$	$4.39 \cdot 10^8$	$4.49 \cdot 10^{11}$	$1.56 \cdot 10^{11}$
	50	$6.51 \cdot 10^6$	$6.32 \cdot 10^8$	$1.21 \cdot 10^{11}$	$1.09 \cdot 10^{11}$
KCs2-c-50-d-10-f-1.4-T-10	10	$7.91 \cdot 10^7$	$5.67 \cdot 10^8$	$1.24 \cdot 10^{13}$	$3.31 \cdot 10^{11}$
	30	$1.25 \cdot 10^7$	$4.24 \cdot 10^8$	$4.33 \cdot 10^{11}$	$1.50 \cdot 10^{11}$
	50	$6.34 \cdot 10^6$	$6.09 \cdot 10^8$	$1.09 \cdot 10^{11}$	$1.05 \cdot 10^{11}$
KCs2-c-100-d-5-f-1.1-T-4	10	$8.60 \cdot 10^7$	$6.05 \cdot 10^8$	$1.76 \cdot 10^{13}$	$3.41 \cdot 10^{11}$
	30	$1.33 \cdot 10^7$	$4.99 \cdot 10^8$	$4.95 \cdot 10^{11}$	$1.58 \cdot 10^{11}$
	50	-	$6.99 \cdot 10^8$	$1.27 \cdot 10^{11}$	$1.12 \cdot 10^{11}$

Tabelle 21: Durchschnittliche Kosten der Algorithmen für $k \in \{10, 30, 50\}$ (je kleiner, desto besser). Bei den Kernmengen-Algorithmen wurde anschließend der k-means-Algorithmus ausgeführt.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	3.2	4.1	71.6	241.8
	30	16.1	20.2	98.1	381.7
	50	44.1	67.3	230.3	529.0
BICO	10	0.1	1.5	15.2	25.2
	30	0.1	2.0	17.1	45.7
	50	0.1	2.4	20.5	42.3
k-means++	10	1.2	2.9	23.8	181.2
	30	5.7	10.2	897.3	2024.7
	50	13.3	28.3	644.0	6922.9
KCs2-c-200-d-/-f-/-T-1	10	3.5	7.8	51.2	397.2
	30	-	22.7	1664.7	4323.4
	50	-	61.2	1402.1	13098.9
KCs2-c-1-d-50-f-1.3-T-3	10	2.1	6.2	42.8	344.0
	30	12.0	20.6	1812.7	3517.7
	50	26.8	59.1	1202.5	12021.9
KCs2-c-5-d-25-f-1.2-T-15	10	2.5	5.4	43.5	317.8
	30	11.4	16.5	1712.9	2996.1
	50	25.8	47.3	1155.5	10371.2
KCs2-c-50-d-10-f-1.4-T-10	10	1.8	3.2	37.5	250.7
	30	8.7	14.4	1402.5	2185.7
	50	18.9	40.8	906.6	8418.5
KCs2-c-100-d-5-f-1.1-T-4	10	2.1	17.2	41.1	352.8
	30	9.8	16.9	1787.7	4169.0
	50	-	24.7	1129.5	1978.1

Tabelle 22: Durchschnittliche Laufzeiten der Algorithmen für $k \in \{10, 30, 50\}$ in Sekunden. Bei den Kernmengen-Algorithmen wurde anschließend der k-means-Algorithmus ausgeführt.

Wir können Tabelle 21 entnehmen, dass die Kernel-Variante unserer Kernmengenkonstruktion in den Experimenten mit anschließendem k-means zwischen 2 und 4% geringere Kosten erbringt. Bei den Laufzeiten in Tabelle 22 ist hingegen ein Zuwachs von rund 8% zu verzeichnen. Ausgehend von den vorherigen Experimenten ist anzunehmen, dass wir durch anschließende Ausführung von k-means++ eine weitere Senkung der Kosten und Laufzeiten erreichen sollten. Die Ergebnisse der entsprechenden Experimente zeigen wir im Folgenden.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	$7.85 \cdot 10^7$	$5.67 \cdot 10^8$	$1.27 \cdot 10^{13}$	$3.43 \cdot 10^{11}$
	30	$1.24 \cdot 10^7$	$4.22 \cdot 10^8$	$4.29 \cdot 10^{11}$	$1.57 \cdot 10^{11}$
	50	$6.29 \cdot 10^6$	$6.12 \cdot 10^8$	$1.11 \cdot 10^{11}$	$1.15 \cdot 10^{11}$
BICO	10	$7.81 \cdot 10^7$	$5.52 \cdot 10^8$	$1.19 \cdot 10^{13}$	$3.22 \cdot 10^{11}$
	30	$1.22 \cdot 10^7$	$4.13 \cdot 10^8$	$4.25 \cdot 10^{11}$	$1.48 \cdot 10^{11}$
	50	$6.15 \cdot 10^6$	$5.98 \cdot 10^8$	$1.03 \cdot 10^{11}$	$1.02 \cdot 10^{11}$
k-means++	10	$8.71 \cdot 10^7$	$6.09 \cdot 10^8$	$1.75 \cdot 10^{13}$	$3.42 \cdot 10^{11}$
	30	$1.34 \cdot 10^7$	$4.98 \cdot 10^8$	$4.96 \cdot 10^{11}$	$1.54 \cdot 10^{11}$
	50	$6.68 \cdot 10^6$	$7.01 \cdot 10^8$	$1.29 \cdot 10^{11}$	$1.13 \cdot 10^{11}$
KCs2-c-200-d-/-f-/-T-1	10	$8.63 \cdot 10^7$	$5.94 \cdot 10^8$	$1.58 \cdot 10^{13}$	$3.29 \cdot 10^{11}$
	30	-	$4.82 \cdot 10^8$	$4.77 \cdot 10^{11}$	$1.48 \cdot 10^{11}$
	50	-	$6.85 \cdot 10^8$	$1.20 \cdot 10^{11}$	$1.07 \cdot 10^{11}$
KCs2-c-1-d-50-f-1.3-T-3	10	$8.21 \cdot 10^7$	$5.83 \cdot 10^8$	$1.50 \cdot 10^{13}$	$3.25 \cdot 10^{11}$
	30	$1.28 \cdot 10^7$	$4.76 \cdot 10^8$	$4.66 \cdot 10^{11}$	$1.42 \cdot 10^{11}$
	50	$6.51 \cdot 10^6$	$6.72 \cdot 10^8$	$1.17 \cdot 10^{11}$	$1.03 \cdot 10^{11}$
KCs2-c-5-d-25-f-1.2-T-15	10	$8.14 \cdot 10^7$	$5.71 \cdot 10^8$	$1.35 \cdot 10^{13}$	$3.33 \cdot 10^{11}$
	30	$1.29 \cdot 10^7$	$4.32 \cdot 10^8$	$4.42 \cdot 10^{11}$	$1.49 \cdot 10^{11}$
	50	$6.44 \cdot 10^6$	$6.25 \cdot 10^8$	$1.19 \cdot 10^{11}$	$1.07 \cdot 10^{11}$
KCs2-c-50-d-10-f-1.4-T-10	10	$7.83 \cdot 10^7$	$5.58 \cdot 10^8$	$1.22 \cdot 10^{13}$	$3.25 \cdot 10^{11}$
	30	$1.23 \cdot 10^7$	$4.19 \cdot 10^8$	$4.27 \cdot 10^{11}$	$1.44 \cdot 10^{11}$
	50	$6.27 \cdot 10^6$	$5.99 \cdot 10^8$	$1.07 \cdot 10^{11}$	$1.03 \cdot 10^{11}$
KCs2-c-100-d-5-f-1.1-T-4	10	$8.52 \cdot 10^7$	$5.97 \cdot 10^8$	$1.68 \cdot 10^{13}$	$3.36 \cdot 10^{11}$
	30	$1.30 \cdot 10^7$	$4.91 \cdot 10^8$	$4.88 \cdot 10^{11}$	$1.51 \cdot 10^{11}$
	50	-	$6.90 \cdot 10^8$	$1.24 \cdot 10^{11}$	$1.10 \cdot 10^{11}$

