Algorithmen für dynamische geometrische Datenströme

Gereon Frahling

Google Research
76 Ninth Avenue
New York, NY 10011
gereon@google.com

Abstract: Die zunehmende Vernetzung moderner Computersysteme produziert gewaltige Datenmengen, deren Behandlung die heutige Informatik vor unzählige Probleme stellt. Traditionelle Algorithmen, deren Laufzeit zu stark von der Größe der Daten abhängt, sind häufig nicht anwendbar. Oft sind die Eingabedaten sogar zu groß für die verfügbaren Speichermedien. Wir stellen neue, grundlegende Algorithmen zur Analyse gewaltiger Datenmengen vor, die unter minimalen Anforderungen an den Speicher der Computersysteme beweisbar gute Zusammenfassungen der Daten berechnen.

Zunächst erarbeiten wir grundlegende Ergebnisse zum Ziehen von Stichproben aus dynamischen Datenströmen. Mit Hilfe dieser Stichproben können vielfältige Aussagen über die Eingabedaten getroffen werden, was wir anhand von Beispielen beweisen.

Anschließend wenden wir uns dem Clustering gewaltiger Datensätze zu. Wir entwickeln eine Methode, große Datenmengen zu sogenannten *Coresets* zusammenzufassen. Die Methode erzeugt beweisbar k-means- und k-median-Clusterings mit beliebig guter Approximationsgarantie und ist anwendbar in vielen verschiedenen Modellen: Die Clusterings können mit wenig Speicher für dynamische Datenströme berechnet werden, sie lassen sich in sublinearer Zeit durch Bereichsanfragen erstellen, und auch sich bewegende Punkte können effizient zusammengefasst werden. Eine Implementierung zeigt, dass die Methode auf großen realen Testdaten gute Clusterings deutlich schneller berechnen kann als häufig angewendete traditionelle Algorithmen.

Schließlich beschäftigen wir uns mit der Analyse der Struktur von großen Graphen wie dem Webgraph. Wir entwickeln neue Methoden, um die Anzahl von Teilgraphen (z.B. Dreiecken) eines Graphen zu zählen, der als Datenstrom von Kanten gegeben ist.

1 Einführung

Einer ausführlichen Studie der Universität Berkeley zufolge [Lyman and Varian 2003] wurden im Jahr 2002, dem letzten Jahr, für welches solche Daten vorliegen, etwa 23 Exabytes (d.h. 23 Millionen Terabytes) an Informationen erzeugt. Nur etwa 20% dieser Information können auf Papier oder magnetischen und optischen Speichermedien gespeichert werden. Der Großteil jedoch wird nach der Erzeugung über Datenkanäle geschickt und daraufhin sofort wieder verworfen.

Neben Telefon-, Radio- und Fernsehsignalen bestehen diese vergessenen Daten vor allem aus Informationen über Internetverbindungen. Ein Beispiel ist der Internetverkehr an einem großen Backbone Router. Selbstverständlich ist der Router nicht in der Lage, alle von ihm verarbeiteten Pakete lokal zu speichern. Dennoch ist es aus vielen Gründen

ratsam, Statistiken über die verarbeiteten Pakete aufrechtzuerhalten, um später daraus Rückschlüsse auf den gesamten Internetverkehr zu ziehen. *Datenstromalgorithmen* versuchen daher, die Daten sequentiell in ihrer Reihenfolge zu lesen und dabei nur eine kleine Zusammenfassung zu speichern. Da von ihren Resultaten häufig wichtige Entscheidungen abhängen (Einführung neuer Backbones, Ändern von Netzwerkstruktur), sind Datenstromalgorithmen erforderlich, die Zusammenfassungen mit beweisbar hoher Genauigkeit aufrechterhalten.

Selbst für den Großteil der gespeicherten Daten sind Datenstromalgorithmen in vielen Fällen sinnvoll. Gemäß der oben zitierten Studie werden 90% der gespeicherten Daten auf Festplatten gehalten, von denen sie in hoher Geschwindigkeit nur sequentiell wieder ausgelesen werden können.

In meiner Dissertation [Fra06] stelle ich neue, grundlegende Algorithmen vor, um Statistiken in solchen Szenarien aufrechtzuerhalten. Die Algorithmen arbeiten mit wenig Speicher und können während des Lesens eines großen Datenstromes uniforme Stichproben und kompliziertere Strukturen wie ϵ-Nets oder sogar komplette Clusterings der Daten aufrechterhalten. Für alle Algorithmen werden beliebig gute Approximationsgarantien bewiesen. Aufgrund ihrer Allgemeinheit sind die Algorithmen in unzähligen Bereichen einsetzbar.

