Dynamische Graphalgorithmen and Graphsparsifikation: Neue Techniken und Zusammenhänge[1]

Gramoz Goranci[2]

Abstract: Ein dynamischer Graphalgorithmus ist eine Datenstruktur, die einen Graphen speichert, der sich über die Zeit ändert, und die die Lösung für ein zugrunde liegendes Graphproblem nach jeder Änderung effizient aktualisiert. Ein Graph-Sparsifier ist eine komprimierte Darstellung eines großen Eingabegraphen, bei dem einige Grapheigenschaften (annähernd) erhalten bleiben. In dieser Dissertation entwickeln wir neue Techniken sowohl für dynamische Graphalgorithmen als auch für Sparsifikationsalgorithmen. Wir fokussieren uns auf verschiedene graphbasierte Optimierungsprobleme, die in der spektralen Graphentheorie, der Graphpartitionierung und bei metrischen Einbettungen auftreten. Unsere dynamische Algorithmen haben schnellere Laufzeiten als vorherige Ergebnisse. Unsere Sparsifier-Konstruktionen erzeugen kleinere Sparsifier und verbessern gleichzeitig ihre Approximationsqualität. Wir führen zudem neuartige Reduktionstechniken ein, die unerwartete Zusammenhänge zwischen dynamischen Graphalgorithmen und sparsifizierten Graphen aufdecken.

1 Einführung

Die zunehmende Digitalisierung, sowie neue Entwicklungen wie das Internet of Things sorgen jährlich für einen enormen Anstieg des Datenvolumens. Ein beträchtlicher Teil dieser Daten kann mithilfe von *Graphen* modelliert werden. Ein Graph ist eine Menge von Knoten und Kanten, wobei jede Kante ein Knotenpaar verbindet. Graphen sind in der Mathematik und der Informatik allgegenwärtige Strukturen und treten in verschiedenen Zusammenhängen der realen Welt auf, z. B. in sozialen Netzwerken wie Facebook oder Twitter, das Web-Netzwerk und in Telekommunikationsnetzwerken.

Im Vergleich zu anderen Datendarstellungen sind Graphen besonders geeignet, da ihre Visualisierung häufig Möglichkeiten bietet, interessante Muster in den Daten zu identifizieren, z. B. das Erkennen von Communities in sozialen Netzwerken. Graphdarstellungen sind auch nützlich, weil Algorithmen zu ihrer Manipulation und Speicherung seit den Anfängen des Informatik gründlich untersucht wurden. Dennoch arbeitet eine große Anzahl dieser Graphalgorithmen unter der Annahme, dass Graphen statisch sind, d. h. dass sie sich nicht ändern und dass sie im Speicher eines einzelnen Computers gespeichert werden können. Leider sind diese Annahmen für Graphen aus vielen wichtigen realen Szenarien nicht zutreffend.

Betrachten wir etwa einen Kartengraphen, in dem jeder Knoten einer Stadt entspricht und eine Kante zwischen zwei Knoten die Route darstellt, die sie verbindet. Dieser Kartengraph

[1] Englischer Titel der Dissertation:"Dynamic Graph Algorithms and Graph Sparsification: New Techniques and Connections" [Go19]

[2] Department of Computer Science, University of Toronto, ggoranci@cs.toronto.edu

enthält außerdem die Länge der Routen, indem die Kanten mit der Distanz zwischen den Städten gewichtet ist. Eine grundlegende Frage beim Entwurf von Algorithmen besteht darin, die metrische Struktur des Kartengraphen zu verstehen. Ein konkretes Problem besteht darin, die kürzesten Wege zwischen allen Knotenpaaren im Kartengraphen berechnen. Diese Aufgabe wird von vielen klassischen Algorithmen gelöst, und die Laufzeitkomplexität dieses Problems ist in der Anzahl der Knoten kubisch. In der Realität sind Kartengraphen allerdings nicht statisch, sondern ändern sich mit der Zeit. Beispielsweise kann es aufgrund von Bauarbeiten vorkommen, dass eine Straße, die zwei Städte verbindet, gesperrt wird, was wiederum bedeutet, dass die Kante, die diese Städte verbindet, aus dem Kartengraph gelöscht wird. Das Löschen einer solchen Kante kann sich auf den kürzesten Weg zwischen Städten auswirken, was bedeutet, dass die alte Lösung für den neuen Kartengraph falsch ist. Eine naheliegende Möglichkeit die Lösung zu korrigieren, besteht darin, die kürzesten Wege im veränderten Graphen neu zu berechnen. Diese Lösung hat jedoch einen hohen Rechenaufwand, der für kleine Geräte mit begrenzten Ressourcen, wie z. B. Navigationssysteme, nicht durchführbar ist. Daher ergeben sich folgende Fragen: Können wir Algorithmen entwerfen, die die Lösung nicht jedes Mal komplett neu berechnen müssen, wenn sich der Graph ändert? Wenn ja, was ist die bestmögliche Laufzeit, die wir erreichen können? Müssen wir eine etwas schlechtere Lösungsqualität akzeptieren, um eine bessere Laufzeit zu erzielen?