Tabelle 23: Durchschnittliche Kosten der Algorithmen für $k \in \{10, 30, 50\}$ (je kleiner, desto besser). Bei den Kernmengen-Algorithmen wurde anschließend der k-means++-means-Algorithmus ausgeführt.

Algorithmus	k	**Datensatz**			
		Spambase	NYSK	Intrusion	Covertype
Stream-KM++	10	1.4	2.7	28.7	115.2
	30	6.3	9.3	45.3	170.5
	50	18.2	25.2	100.1	290.2
BICO	10	0.1	0.8	7.6	13.7
	30	0.1	0.9	9.8	16.3
	50	0.1	1.0	12.1	19.2
k-means++	10	1.2	2.9	23.8	181.2
	30	5.7	10.2	897.3	2024.7
	50	13.3	28.3	644.0	6922.9
KCs2-c-200-d-/-f-/-T-1	10	1.4	3.3	24.7	217.5
	30	-	11.1	922.1	2007.0
	50	-	29.6	673.3	6536.7
KCs2-c-1-d-50-f-1.3-T-3	10	1.3	2.8	23.0	171.1
	30	6.1	9.9	880.8	1911.4
	50	13.8	28.1	607.9	6072.1
KCs2-c-5-d-25-f-1.2-T-15	10	1.1	2.6	21.7	164.3
	30	5.8	8.9	816.9	1721.7
	50	12.9	25.0	578.2	5649.1
KCs2-c-50-d-10-f-1.4-T-10	10	1.0	2.0	19.9	135.8
	30	4.6	8.2	702.1	1281.9
	50	10.7	22.2	478.5	4649.2
KCs2-c-100-d-5-f-1.1-T-4	10	1.1	9.2	22.9	180.1
	30	5.5	8.9	872.2	1993.5
	50	-	26.9	612.1	5911.8

Tabelle 24: Durchschnittliche Laufzeiten der Algorithmen für $k \in \{10, 30, 50\}$ in Sekunden. Bei den Kernmengen-Algorithmen wurde anschließend der k-means++-means-Algorithmus ausgeführt.

Wie erwartet erbringt KCs2 die besten Ergebnisse bezüglich k-means-Kosten und Laufzeit, wenn auf die berechnete Kernmenge der k-means++-Algorithmus angewandt wird. KCs2 erzielt mit $c = 50$ und $d = 10$ durchweg Ergebnisse, die um einige Prozentpunkte besser sind als die für k-means++ gemessenen Werte. In der Spitze ist KCs2 bei den k-means-Kosten in etwa gleichauf mit Stream-KM++ und wenige Prozentpunkte teurer als BICO. Bei den Laufzeiten ist KCs2 bis zu 10% schneller als k-means++. Die Datenstromalgorithmen sind jedoch über alle Konfigurationen hinweg durchgehend um einige Größenordnungen schneller.

4.4.2 Graphpartitionierung mit der Kernmengenkonstruktion

Es ist naheliegend, dass wir unter Einsatz von `KCs2` noch einmal eine Leistungssteigerung in der Graphpartitionierung erreichen können, da `KCs2` bessere k-means-Lösungen produziert als k-means++. Wir wollen daher auch für `KCs2` die Experimente zur Graphpartitionierung durchführen. Dazu verwenden wir wieder die Datensätze in Tabelle 7. Wie zuvor werden jeweils 128 und 512 Partitionierungen berechnet. Dadurch ist es nötig, die Parameter von `KCs2` leicht zu modifizieren, damit wir wieder eine Kernmenge der Größe $200k$ erhalten. Für $k = 128$ wurde `KCs2` mit $c = 45, d = 20, f = 1.5, T = 10$ parametrisiert. Für $k = 512$ wurde $c = 50, d = 30, f = 1.3, T = 12$ gewählt. Bei den Experimenten haben wir wiederum 100 Durchläufe pro Algorithmus durchgeführt und geben in den Ergebnistabellen jeweils den durchschnittlichen Wert an. Wir betrachten zunächst die Werte und Laufzeiten für die Ratio Association.