2 Dynamische Datenströme

Im Großteil dieser Arbeit gehen wir von dem allgemeinen Modell der dynamischen Datenströme aus. Ein solcher Datenstrom besteht aus Einfüge- und Löschoperationen von Elementen in eine (Multi-)Menge Q, die im allgemeinen zu groß ist, um sie zu speichern. Um Aussagen über die Struktur dieser großen Menge zu treffen, brauchen wir Informationen über die Beziehungen zwischen einzelnen Elementen, vor allem über deren Ähnlichkeiten und Unterschiede. Da das Berechnen von Unterschieden aller Paare von Elementen nicht möglich ist, wird in der Praxis häufig eine Distanzfunktion zwischen solchen Paaren implizit definiert: Zuerst bildet man alle Elemente aus Q durch eine Abbildung $\alpha : Q \to \mathbb{R}^d$ auf Punkte in einem d-dimensionalen Raum ab. Die Distanz zwischen zwei Elementen $p, q \in Q$ wird dann durch den euklidischen Abstand $d(\alpha(p), \alpha(q))$ bestimmt. Der dynamische Datenstrom von Einfüge- und Löschoperationen von Elementen in Q transformiert sich somit in einen Datenstrom aus Einfüge und Löschoperationen von Punkten aus $\mathbb{R}^d$.

Dynamische Geometrische Datenströme. Unser Modell geht daher aus von einem Datenstrom, der aus m Einfüge- und Löschoperationen von Punkten des diskreten euklidischen Raumes $\{1, \ldots, \Delta\}^d$ in eine dynamische (Multi-)Menge Q besteht. Die Zeit- und Platzkomplexität aller unserer Algorithmen wird nur polylogarithmisch von Δ abhängen.

Uniforme Stichproben aus dynamischen Datenströmen. Wir beantworten zunächst die Frage, wie man effizient uniforme Stichproben aus der Menge Q erhält. Die Schwierigkeit liegt hier darin, dass die Menge unter vielen Einfügeoperationen stark wachsen kann. Schrumpft sie anschließend durch Löschoperationen wieder auf eine kleine Anzahl von

Elementen, müssen wir diese wenigen Elemente exakt bestimmen können. Wir stellen verschiedene Methoden vor, um während des gesamten Datenstromes uniforme Stichproben der Menge Q aufrechtzuerhalten. Unser Algorithmus benötigt dazu nur eine Anzahl von Speicherbits, die polylogarithmisch in der Größe des Datenstromes ist. Auch die Zeit, die benötigt wird, um eine Einfüge- und Löschoperation zu verarbeiten, ist polylogarithmisch in der Größe des Datenstromes. Unter mehreren Ergebnissen zum Ziehen von Stichproben aus dynamischen Datenströmen möchte ich eines exemplarisch angeben:

Theorem 1 *[FIS05] Gegeben sei eine Sequenz von Einfüge- und Löschoperationen von Punkten des diskreten Raumes $\{1, \ldots, \Delta\}^d$ in eine Menge Q. Es existiert eine Datenstruktur, die zu jedem Zeitpunkt des Datenstromes mit Wahrscheinlichkeit $1 - \delta$ eine Stichprobe von s Punkten $r_0, \ldots, r_{s-1} \in Q$ zurückgibt und mit Wahrscheinlichkeit δ einen Rückgabewert $FAIL$. Die zurückgegebenen Samples sind unabhängig voneinander gezogen und können daher Duplikate beinhalten. Die statistische Differenz zwischen der Verteilung eines dieser Samples und einer uniformen Verteilung ist höchstens $\frac{\delta}{\Delta^d}$, d.h. insbesondere auch $\Pr[r_i = p] = \frac{1}{|Q|} \pm \frac{\delta}{\Delta^d}$ für jedes $p \in Q$.*
Der Algorithmus arbeitet mit höchstens $O\left(\left(s + \log(1/\delta)\right) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ Speicherbits.

In der Dissertation werden weiterhin viele Verallgemeinerungen und Spezialisierungen der Sampling-Technik betrachtet. Unabhängig von unserer Veröffentlichung [FIS05] wurde ein ähnliches Ergebnis auch von [Cormode, Muthukrishnan und Rozenbaum 2005] erreicht. Diese zwei parallelen Veröffentlichungen bilden inzwischen die Grundlage für die approximative Lösung vieler Probleme auf dynamischen Datenströmen.

Anwendungen der Stichproben. Wir möchten nun exemplarisch einige direkte Anwendungen des Samplings in Datenströmen angeben. Das Aufrechterhalten von Samples gibt uns beispielsweise die Möglichkeit, zu jedem Zeitpunkt des Datenstromes Anfragen über die Verteilung der Objekte in der aktuellen Menge Q (z.B. „Wieviele Punkte sind in folgendem Rechteck enthalten?“) approximativ zu beantworten. Solche Anfragen haben ein sehr breites Anwendungsfeld in Datenbanken, der Analyse von Internetverbindungen und unzähligen anderen Bereichen. In den Jahren seit den zwei parallelen Veröffentlichungen über Stichproben in Datenströmen wurden beispielsweise von Telefonunternehmen große Systeme gebaut, die diese Techniken einsetzen, um Statistiken über den Telefon- und Datenverkehr zu erstellen.