Andere Herausforderungen beim Umgang mit großen Graphen sind Rechen- und Speicherressourcen. Dies liegt daran, dass die Größe eines Graphen (bei sehr dichten Graphen) quadratisch zur Anzahl der Knoten anwächst. Ein traditioneller Ansatz zur Behebung dieses Problems bestand darin, große Graphen in kleinere zu *komprimieren* und dabei die gewünschten Eigenschaften beizubehalten. Diese komprimierten Versionen von Graphen sind besonders beliebt, da damit Berechnungen für den ursprünglichen Graph auf dem komprimierten Graph ausgeführt werden können, was zu erheblichen Einsparungen bei den Rechen- und Speicherressourcen führt. Die Komprimierung von Graphen wird üblicherweise aus zwei Perspektiven untersucht: (1) Verringerung der Anzahl von Kanten eines Graphen, (2) Verringerung der Anzahl von Knoten. Während der erste Ansatz erfolgreich eingesetzt wurde, um die Laufzeit vieler grundlegender Graphprobleme zu verbessern, ist seine praktische Anwendbarkeit eingeschränkt, da die meisten großen Netzwerke bereits nicht sehr dicht sind und daher wenig Kanten gelöscht werden können. Infolgedessen haben Komprimierungswerkzeuge, die die Anzahl der Knoten reduzieren, in den letzten zehn Jahren immer mehr Beachtung gefunden. Zur Veranschaulichung kehren wir zum Beispiel des Kartengraphen zurück. Nehmen wir an, dass wir unter allen Städten im Graph nur an einer kleinen Teilmenge von Knoten, die für uns wichtig sind, interessiert sind. Dies ist in vielen praktischen Szenarien relevant, z. B. möchte man Entfernungsinformationen nur zwischen großen Städten erhalten, während die kleinen ignoriert werden können. Daher ergeben sich folgende Fragen: Können wir den Kartengraphen zu einem Graphen komprimieren, welcher nur die großen Städte enthält, während die Entfernungen (approximativ) beibehalten werden? Wie gut approximieren die Distanzen im kleinen Graphen die echten Distanzen? Was ist der Kompromiss zwischen der Approximationsqualität und der Größe des komprimierten Graphen?

In dieser Dissertation beschäftigen wir uns mit allen oben genannten Fragen. Wir präsentieren beweisbare algorithmische Werkzeuge sowohl aus dynamischer als auch aus Kompressionsperspektive für verschiedene graphbasierte Optimierungsprobleme, die in der spektralen Graphentheorie, der Graphpartitionierung und bei metrischen Einbettungen auftreten. Darüber hinaus führen wir neuartige Reduktionstechniken ein, die unerwartete Zusammenhänge zwischen sich zeitlich verändernden Graphen und der Graphkomprimierung aufdecken. Im Folgenden werden zunächst die Ergebnisse für dynamische Graphalgorithmen vorstellen und anschließend unsere Beiträge im Bereich der Graphsparsifikation diskutieren.