Bezüglich der Zielfunktionswerte für die Ratio Association erreichen wir mit `KCs2` in etwa die Qualität des spektralen Algorithmus. Auf einigen Datensätzen erzielt `KCs2` geringfügig bessere Werte, auf AUTO und BCSSTK31 hingegen leicht schlechtere Werte. Insgesamt ist `KCs2` geringfügig schlechter als der spektrale Algorithmus. Dafür ist die Kernmengenkonstruktion mindestens gleichauf mit METIS und verbessert Kernel-k-means++ insgesamt um durchschnittlich 5%. Bei den Laufzeiten in den Tabellen 27 und 28 ist eine zusätzliche Beschleunigung von rund 10% gegenüber Kernel-k-means++ zu erkennen. Damit erhalten wir einen Speedup im Vergleich zum spektralen Algorithmus von mehr als zwei. METIS ist im Schnitt etwas weniger als einen Faktor Zwei langsamer. Bei $k = 512$ ist der Speedup gegenüber dem spektralen Algorithmus wie schon zuvor bei Kernel-k-means++ auf einigen Datensätzen sogar noch größer.

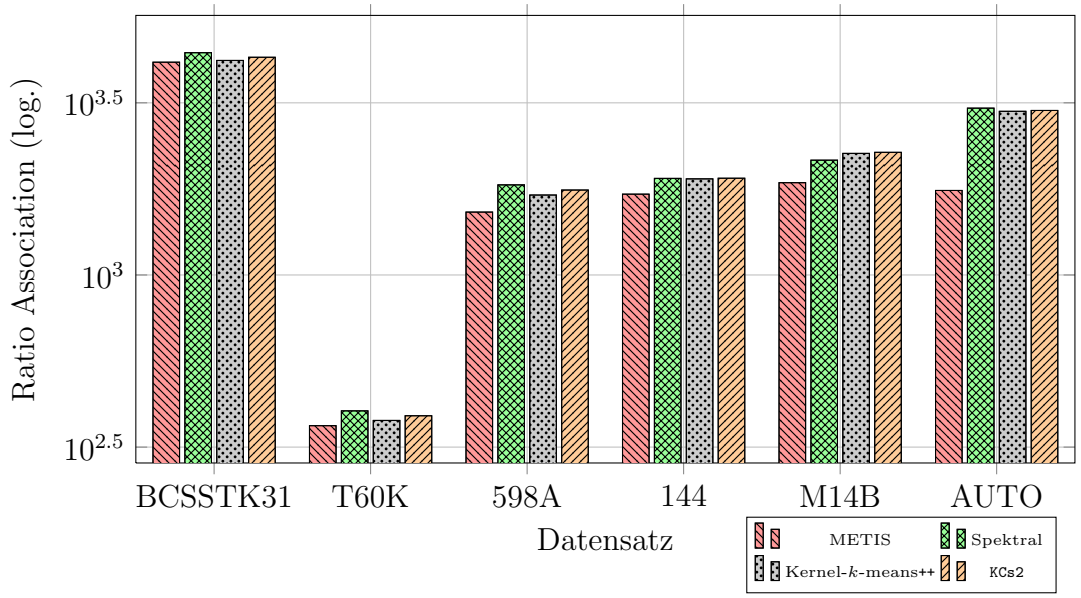


Abbildung 12: Durchschnittliche Ratio Association Werte der Algorithmen für $k = 128$ (je größer, desto besser).

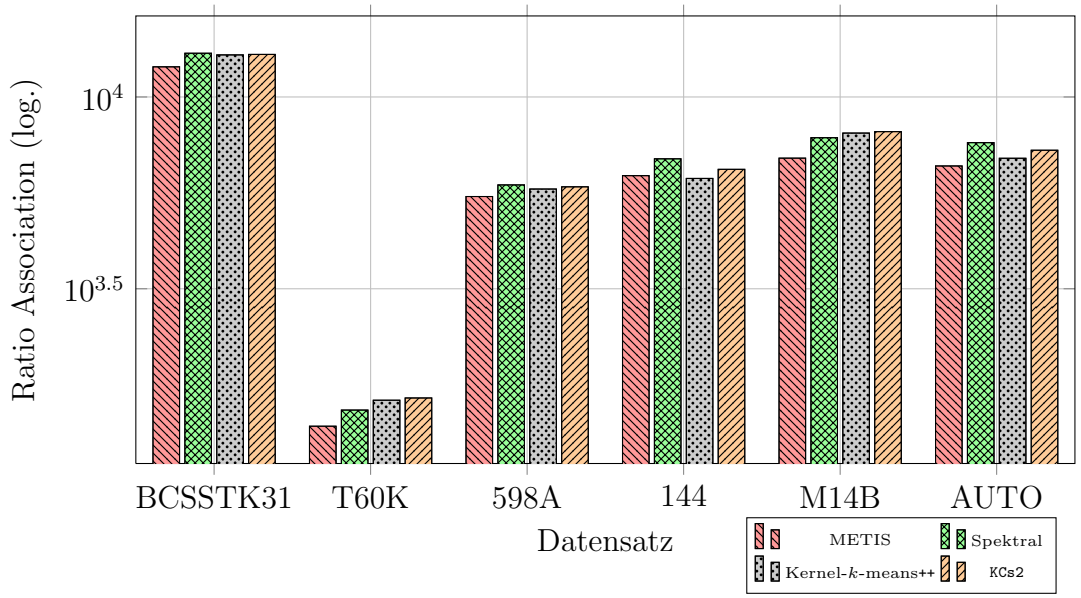


Abbildung 13: Durchschnittliche Ratio Association Werte der Algorithmen für $k = 512$ (je größer, desto besser).