Definition 2.1 (Bereichsräume, VC-Dimension) *Sei X eine Menge von Objekten und R eine Familie von Untermengen von X. Wir nennen dann das System $\Sigma = (X, R)$ einen* Bereichsraum *(range space). Die Elemente aus R heißen* Bereiche *von Σ. Falls X eine endliche Menge ist, dann heißt Σ* endlicher Bereichsraum.

Wir sagen, dass $A \subset X$ vollständig von R zerteilt *wird, falls $\{A \cap r | r \in R\} = 2^A$. Die* Vapnik-Chervonenkis-Dimension (VC-Dimension) *des Bereichsraumes $\Sigma = (X, R)$ ist die Größe der größten Teilmenge von X, die vollständig von R* zerteilt *wird.*

Ein Beispiel für einen Bereichsraum ist die Menge aller achsenparallelen Rechtecke. Man kann zeigen, dass ihre VC-Dimension genau 4 ist.

Definition 2.2 (ϵ-Netze, ϵ-Approximationen) *Sei $\Sigma = (X, R)$ ein endlicher Bereichsraum. Eine Teilmenge $N \subset X$ heißt ϵ-Netz, falls $N \cap r \neq \emptyset$ für jedes $r \in R$ mit $|r| \geq \epsilon|X|$.*

$A \subseteq X$ heißt ϵ-Approximation, falls für jedes $r \in R$ gilt: $\left|\frac{|A \cap r|}{|A|} - \frac{|r|}{|X|}\right| \leq \epsilon$.

Haben wir eine ϵ-Approximation einer Punktmenge, so können wir nach dem Einlesen des Datenstromes beispielsweise Anfragen über den Anteil der Punkte in einem gegebenen Rechteck bis auf einen beliebig kleinen Fehler ϵ approximativ beantworten. Dies gilt für alle Anfragen über den Anteil der Punkte in Bereichen mit kleiner VC-Dimension. Mit Hilfe unserer Stichprobentechnik kann man solche ϵ-Approximationen und ϵ-Netze effizient in dynamischen Datenströmen aufrechterhalten:

Theorem 2 *[FIS05] Gegeben sei eine Sequenz von Einfüge- und Löschoperationen von Punkten des diskreten Raumes $\{1, \ldots, \Delta\}^d$ in eine Menge Q und ein Bereichsraum (X, R) der VC-Dimension $\mathcal{D}$. Es existiert eine Datenstruktur, die zu jedem Zeitpunkt des Datenstromes mit Wahrscheinlichkeit $1 - \delta$ eine Menge A von $\widetilde{O}(\frac{\mathcal{D}+\log(1/\delta)}{\epsilon^2})$ Punkten zurückgibt, die eine ϵ-Approximation von (X, R) ist. Der Algorithmus arbeitet mit höchstens $O\left(\frac{1}{\epsilon^2}\left(\mathcal{D}\log\frac{\mathcal{D}}{\epsilon} + \log\frac{1}{\delta}\right) d^3 \log^3\frac{\Delta}{\delta}\right)$ Speicherbits.*

Weiterhin existiert ein Algorithmus, der zu jedem Zeitpunkt mit Wahrscheinlichkeit $1-\delta$ eine Menge N von $\widetilde{O}(\frac{\mathcal{D}+\log(1/\delta)}{\epsilon})$ von Punkten zurückgibt, welche ein ϵ-Netz von (X, R) ist. Der Algorithmus benutzt höchstens $O\left((\frac{\log(1/\delta)}{\epsilon} + \frac{\mathcal{D}}{\epsilon}\log\frac{\mathcal{D}}{\epsilon}) \cdot d^2 \cdot \log^2(\Delta/\delta)\right)$ Speicherbits.

Als weitere Anwendung zeigen wir in der Dissertation, wie der Wert eines minimalen Spannbaumes, der alle momentanen Punkte der Menge Q verbindet, effizient geschätzt werden kann, wenn die Punkte als dynamischer Datenstrom gegeben sind. Der Wert des minimalen Spannbaumes könnte beispielsweise Aufschluss geben über die benötigte Energie einer Kommunikationsstruktur, die alle Punkte verbindet.

Theorem 3 *[FIS05] Gegeben sei eine Sequenz von Einfüge- und Löschoperationen von Punkten des diskreten Raumes $\{1, \ldots, \Delta\}^d$ in eine Menge Q. Es gibt einen Datenstromalgorithmus, der zu jedem Zeitpunkt mit Wahrscheinlichkeit $1-\delta$ eine $(1+\epsilon)$-Approximation des Wertes des euklidischen minimalen Spannbaumes von Q bestimmt. Der Algorithmus benötigt höchstens $O(\log^3(1/\delta) \cdot (\log(\Delta)/\epsilon)^{O(d)})$ Speicherbits.*

3 Die Coreset - Methode

Wir wenden uns nun dem Problem des Clusterings großer Datenmengen zu. Das Problem, Daten zu sinnvollen Einheiten zusammenzufassen, gehört zu den am meisten untersuchten Optimierungsproblemen überhaupt und ist häufig der erste Schritt, die Struktur großer Datenmengen zu verstehen. Ein sinnvolles Zusammenfassen der Datenelemente kann weiterhin die Datenmenge auf ein Maß reduzieren, welches die Ausführung verschiedener aufwändiger Algorithmen erst ermöglicht.