2 Dynamische Graphalgorithmen

Sei $G = (V, E)$ ein Graph mit n Knoten und m Kanten und $\mathscr{P}$ eine Eigenschaft von G (beispielweise könnte $\mathscr{P}$ der Durchmesser von G sein). Angenommen, G ändert sich durch eine Kanteneinfügung oder -löschung. Können wir die Eigenschaft $\mathscr{P}$ im geänderten Graphen effizient aktualisieren, anstatt sie von Grund auf neu zu berechnen? Diese grundlegende Frage wird seit Jahrzehnten für viele wichtige Grapheneigenschaften gestellt, und der Bereich, der diese Fragen untersucht, heißt *dynamische Graphalgorithmen*. Konkret ist ein dynamischer Graphalgorithmus eine Datenstruktur, die die folgenden Operationen auf einem gegebenen Eingabegraphen G ausführt: (a) INSERT(u,v): füge die Kante (u,v) zu G hinzu, (b) DELETE(u,v): lösche die Kante (u,v) aus G und (c) QUERY$(\mathscr{P})$: frage die Eigenschaft $\mathscr{P}$ ab.

In einigen Varianten dynamischer Graphgorithmen wird die Query-Operation möglicherweise nicht unterstützt, und hier besteht das Ziel darin, zu jedem Zeitpunkt einfach eine korrekte Eigenschaft $\mathscr{P}$ in Bezug auf den aktuellen Graphen zu verwalten. Ein dynamischer Algorithmus hat folgende wichtige Laufzeit-Parameter: (1) *Update-Zeit*, die benötigte Zeit für die Operationen INSERT(u,v) und DELETE(u,v) und (2) *Query-Zeit*, die benötigte Zeit für die Operation QUERY$(\mathscr{P})$. Update- und Query-Zeiten können entweder *worst-case* sein, d. h. die Zeit, die für die Verarbeitung jedes Updates oder Query maximal einzeln aufgewendet wird, oder *amortisiert*, d. h., die durchschnittliche Laufzeit, die über eine Sequenz von Updates amortisiert wird.

Abhängig von den Arten der Update-Operationen werden dynamische Algorithmen in drei Hauptkategorien eingeteilt: (i) *voll-dynamisch*, wenn Updates sowohl aus Einfügungen als auch aus Löschungen bestehen; (ii) *inkrementell*, wenn Updates nur aus Einfügungen bestehen; und (iii) *dekrementell*, wenn Updates nur aus Löschungen bestehen. Bei der Untersuchung der Update-Zeiten in Algorithmen vom Typ (ii) und (iii) ist es üblich, die *Gesamtlaufzeit* zu berücksichtigen, d. h., die Zeit, die für eine Sequenz von $\Theta(m)$ Einfügungen oder Löschungen aufgewendet wird. Dynamische Algorithmen können entweder *deterministisch* oder *randomisiert* sein, und häufig kann man durch Randomisierung Algorithmen mit besseren Laufzeiten erreichen.

Insbesondere in den letzten zwei Jahrzehnten wurden große Fortschritte bei der Entwicklung effizienter dynamischer Graphalgorithmen erzielt. Zu den berücksichtigten Grapheigenschaften zählen Konnektivität, Erreichbarkeit, kürzeste Wege, (globaler) Minimalschnitt,

minimaler Spannbaum, Spanners, Schnitt- und Spektralsparsifier. Trotz all dieser Fortschritte gibt es jedoch weiterhin viele wichtige Probleme, deren dynamische Komplexität nur unzureichend verstanden wird. Zum Beispiel gibt keine nicht-trivialen dynamischen Algorithmen für Probleme wie Graphpartitionierung oder zur Lösung von strukturierten linearen Gleichungssystemen, die zuletzt viel Aufmerksamkeit in verschiedenen Bereichen der Informatik erhalten haben. Aus diesem Grund untersuchen wir in dieser Dissertation dynamische Algorithmen für einige dieser Probleme und zeigen, dass ihre dynamische Komplexität nicht-trivial verbessert werden kann. Wir machen zudem Fortschritte bei grundlegenden Graphproblemen, indem wir ihre seit langem bekannten Laufzeitgarantien verbessern.

Dynamische Algorithmen für spektralbasierte Grapheneigenschaften. Wir untersuchen dynamische Algorithmen zur Verwaltung von Lösungen für Laplace-Systeme und effektive Widerstände. Laplace-Systeme sind eine wichtige Unterklasse von linearen Gleichungssystemen, die in vielen natürlichen Kontexten auftreten, etwa im maschinellen Lernen, in der Computergrafik und in der Bildverarbeitung. Das Lösen von Laplace-Systemen hat nach der mit dem Gödel-Preis ausgezeichneten Arbeit von Spielman und Teng [ST04], die den ersten approximativen Löser mit annähernd linearer Laufzeit entwickelt haben, erhebliche Aufmerksamkeit erhalten.