Datensatz	**Zufall**	**METIS**	**Spektral**	**Kernel-k-means++**	`KCs2`
BCSSTK31	3265	4150	4422	4199	4289
T60K	263	365	403	378	390
598A	1144	1523	1827	1707	1765
144	1223	1717	1908	1902	1910
M14B	1022	1854	2155	2255	2271
AUTO	1197	1760	3052	2988	3005

Tabelle 25: Durchschnittliche Ratio Association Werte der Algorithmen für $k = 128$ (je größer, desto besser).

Datensatz	**Zufall**	**METIS**	**Spektral**	**Kernel-k-means++**	`KCs2`
BCSSTK31	8522	11984	13002	12873	12910
T60K	875	1386	1527	1620	1642
598A	4003	5502	5899	5755	5832
144	4833	6236	6899	6133	6477
M14B	4687	6927	7832	8052	8122
AUTO	5498	6610	7603	6922	7264

Tabelle 26: Durchschnittliche Ratio Association Werte der Algorithmen für $k = 512$ (je größer, desto besser).

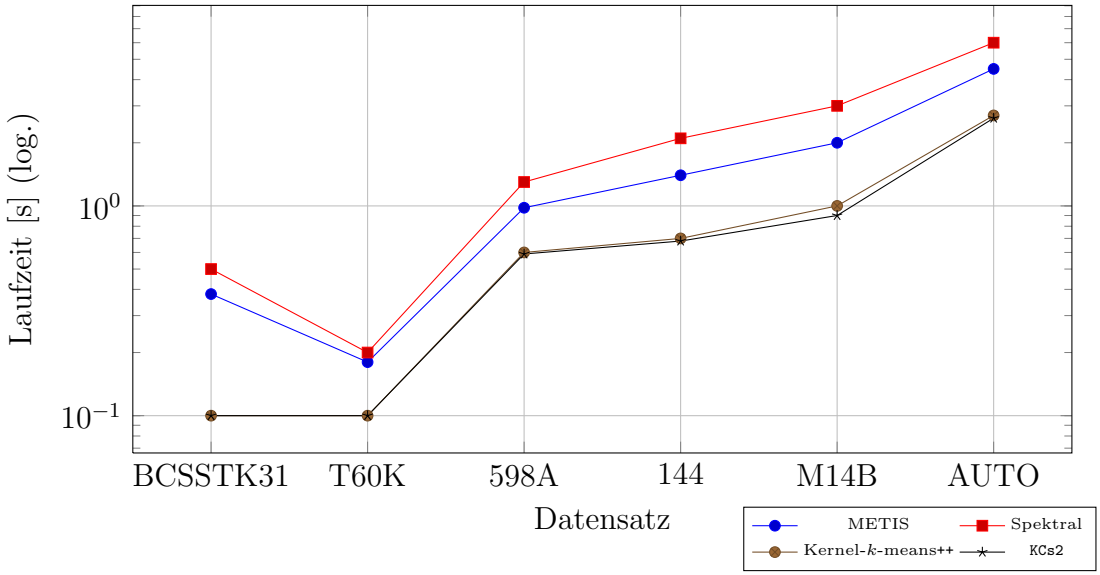


Abbildung 14: Durchschnittliche Laufzeiten der Algorithmen für Ratio Association mit $k = 128$.

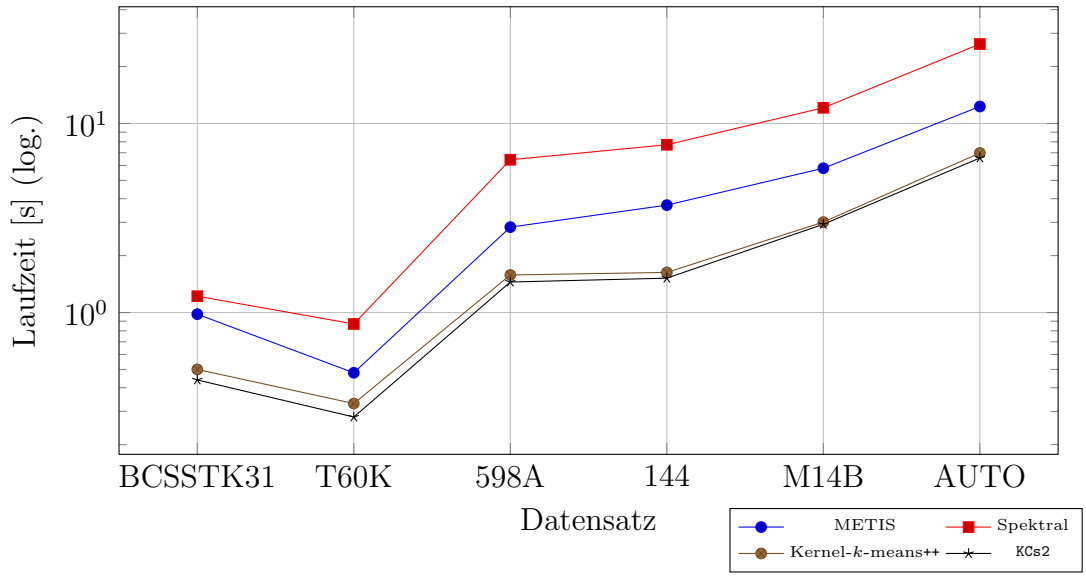


Abbildung 15: Durchschnittliche Laufzeiten der Algorithmen für Ratio Association mit $k = 512$.

Datensatz	**Zufall**	**METIS**	**Spektral**	**Kernel-k-means++**	`KCs2`
BCSSTK31	0.1	0.38	0.51	0.1	0.1
T60K	0.1	0.18	0.25	0.1	0.1
598A	0.51	0.98	1.35	0.63	0.59
144	0.62	1.43	2.16	0.74	0.68
M14B	0.95	2.0	3	1.0	0.9
AUTO	2.13	4.5	6	2.72	2.62

Tabelle 27: Durchschnittliche Laufzeiten der Algorithmen für Ratio Association mit $k = 128$.