Wir beschäftigen uns unter anderem mit dem wohl beliebtesten Clustering-Verfahren, dem *k-means Clustering*. Wir zeigen erstmals eine Methode, eine $(1+\epsilon)$-approximative Lösung der k-means Zielfunktion aufrechtzuerhalten, wenn die Punkte als dynamischer Datenstrom gegeben sind. Dies führt direkt zu der Möglichkeit, sinnvolle Zusammenfassungen riesiger Datenmengen zu erstellen, ohne die Daten selbst jemals speichern zu müssen.

Unsere Methode berechnet aus der riesigen Eingabepunktmenge eine sehr kleine gewichtete Punktmenge, das sogenannte *Coreset*. Dabei stellt sie sicher, dass auf dem Coreset berechnete Clusteringlösungen immer auch $(1+\epsilon)$-approximative Clusteringlösungen für die Ausgangsmenge sind.

Definitionen der Probleme und Begriffe. Wir definieren zunächst einige häufig verwendete Clustering-Zielfunktionen. Als Eingabe setzen wir eine endliche gewichtete Punktmenge $P \subset \mathbb{R}^d$ und eine Funktion der Punktgewichte $w : P \to \mathbb{R}^+$ voraus. Wir bezeichnen mit $d(p,q)$ den Euklidischen Abstand zwischen zwei Punkten $p, q \in \mathbb{R}^d$.

Definition 3.1 *Das* k-median Clusteringproblem *fragt nach einer Menge $C = \{c_1, \ldots, c_k\}$ von k Clusterzentren in $\mathbb{R}^d$ und einer Partition der Punkte P in k Cluster $C_1, \ldots, C_k$, so dass $Median(P, C, C_1, \ldots, C_k) := \sum_{i=1}^{k} \sum_{p \in C_i} w(p) \cdot d(p, c_i)$ minimiert wird. In der ungewichteten Version des Problems sind alle Punktgewichte 1. Für den Fall, dass die Partition $C_1, \ldots, C_k$ jeden Punkt dem nächsten Clusterzentrum zuordnet, definieren wir die Kurzschreibweise $Median(P, C) := Median(P, C, C_1, \ldots, C_k)$.*

Das k-means Clusteringproblem *fragt nach einer Menge von Clusterzentren und einer Partition, so dass $Means(P, C, C_1, \ldots, C_k) := \sum_{i=1}^{k} \sum_{p \in C_i} w(p) \cdot d^2(p, c_i)$ minimiert wird.*

Das MaxCut Clusteringproblem *fragt nach einer Partition der Punkte in zwei Mengen $\mathcal{S}$ und $\mathcal{T}$, so dass $\sum_{p \in \mathcal{S}} \sum_{q \in \mathcal{T}} d(p, q)$ maximiert wird.*

In der Dissertation zeigen wir, dass relativ schwache Anforderungen an ein Problem ausreichen, um es mit unserer Coreset-Methode effektiv zu approximieren. Beispiele für Probleme, die diese Voraussetzungen erfüllen, sind neben den oben definierten Problemen auch *Average Distance*, *MaxMatching*, und *Maximum Travelling Salesman*.

Definition 3.2 (Coresets) *Sei P eine gewichtete Menge von n Punkten in $\mathbb{R}^d$. Eine gewichtete Punktmenge P_{core} in $\mathbb{R}^d$ heißt* ϵ-coreset *für P (bzgl. des k-median problems), wenn für jede Menge C von k Zentren gilt:*
$(1-\epsilon) \cdot Median(P, C) \leq Median(P_{core}, C) \leq (1+\epsilon) \cdot Median(P, C)$.
Wir definieren ein Coreset für P bzgl. des k-means-Problems analog.

Die genaue Definition von Coresets für das MaxCut Problem ist etwas komplizierter. Coresets für das MaxCut-Problem sind gewichtete Punktmengen mit einer zugehörigen Abbildung $\gamma : P \to P_{core}$. Wir verlangen, dass das Gewicht eines jeden Coresetpunktes dem Gewicht der abgebildeten Punkte entspricht und dass für jede Partition der Punktmenge die MaxCut-Zielfunktion approximiert wird durch die MaxCut-Zielfunktion der entsprechenden Partition der Coresetmenge. Eine genaue Definition (auch für Average Distance,

Maximum Matching, Maximum Travelling Salesman und verallgemeinerte Probleme) ist in der Dissertation nachzulesen.

Beschreibung der Coreset-Methode anhand von k-Median. Wir nehmen an, dass alle Punkte in einem Würfel der Seitenlänge 1 liegen, zum Beispiel $[0,1]^d$. Dies kann durch einfaches Skalieren der Punktmenge erreicht werden. Weiterhin nehmen wir an, dass die optimale Zielfunktion des Problems mindestens den Wert $1/\widetilde{\Delta}$ besitzt. Wir benötigen hier nur eine sehr schwache Schranke an $\widetilde{\Delta}$, da der von unserer Methode benutzte Speicher und Platz nur logarithmisch von $\widetilde{\Delta}$ abhängen wird.