Bei einem ungerichteten, ungewichteten Graphen $G = (V,E)$ mit n Knoten und m Kanten bezeichnen $\mathbf{L} := \mathbf{D} - \mathbf{A}$ die *Laplace-Matrix* von G, wobei $\mathbf{D}$ und $\mathbf{A}$ die zugehörigen Grad- und die Adjazenzmatrizen von G sind. Die Laplace-Matrix $\mathbf{L}$ bildet zusammen mit einem Vektor $\mathbf{b} \in \mathbb{R}^n$ ein lineares Gleichungssystem $\mathbf{Lx} = \mathbf{b}$, das als *Laplace-System* bezeichnet wird. Sei $\mathbf{L}^\dagger$ die Pseudoinverse von $\mathbf{L}$. Ein *Lösungsvektor* $\tilde{\mathbf{x}} \in \mathbb{R}^n$ ist *ε-approximativ*, wenn er $\left\|\tilde{\mathbf{x}} - \mathbf{L}^\dagger\mathbf{b}\right\|_{\mathbf{L}} \leq \varepsilon \left\|\mathbf{L}^\dagger\mathbf{b}\right\|_{\mathbf{L}}$ erfüllt, wobei $\|\mathbf{x}\|_{\mathbf{L}} := \sqrt{\mathbf{x}^\top\mathbf{Lx}}$ und $\varepsilon > 0$ ist ein Fehlerparameter. Sei $\mathbf{1}_u \in \mathbb{R}^n$ der Indikatorvektor eines Konten u, so dass $\mathbf{1}_u(v) = 1$, wenn $v = u$ und 0 andernfalls.

Wir führen ein dynamisches Modell zum Lösen von Laplace-Systemen ein, das folgende Operationen zulässt: (1) Das Einfügen und Löschen von Kanten im zugrunde liegenden Graphen G (dies entspricht dem Ändern von Einträgen von $\mathbf{L}$, die sich nicht auf der Diagonalen befinden), (2) das Ändern des Vektors $\mathbf{b}$ und (3) Zugriff auf die Koordinaten eines ε-approximativen Lösungsvektors $\tilde{\mathbf{x}}$. Für ungerichtete, ungewichtete Graphen mit beschränktem Grad zeigen wir den ersten nicht-trivialen dynamischen Algorithmus, welcher eine erwartete amortisierte Laufzeit[3] von $\tilde{O}(n^{11/12}\varepsilon^{-5})$ erreicht und damit schneller ist als der $\tilde{O}(n)$-Zeitalgorithmus, der nach jedem Update einen Lösungsvektor von Grund auf neu berechnet. Wenn der Vektor $\mathbf{b}$ nur wenige Nicht-Null-Einträge enthält, erstreckt sich unser Ergebnis auf allgemeine Graphen, wobei eine verbesserte Laufzeit von $\tilde{O}(m^{3/4}\varepsilon^{-5})$ erreicht wird.

Ein spektrales Objekt im Zusammenhang mit Laplace-Systemen ist der *effektive Widerstand*. Die Berechnung von effektiven Widerständen führte zuletzt zu schnelleren Algorithmen zur Berechnung von maximalen Flüssen und gewinnt zunehmen an Popularität in der theoretischen Informatik. Die algebraische Definition des effektiven Widerstands $R^G_{\text{eff}}(u,v)$

[3] Mit der $\tilde{O}(\cdot)$-Notation unterdrücken wir der Übersichtlichkeit halber polylogarithmische Faktoren.

zwischen einem Knotenpaar u und v in einem Graphen G ist gegeben durch

$$R_{\text{eff}}^{G}(u,v) := (\mathbf{1}_u - \mathbf{1}_v)^{\top}\mathbf{L}^{\dagger}(\mathbf{1}_u - \mathbf{1}_v).$$

Physikalisch gesehen, d. h. wenn G als Widerstandsnetzwerk betrachtet wird, ist der effektive Widerstand zwischen u und v die Energie des Flusses, wenn eine Stromeinheit von u nach v in G gesendet wird.