Datensatz	**Zufall**	**METIS**	**Spektral**	**Kernel-k-means++**	`KCs2`
BCSSTK31	0.42	0.98	1.22	0.5	0.44
T60K	0.21	0.48	0.87	0.33	0.28
598A	1.35	2.83	6.43	1.58	1.45
144	1.47	3.7	7.72	1.63	1.52
M14B	2.73	5.8	12.11	3.01	2.93
AUTO	6.25	12.3	26.32	6.98	6.57

Tabelle 28: Durchschnittliche Laufzeiten der Algorithmen für Ratio Association mit $k = 512$.

Wir betrachten nun die Ergebnisse für die Normalized Cut Zielfunktion.

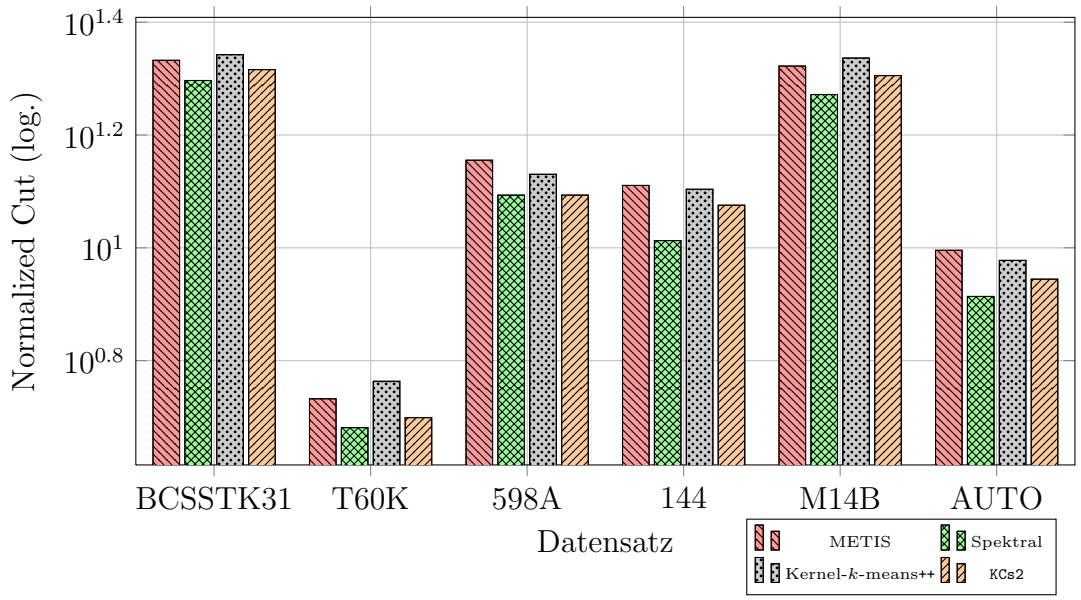


Abbildung 16: Durchschnittliche Normalized Cut Werte der Algorithmen für $k = 128$ (je kleiner, desto besser).

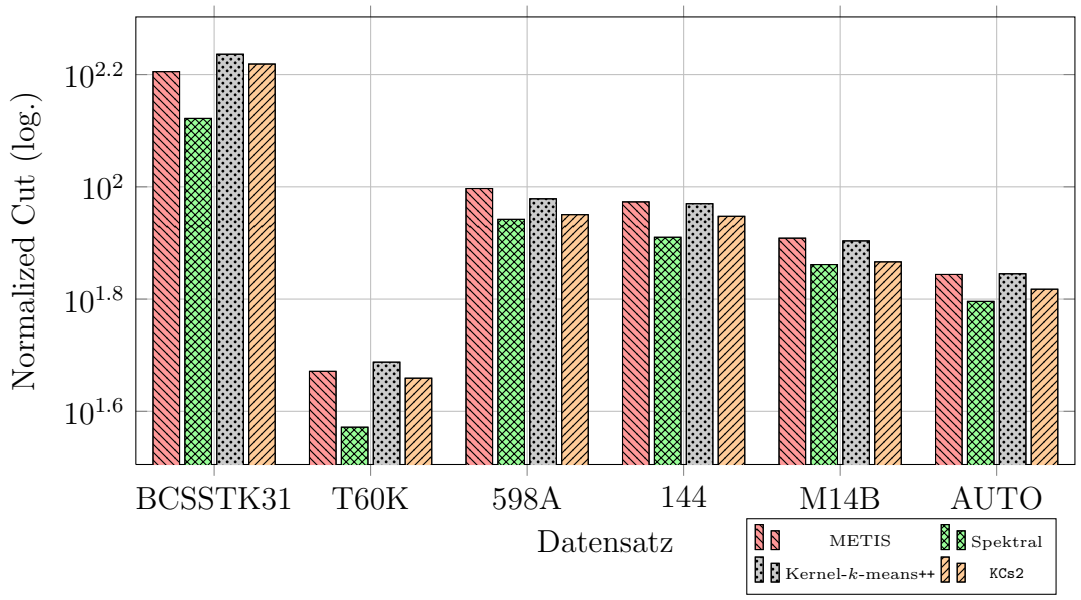


Abbildung 17: Durchschnittliche Normalized Cut Werte der Algorithmen für $k = 512$ (je kleiner, desto besser).

Datensatz	**Zufall**	**METIS**	**Spektral**	**Kernel-k-means++**	`KCs2`
BCSSTK31	29.2	21.5	19.8	22	20.7
T60K	6.9	5.4	4.8	5.8	5.0
598A	16.1	14.3	12.4	13.5	12.4
144	15.3	12.9	10.3	12.7	11.9
M14B	24.2	21	18.7	21.7	20.2
AUTO	12.1	9.9	8.2	9.5	8.8

Tabelle 29: Durchschnittliche Normalized Cut Werte der Algorithmen für $k = 128$ (je kleiner, desto besser).