Unsere Methode betrachtet Z verschieden große Gitter $\mathcal{G}_0, \ldots, \mathcal{G}_{Z-1}$ (für einen Parameter $Z = O(d \cdot \log(k \cdot n \cdot d \cdot \widetilde{\Delta}))$). Die Zellgröße einer Gitterzelle des Gitters $\mathcal{G}_i$ ist $\frac{1}{2^i}$. Jede Zelle in $\mathcal{G}_i$ besteht daher aus 2^d Zellen des Gitters $\mathcal{G}_{i+1}$.

Unser Ziel ist es, in jedem Gitter die *schweren Zellen* zu identifizieren. Dies sind Zellen, die mehr als eine bestimmte Anzahl von Punkten enthalten. Wir parametrisieren diese Schranke durch einen Parameter δ, den wir später exakt festsetzen werden.

Definition 3.3 (Schwere Zellen) *Wir nennen eine Zelle des Gitters $\mathcal{G}_i$* schwer*, wenn sie mindestens $\delta \cdot 2^i$ Punkte enthält.*

Unsere Methode besteht aus zwei Phasen. In einer ersten Phase berechnen wir die Coresetpunkte. In der zweiten Phase bestimmen wir das Gewicht der Coreset-Punkte.

Der Algorithmus der ersten Phase startet mit dem gröbsten Gitter $\mathcal{G}_0$. Er identifiziert alle schweren Zellen in $\mathcal{G}_0$ (Das Gitter $\mathcal{G}_0$ besteht im Grunde nur aus einer schweren Zelle, die die gesamte Punktmenge umfasst). Danach wendet er folgende iterative Methode auf allen Gitterstufen an. Nehmen wir an, der Algorithmus hat die schweren Zellen im Gitter $\mathcal{G}_i$ identifiziert. Er zerteilt nun iterativ jede schwere Zelle $\mathcal{C} \in \mathcal{G}_i$ in 2^d quadratische Unterzellen des Gitters $\mathcal{G}_{i+1}$. Wir nennen $\mathcal{C}$ dann die Mutterzelle dieser Unterzellen. Wenn keine dieser Unterzellen schwer ist, führt der Algorithmus einen Coreset-Punkt im Mittelpunkt der Zelle ein. Andernfalls werden in der nächsten Iteration die schweren Unterzellen weiter unterteilt. Die Unterteilung stoppt in jedem Fall, da zu einem bestimmten Zeitpunkt eine schwere Zelle mehr als n Punkte enthalten müsste.

In der zweiten Phase bestimmt der Algorithmus das Gewicht der Coresetpunkte. Wir können uns diese zweite Phase vorstellen als eine Zuordnung von Punkten der Ausgangspunktmenge zu den entsprechenden Coreset-Punkten. Das Gewicht eines Coresetpunktes entspricht dann der Anzahl der zugeordneten Punkte aus der Eingangspunktmenge P. Der Algorithmus ordnet die Punkte den Coresetpunkten so zu, dass jeder Punkt $p \in P$ in der kleinsten schweren Zelle verbleibt, in der er enthalten ist. Da Phase 1 sicherstellt, dass jede schwere Zelle einen Coresetpunkt enthält, ist diese Invariante leicht zu erhalten.

Wir bezeichnen mit P_{core} das vom Algorithmus berechnete Coreset und mit Opt den besten erreichbaren Zielfunktionswert. In der Dissertation wurden die folgenden Eigenschaften bewiesen:

Lemma 3.4 *Falls $\delta \leq \frac{\epsilon^{d+1} \cdot Opt}{4 \cdot k \cdot 10^d \cdot (1+\log n) \cdot d^{(d+1)/2}}$, dann ist P_{core} ein ϵ-coreset für P bezüglich des k-median-Problems. Falls $\delta \geq \frac{\epsilon^{d+1} \cdot Opt}{8 \cdot k \cdot 10^d \cdot (1+\log n) \cdot d^{(d+1)/2}}$, ist die Größe des Coresets P_{core} beschränkt durch $O(k \cdot \log n / \epsilon^{d+1})$.*

Um ein ϵ-coreset zu berechnen, muss man daher nur noch einen entsprechenden Wert von δ bestimmen. Dazu kann man eine untere und obere Schranke an die möglichen Werte von δ herleiten. Man testet daraufhin für verschiedene Werte von j und $\delta := \delta_0 \cdot 2^j$, ob das berechnete Coreset die gewünschte Größe hat. Durch binäre Suche auf den Werten von j kann man effizient den kleinsten Wert von j finden, so dass das entsprechende Coreset die gewünschte Größe nicht überschreitet. Es ist leicht zu sehen, dass dieses Coreset nach Lemma 3.4 dann auch ein ϵ-coreset für die Ausgangspunktmenge sein muss.