Datensatz	**Zufall**	**METIS**	**Spektral**	**Kernel-k-means++**	`KCs2`
BCSSTK31	200.3	160.4	132.4	172.3	165.5
T60K	58.2	46.9	37.3	48.7	45.6
598A	124.7	99.3	87.5	95.2	89.2
144	122.5	94	81.3	93.3	88.6
M14B	112.6	81	72.7	80.1	73.5
AUTO	98.3	69.8	62.5	70	65.7

Tabelle 30: Durchschnittliche Normalized Cut Werte der Algorithmen für $k = 512$ (je kleiner, desto besser).

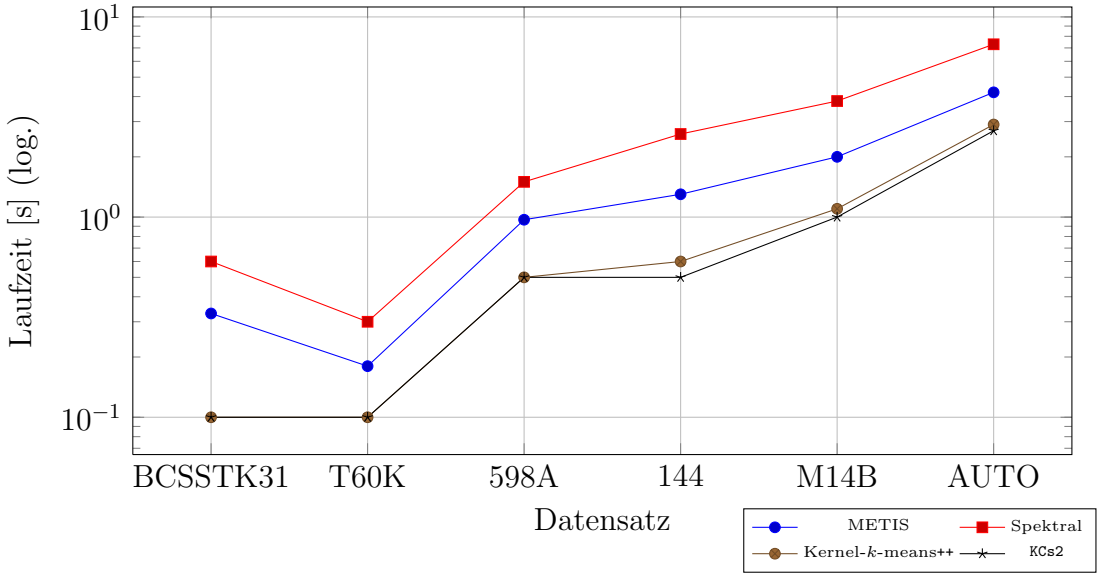


Abbildung 18: Durchschnittliche Laufzeiten der Algorithmen für Normalized Cut mit $k = 128$.

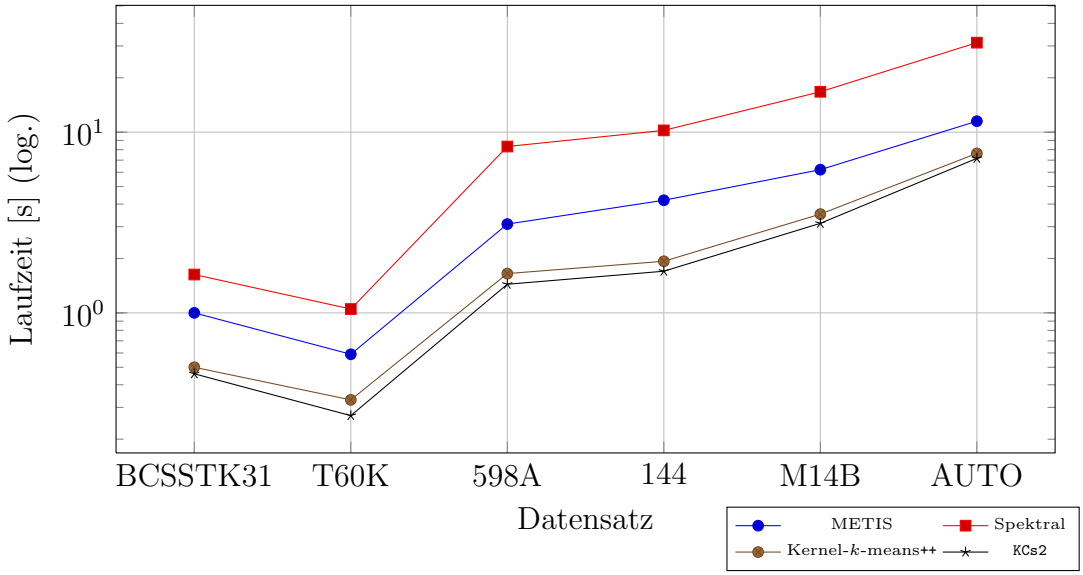


Abbildung 19: Durchschnittliche Laufzeiten der Algorithmen für Normalized Cut mit $k = 512$.

Datensatz	Zufall	METIS	Spektral	Kernel-k-means++	KCs2
BCSSTK31	0.1	0.33	0.6	0.1	0.1
T60K	0.1	0.18	0.3	0.1	0.1
598A	0.4	0.97	1.5	0.5	0.5
144	0.5	1.3	2.6	0.6	0.5
M14B	0.8	2.0	3.8	1.1	1.0
AUTO	1.9	4.2	7.3	2.9	2.7

Tabelle 31: Durchschnittliche Laufzeiten der Algorithmen für Normalized Cut mit $k = 128$.

Datensatz	Zufall	METIS	Spektral	Kernel-k-means++	KCs2
BCSSTK31	0.45	1.0	1.63	0.5	0.46
T60K	0.27	0.59	1.05	0.33	0.27
598A	1.42	3.1	8.33	1.65	1.44
144	1.59	4.2	10.23	1.93	1.70
M14B	2.81	6.2	16.73	3.52	3.13
AUTO	6.06	11.5	31.29	7.63	7.16

Tabelle 32: Durchschnittliche Laufzeiten der Algorithmen für Normalized Cut mit $k = 512$.

Bezüglich der Normalized Cut Zielfunktionswerte erzielt KCs2 eine etwas größere Verbesserung als bei der Ratio Association. Wir können den Tabellen 29 und 30 entnehmen, dass die Kernmengenkonstruktion beispielsweise im Vergleich zu Kernel-k-means++ durchschnittlich etwa 10% bessere Ergebnisse erzielt. Bei der Ratio Association lag die Verbesserung nur bei etwa 5%. Im Vergleich zum spektralen Algorithmus sind die Zielfunktionswerte von KCs2 beim Normalized Cut jedoch immer noch etwas schlechter. Mit Ausnahme des Datensatzes 598A erzielt die Kernmengenkonstruktion im Schnitt etwa 7% schlechtere Werte als der spektrale Algorithmus. METIS hingegen ist im Durchschnitt wiederum etwa 5% schlechter als KCs2.

Bei den Laufzeiten in den Tabellen 31 und 32 zeichnet sich ein ähnliches Bild wie bei der Ratio Association. Für $k = 128$ erzielt KCs2 einen Speedup von rund 10% im Vergleich zu Kernel-k-means++. Bei $k = 512$ ist der Speedup mit durchschnittlich 12% sogar noch etwas größer. Im Vergleich zum spektralen Algorithmus und METIS ist der Speedup beim Normalized Cut noch etwas größer als bei der Ratio Association. METIS ist insgesamt etwa einen Faktor $2,5$ langsamer, der spektrale Algorithmus insbesondere bei $k = 512$ sogar noch deutlich langsamer.

5 Zusammenfassung

Die Kernel Methode ist eine wirkungsvolle Technik in der Clusteranalyse. Ihr Einsatz erlaubt es, die Effektivität von Verfahren signifikant zu steigern, die auf die lineare Separierung von Clustern beschränkt sind. Angewandt auf den k-means-Algorithmus erhält man den Kernel-k-means-Algorithmus, mit dem sich beispielsweise das Graphpartitionierungsproblem lösen lässt, wie wir in Abschnitt 3.1 erläutert haben.

Der k-means++-Algorithmus sieht eine geschicktere Wahl der initialen Zentren für den k-means-Algorithmus vor. Wir haben die Kernel Methode auf k-means++ angewandt und den resultierenden Kernel-k-means++-Algorithmus vorgestellt. Dabei konnten wir zeigen, dass die $\mathcal{O}(\log k)$-Approximationseigenschaft von k-means++ erhalten bleibt. Bislang haben in der Graphpartitionierung spektrale Verfahren die qualitativ besten Ergebnisse geliefert. Diese bringen jedoch wegen der nötigen Eigenvektorberechnungen hohe Laufzeiten mit sich. Mit Kernel-k-means++ war es uns möglich, Graphpartitionierungen ohne spektrale Techniken zu berechnen, die qualitativ kompetitiv zu den bisherigen Verfahren sind. Wir konnten jedoch gleichzeitig die Laufzeiten um einen Faktor zwei verbessern. Dies ist besonders erfreulich, wenn man bedenkt, dass das modifizierte Framework Graclus die derzeit vielleicht schnellste quelloffene Software im Bereich der Graphpartitionierung ist, die von einer Reihe von quelloffenen Bildverarbeitungsprogrammen eingesetzt wird.

Wir haben zudem eine Variante der Kernmengenkonstruktion von Feldman, Schmidt und Sohler [FSS13] vorgestellt, die in den Experimenten praktikable Qualität und Laufzeit aufweist. Insbesondere die Kernel-basierte Variante liefert bessere Ergebnisse als der k-means++-Algorithmus. Mit der Kernmengenkonstruktion war es uns möglich, den Kernel-k-means-basierten Algorithmus zur Graphpartitionierung noch einmal zu verbessern.

Über diese Arbeit hinaus wäre es sicherlich interessant, zu untersuchen, ob die Kernel Methode auch auf andere k-means-Algorithmen angewandt werden kann und ob dies zu Verbesserungen führt.

Für die Kernmengenkonstruktion haben wir in der Einleitung eine mögliche Anwendung in der verteilten Clusteranalyse erwähnt. Es wäre daher vermutlich aufschlussreich, eine verteilte Variante der Konstruktion zu implementieren und diese dann auf verteilt vorliegenden zu clusternden Daten zu untersuchen. Dabei sollte der Algorithmus nicht nur auf einigen wenigen Maschinen ausgeführt werden,

sondern auf 16 oder mehr Knoten, um die Leistungsfähigkeit in verteilten Systemen zu analysieren.

Literatur

[ADHP09] Aloise, Daniel ; Deshpande, Amit ; Hansen, Pierre ; Popat, Preyas: NP-hardness of Euclidean sum-of-squares clustering. In: *Machine Learning* (2009), S. 245–248

[AMR+12] Ackermann, Marcel R. ; Märtens, Marcus ; Raupach, Christoph ; Swierkot, Kamil ; Lammersen, Christiane ; Sohler, Christian: StreamKM++: A clustering algorithm for data streams. In: *ACM Journal of Experimental Algorithmics* 17 (2012), Nr. 1

[AV07] Arthur, David ; Vassilvitskii, Sergei: k-means++: The advantages of careful seeding. In: *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, 2007, S. 1027–1035

[BGV92] Boser, Bernhard E. ; Guyon, Isabelle ; Vapnik, Vladimir: A Training Algorithm for Optimal Margin Classifiers. In: *Proceedings of the Fifth Annual ACM Conference on Computational Learning Theory, COLT 1992, Pittsburgh, PA, USA, July 27-29, 1992.*, 1992, S. 144–152