Theorem 4 (Coresets für k-median) *[FS05] Wenn wir ein Orakel voraussetzen, welches Anfragen an die Anzahl von Punkten in Gitterzellen in konstanter Zeit beantworten kann, können wir in Zeit $O(k \cdot \log n \cdot \log(\widetilde{\Delta} \cdot n/\epsilon) \cdot \log\log(\widetilde{\Delta} \cdot n)/\epsilon^{d+1})$ ein ϵ-coreset der Größe $O(k \cdot \log n/\epsilon^{d+1})$ für k-median berechnen.*

Ein entsprechendes Orakel kann in $O(n \cdot \log(n \cdot \widetilde{\Delta}/\epsilon))$ Zeit konstruiert werden. Aus der Lösung auf dem Coreset kann in Zeit $O((k \cdot \log n/\epsilon)^{O(1)})$ eine $(1+\epsilon)$-approximative Lösung für k-median bestimmt werden.

Ein genauer Vergleich mit sonstigen bekannten Methoden [Har-Peled, Mazumdar 2004] zeigt, dass für bestimmte, sinnvolle Werte von k,ϵ und d und sehr große Werte von n unsere Methode den schnellsten PTAS für das k-median Problem darstellt.

Erweiterungen dieser Methode können analog auch auf die anderen oben genannten Probleme angewandt werden. Für das euklidische MaxCut-Problem erhalten wir den schnellsten bisher bekannten PTAS für große Werte von n.

Theorem 5 (Coresets für k-means) *[FS05] Wenn wir ein Orakel voraussetzen, welches Anfragen an die Anzahl von Punkten in Gitterzellen in konstanter Zeit beantworten kann, können wir in Zeit $O(k \cdot \log n \cdot \log(\widetilde{\Delta} \cdot n/\epsilon) \cdot \log\log(\widetilde{\Delta} \cdot n)/\epsilon^{d+2})$ ein ϵ-coreset der Größe $O(k \cdot \log n/\epsilon^{d+2})$ für k-means berechnen. Aus dem Coreset kann in $O(k^{2k+1} \cdot \epsilon^{-2kd-d-2} \cdot \log^{k+1} n)$ Zeit eine $(1+\epsilon)$-approximative Lösung für k-means bestimmt werden.*

Theorem 6 (Coresets für MaxCut) *[FS05] Wenn wir ein Orakel voraussetzen, welches Anfragen an die Anzahl von Punkten in Gitterzellen in konstanter Zeit beantworten kann, können wir in Zeit $O(k \cdot \log n \cdot \log(\widetilde{\Delta} \cdot n/\epsilon) \cdot \log\log(\widetilde{\Delta} \cdot n)/\epsilon^{d+1})$ ein ϵ-coreset der Größe $O(k \cdot \log n/\epsilon^{d+1})$ für MaxCut berechnen. Aus dem Coreset kann in $O(\log^2 n \cdot 2^{((1/\epsilon)^{O(1)})})$ Zeit eine $(1+\epsilon)$-approximative Lösung für MaxCut bestimmt werden.*

4 Anwendungen der Coreset-Methode

Dynamische Datenströme. Man kann zeigen, dass für die Berechnung eines Coresets nur approximative Informationen über schwere Zellen benötigt werden. Ist uns die Menge aller schweren Zellen gegeben und für jede schwere Zelle eine $(1+\epsilon)$-Approximation der Anzahl der enthaltenen Punkte, so können wir effizient ein Coreset berechnen. Die Informationen können durch Stichproben von Punkten in den unterschiedlichen Gittern gewonnen werden. Obwohl dazu die Stichprobengröse in den groben Gittern linear in

der Punktanzahl sein muss, kann die Information, wieviele Stichprobenpunkte in den verschiedenen Zellen enthalten sind, in polylogarithmischem Platz gespeichert werden. Dies ermöglicht uns, mit Hilfe der zuvor beschriebenen Sampling - Techniken Datenstromalgorithmen für Clusteringprobleme zu entwerfen:

Theorem 7 *[FS05] Gegeben sei eine Sequenz von Einfüge- und Löschoperationen von Punkten des diskreten Raumes* $\{1, \ldots, \Delta\}^d$ *in eine Menge* Q*. Es gibt einen Datenstromalgorithmus, der zu jedem Zeitpunkt mit Wahrscheinlichkeit* $1 - \psi$ *ein* ϵ*-coreset für* k*-median,* k*-means, und MaxCut aufrechterhält. Die Datenstruktur benötigt für* k*-median* $\widetilde{O}(k \cdot \log^6(\Delta) \cdot \log(\Delta/\psi)/\epsilon^{d+3})$ *Speicherbits, für* k*-means* $\widetilde{O}(k \cdot \log^6(\Delta) \cdot \log(\Delta/\psi)/\epsilon^{d+4})$ *Speicherbits und für MaxCut* $\widetilde{O}(\log^6(\Delta) \cdot \log(\Delta/\psi)/\epsilon^{d+3}))$ *Speicherbits.*

Einfüge- und Löschoperationen können in $\widetilde{O}(k \cdot \log^6(\Delta) \cdot \log(\Delta/\psi)/\epsilon^{d+3})$ *Zeit für* k*-median bearbeitet werden, in* $\widetilde{O}(k \cdot \log^6(\Delta) \cdot \log(\Delta/\psi)/\epsilon^{d+4})$ *Zeit für* k*-means und in* $\widetilde{O}(\log^6(\Delta) \cdot \log(\Delta/\psi)/\epsilon^{d+3}))$ *Zeit für MaxCut.*

Für Datenströme, die keine Löschoperationen enthalten, werden in der Dissertation weiterhin deutlich bessere Zeit- und Speicherschranken bewiesen [FS05].

Parallele Berechnung von k-means Lösungen. Unsere Coreset-Technik kann sehr effizient auf parallelen Rechensystemen implementiert werden. Die Punkte werden auf verschiedene Maschinen verteilt. Nach der parallelen Berechnung der Gitterstatistiken der verschiedenen Teilmengen können die Statistiken auf einem Computer gesammelt werden, um daraus eine Gitterstatistik für die gesamte Punktmenge zu berechnen. Aus dieser gesamten Gitterstatistik kann dann das Coreset effizient ermittelt werden.

Eine so implementierte k-means-Methode benötigt nur eine Kommunikationsrunde des parallelen Systems. Die Menge der ausgetauschten Information ist logarithmisch in der Punktanzahl, was die Methode sehr gut anwendbar macht für sehr große verteilte Datenmengen.

Kinetische Datenstrukturen. Auch auf sich bewegende Punktmengen kann die Coreset-Technik angewandt werden. Vor der Veröffentlichung unseres folgenden Resultats war nicht klar, ob $\Omega(n^2)$ Berechnungen benötigt werden, um MaxCut-Clusterings von n Punkten unter linearen Bewegungen aufrechtzuerhalten. Wir zeigen, dass eine Aufrechterhaltung deutlich effizienter in fast linearer Zeit möglich ist. Wir gehen davon aus, dass das Resultat auch auf k-median und k-means - Clusterings übertragbar ist.

Theorem 8 *Es existiert eine kinetische Datenstruktur, die mit Wahrscheinlichkeit* $1 - \psi$ *eine* $(1 + \epsilon)$*-Approximation des Euclidean MaxCut problems aufrechterhält. Die Datenstruktur beantwortet Anfragen der Form „Auf welche Seite des Schnittes gehört ein Punkt* p*?“ in* $O(\log^2 n \cdot \log\log n \cdot \epsilon^{-2(d+1)} \cdot 2^{1/\epsilon^{O(1)}})$ *Zeit. Unter linearer Bewegung behandelt die Datenstruktur* $\widetilde{O}(\frac{n \log(\psi^{-1})}{\epsilon^{d+3}})$ *Ereignisse, jedes in* $O(\log^2 n)$ *Zeit. Ein Flugplanupdate eines Punktes kann in* $\widetilde{O}(\frac{\log^4 n \cdot \log(\psi^{-1})}{\epsilon^{d+3}})$ *mittlerer erwarteter Zeit beantwortet werden, wobei über die Worst Case Updatezeiten der Punkte an beliebigen Zeitpunkten gemittelt wird. Die Datenstruktur benötigt erwartete Setup-Zeit* $\widetilde{O}(\frac{n \cdot \log(\psi^{-1})}{\epsilon^{d+3}})$.

5 Implementierungsresultate

In der Praxis werden zum Clustering von Punktmengen häufig iterative Algorithmen wie Lloyd's Algorithmus (auch k-means-Algorithmus genannt) eingesetzt. Die von uns vorgestellte Coreset-Technik kann hier helfen, deutliche Geschwindigkeitssteigerungen zu erreichen. Wir betrachten als Beispiel den Algorithmus KM-Hybrid, welcher in jeder Iteration entweder einen k-Means-Schritt ausführt oder zufällige Veränderungen des Clusterings vornimmt, um ein Festsetzen in lokalen Minima des k-means Algorithmus zu vermeiden.

Da die ersten Schritte iterativer Algorithmen nur eine begrenzte Genauigkeit der Punktmenge benötigen, können wir zuerst die Punktmenge durch das Coreset auf ein Mindestmaß komprimieren. Die ersten Iterationen können so in einem Bruchteil der sonst benötigten Zeit ausgeführt werden. Durch sukzessives Vergrößern des Coresets wird erreicht, dass die Approximation der Punktmenge auch für spätere Iterationen genau genug ist.

Wir geben Implementierungsresultate an, in denen wir unseren Algorithmus (genannt *CoreMeans*) mit verschiedenen Implementierungen von David Mount vergleichen, welche als äußerst schnell anerkannt sind. Als Testinstanzen dienen große digitale Fotos (bei denen jeder Pixel einen Punkt im dreidimensionalen Farbraum repräsentiert) und künstlich erzeugte normalverteilte Instanzen. In allen Experimenten erreicht unser CoreMeans-Algorithmus gute Clusteringwerte deutlich schneller als die Vergleichsalgorithmen [FS06].