[BPKK99] Bezdek, James C. ; Pal, Mikhil R. ; Keller, James ; Krisnapuram, Raghu: *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing.* Norwell, MA, USA : Kluwer Academic Publishers, 1999

[CKV13] Celebi, M. E. ; Kingravi, Hassan A. ; Vela, Patricio A.: A comparative study of efficient initialization methods for the k-means clustering algorithm. In: *Expert Syst. Appl.* 40 (2013), Nr. 1, S. 200–210

[Cov65] Cover, T.M.: Geometrical and Statistical Properties of Systems of Linear Inequalities with Applications in Pattern Recognition. In: *Electronic Computers, IEEE Transactions on* EC-14 (1965), June, Nr. 3, S. 326–334

[DGK04] Dhillon, Inderjit ; Guan, Yuqiang ; Kulis, Brian: A Unified View of Kernel k-means, Spectral Clustering and Graph Cuts / University of Texas at Austin. 2004 (TR-04-25). – Forschungsbericht

[DGK07] Dhillon, Inderjit S. ; Guan, Yuqiang ; Kulis, Brian: Weighted Graph Cuts without Eigenvectors A Multilevel Approach. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 29 (2007), Nr. 11, S. 1944–1957

[FGS+13] Fichtenberger, Hendrik ; Gillé, Marc ; Schmidt, Melanie ; Schwiegelshohn, Chris ; Sohler, Christian: BICO: BIRCH Meets Coresets for k-Means Clustering. In: *Algorithms - ESA 2013 - 21st Annual European Symposium, Sophia Antipolis, France, September 2-4, 2013. Proceedings*, 2013, S. 481–492

[FSS13] Feldman, Dan ; Schmidt, Melanie ; Sohler, Christian: Turning big data into tiny data: Constant-size coresets for k-means, PCA and projective clustering. In: *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2013, New Orleans, Louisiana, USA, January 6-8, 2013*, 2013, S. 1434–1453

[GBK01] Georghiades, Athinodoros S. ; Belhumeur, Peter N. ; Kriegman, David J.: From Few to Many: Illumination Cone Models for Face Recognition under Variable Lighting and Pose. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 23 (2001), Nr. 6, S. 643–660

[GL96] Golub, Gene H. ; Loan, Charles F.: *Matrix computations*. 3. Auflage. Johns Hopkins University Press, 1996

[HM04] Har-Peled, Sariel ; Mazumdar, Soham: On coresets for k-means and k-median clustering. In: *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, 2004, S. 291–300

[HSS08] Hofmann, Thomas ; Schölkopf, Bernhard ; Smola, Alexander J.: Kernel methods in machine learning. In: *Ann. Statist.* 36 (2008), 06, Nr. 3, S. 1171–1220

[KK98] Karypis, George ; Kumar, Vipin: A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. In: *SIAM J. Sci. Comput.* 20 (1998), Nr. 1, S. 359–392. – ISSN 1064–8275

[KL70] Kernighan, B. W. ; Lin, S.: An Efficient Heuristic Procedure for Partitioning Graphs. In: *The Bell System Technical Journal* 49 (1970), Nr. 1, S. 291–307

[KLLL05] Kim, Dae-Won ; Lee, Ki Y. ; Lee, Doheon ; Lee, Kwang H.: Evaluation of the performance of clustering algorithms in kernel-induced feature space. In: *Pattern Recognition* 38 (2005), Nr. 4, S. 607–611

[Lan50] Lanczos, Cornelius: An iteration method for the solution of the eigenvalue problem of linear differential and integral operators. In: *Journal of Research of the National Bureau of Standards* 45 (1950), S. 255–282

[Lic13] Lichman, M.: *UCI Machine Learning Repository*. `http://archive.ics.uci.edu/ml`. Version: 2013

[Llo82] Lloyd, Stuart P.: Least squares quantization in PCM. In: *IEEE Transactions on Information Theory* 28 (1982), Nr. 2, S. 129–136

[Lux07] Luxburg, Ulrike von: A Tutorial on Spectral Clustering. In: *Statistics and Computing* 17 (2007), Nr. 4, S. 395–416

[MS84] Megiddo, Nimrod ; Supowit, Kenneth J.: On the Complexity of Some Common Geometric Location Problems. In: *SIAM J. Comput.* 13 (1984), Nr. 1, S. 182–196

[NJW01] Ng, Andrew Y. ; Jordan, Michael I. ; Weiss, Yair: On Spectral Clustering: Analysis and an algorithm. In: *Advances In Neural Information Processing Systems*, 2001, S. 849–856

[ORSS12] Ostrovsky, Rafail ; Rabani, Yuval ; Schulman, Leonard J. ; Swamy, Chaitanya: The effectiveness of Lloyd-type methods for the k-means problem. In: *J. ACM* 59 (2012), Nr. 6, S. 28

[SC04] Shawe-Taylor, John ; Cristianini, Nello: *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004

[Sch14] Schmidt, Melanie: *Coresets and Streaming Algorithms for the k-means Problem and Related Clustering Objectives*, Technische Universität Dortmund, Diss., 2014

[SM00] Shi, Jianbo ; Malik, Jitendra: Normalized Cuts and Image Segmentation. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 22 (2000), Nr. 8, S. 888–905

[SWC04] Soper, Alan J. ; Walshaw, Chris ; Cross, Mark: A Combined Evolutionary Search and Multilevel Optimisation Approach to Graph-Partitioning. In: *J. Global Optimization* 29 (2004), Nr. 2, S. 225–241

[Swi08] Swierkot, Kamil: *Graph-Partitionierung mit Kernel-Clustering. Bachelorarbeit, Universität Paderborn*. 2008

Eidesstattliche Versicherung

_____ _____

Name, Vorname Matr.-Nr.

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem Titel

_____

_____

_____

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

_____ _____

Ort, Datum Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -)

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin“) zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

_____ _____

Ort, Datum Unterschrift