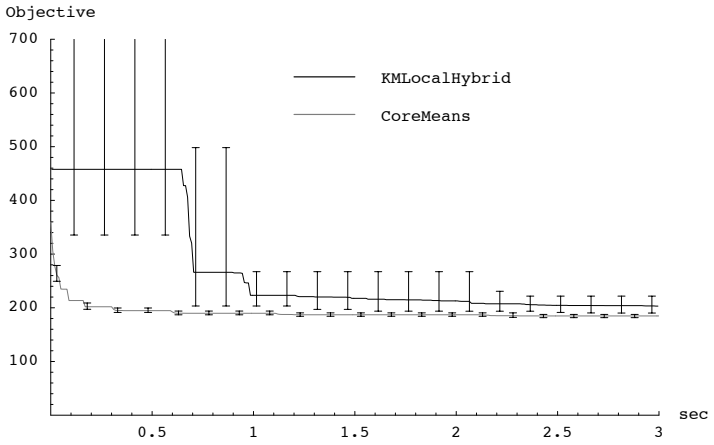

Abbildung 1: Vergleich für das Bild „PaSCo“ mit $k = 50$ centern. Genauere Resultate in Dissertation

6 Zählen von Teilgraphen in Datenströmen

Am Schluss der Dissertation beschäftigen wir uns mit dem Problem, die Anzahl bestimmter Teilgraphen in großen Graphen zu zählen. Sie gibt häufig Aufschluss über die Struktur des Graphen. Durch ein Zählen der Teilgraphen in großen Graphen wie dem Webgraph

können durch Vergleiche mit theoretischen Vorhersagen verschiedene Modelle bewertet werden, welche die Konstruktion des Webgraphen beschreiben. Das Zählen von Teilgraphen gestaltet sich schwierig, weil traditionelle Algorithmen den gesamten Graphen im Speicher halten müssen und oft auf Graphen mit n Knoten $\Omega(n^2)$ Zeit verbrauchen. Dies ist auf großen Graphen wie dem Webgraph nicht anwendbar. Wir stellen daher neue Algorithmen vor, die die Anzahl von Teilgraphen (vollständige Cliquen wie Dreiecke, bipartite Cliquen) approximativ schätzt [BFL$^+$06]. Die Algorithmen arbeiten auf Datenströmen von Kanten und sind daher auch für äußerst große Graphen anwendbar.

Literatur

[BFL$^+$06] Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela und Christian Sohler. Counting triangles in data streams. In *Proc. of the 25th ACM Symposium on Principles of database systems (PODS)*, Seiten 253–262, 2006.

[FIS05] Gereon Frahling, Piotr Indyk und Christian Sohler. Sampling in dynamic data streams and applications. In *To appear in International Journal of Computational Geometry an Applications (IJCGA) (invited Paper). Preliminary version also in Proc. 21st ACM Symposium on Computational Geometry (SoCG 2005)*, Seiten 142–149, 2005.

[Fra06] Gereon Frahling. *Algorithms for Dynamic Geometric Data Streams*. Dissertation, Universität Paderborn, 2006.

[FS05] Gereon Frahling und Christian Sohler. Coresets in dynamic geometric data streams. In *Proc. of the 37th annual ACM symposium on Theory of computing (STOC)*, Seiten 209–217, 2005.

[FS06] Gereon Frahling und Christian Sohler. A fast k-means implementation using coresets. In *To appear in International Journal of Computational Geometry an Applications (IJCGA) (invited Paper). Preliminary version also in Proc. 22nd ACM Symposium on Computational Geometry (SoCG 2006)*, Seiten 135–143, 2006.

Gereon Frahling wurde am 8. Oktober 1976 in Köln geboren. Nach dem Abitur 1996 studierte er von 1997 bis 2002 Mathematik an der Universität Köln, wo er 2002 seinen Abschluss als Diplom-Mathematiker erhielt. Von 1998-2001 war er zudem bei der Denkwerk GmbH hauptverantwortlich für die Entwicklung des Internet-Dienstes „Oneview“, der 1999 den deutschen Multimediaaward in Silber und die „Silver World Medal“ bei den „New York Festivals Interactive Multimedia“ gewann. Nach einem Jahr als wissenschaftlicher Mitarbeiter an der Universität Freiburg erhielt er 2003 ein Stipendium des PaSCo Graduiertenkollegs an der Universität Paderborn. 2006 promovierte Gereon Frahling mit dem Thema „Algorithms for Dynamic Geometric Data Streams“ und arbeitet seitdem als Post-Doc in der Forschungsabteilung von Google Inc. in New York. Seine Forschungstätigkeiten konzentrieren sich auf die effiziente Behandlung sehr großer Datenmengen. Die Ergebnisse seiner Dissertation und Folgeergebnisse haben direkte Anwendungen in der Analyse großer Mengen von Webseiten und Nachrichtenartikeln.