Wir untersuchen voll-dynamische Algorithmen, die den effektiven Widerstand zwischen allen Knotenpaaren verwalten. Wir erhalten einen Algorithmus, der Updates und Querys in $\tilde{O}\left(\min\{m^{3/4}, n^{5/6}\}\varepsilon^{-4}\right)$ Zeit durchführt und effektive Widerstände bis auf einen $(1+\varepsilon)$-Faktor approximiert. Zudem erweitern wir das obige Ergebnis auf zwei Arten: Erstens wird der Algorithmus auf *gewichtete*, ungerichtete Graphen verallgemeinert mit einer Laufzeit von $\tilde{O}(n^{5/6}\varepsilon^{-4})$ für die erwartete amortisierte Update- und Query-Zeit. Wenn zweitens zudem angenommen wird, dass die gewichteten Graphen *kleine* Knoten-Separatoren zulassen (z. B. planare Graphen), verbessert sich unsere *worst-case* Laufzeitgarantie auf $\tilde{O}(\sqrt{n}\varepsilon^{-2})$. Unser Ergebnis für allgemeine Graphen steht in starkem Kontrast zu eng verwandten Grapheigenschaften wie dem kürzesten Weg, für den (bedingte) untere Schranken derartige Laufzeitverbesserungen im voll-dynamischen Setting ausschließen, und dem maximalen Fluss, dessen dynamische Komplexität bisher nur unzureichend verstanden ist.

Die Grundidee unserer Datenstrukturen ist es, dynamisch eine Approximation des Schur-Komplements zu verwalten. Ein *Schur-Komplement* (auch *spektraler Knotensparsifier* genannt) H eines Graphen G in Bezug auf $K \subseteq V(G)$ ist ein Graph mit $V(H) = K$, wobei die effektiven Widerstände zwischen jedem Paar von Knoten in K, die gleichen sind wie ihre effektiven Widerstände in G. Abhängig von der Familie der Graphen, an denen wir interessiert sind, gibt es geringfügige Unterschiede in der Art und Weise, wie wir diesen Sparsifier nutzen. Bei allgemeinen Graphen beruhen unsere Techniken entscheidend darauf, dass das Schur-Komplement als Vereinigung über Random Walks mit Endpunkten in K betrachtet werden kann. Dies ermöglicht es uns, Knoten aus dem Originalgraphen zufällig auszuwählen und dann ein Schur-Komplement in Bezug auf diese zufällig gewählte Knotenmenge zu konstruieren. Das Subsampling stellt sicher, dass die Random Walks kurz sind, und dies ermöglicht es uns, sie unter Verwendung elementarer Datenstrukturen effizient zu samplen und zu verwalten. Für planare Graphen nutzen wir, dass sie Knoten-Separatoren sublinearer Größe haben (daher die Abhängigkeit von $\sqrt{n}$ von der Laufzeit) und dass approximative Schur-Komplemente in nahezu linearer Zeit berechnet werden können [KS16]. Inspiriert von der Arbeit von Lipton, Rose und Tarjan [LRT79] zu *nested dissection*, werden diese beiden Bestandteile dann zusammengeführt, um die Schur-Komplement für diese Graphenfamilie dynamisch zu verwalten.

Alle Resultate, die wir oben präsentiert haben, garantieren $(1+\varepsilon)$-approximative effektive Widerstände. Wir zeigen, dass es nicht möglich ist, exakte effektive Widerstände mit Update-Zeiten von $O(n^{1-\delta})$ und einer Query-Zeit von $O(n^{2-\delta})$ für $\delta > 0$ zu erhalten, außer eine populäre Hypothese der Komplexitätstheorie ist falsch.

Dynamische Low-Stretch Bäume. Wir untersuchen dynamische Algorithmen für Low-Stretch Spannbäume und Spanners. Bäume gehören zu den einfachsten Graphklassen und sind aus algorithmischer Sicht besonders praktisch, da viele Probleme einfache Lösungen

für Bauminstanzen zulassen. Ein gängiger Ansatz zur Ausnutzung dieses Verhaltens von Bäumen besteht darin, allgemeine Graphen als Bäume zu approximieren und dabei die relevanten Grapheigenschaften beizubehalten. Der Begriff des Low-Stretch Spannbaums ist ein prominentes Beispiel für eine solche Graphenapproximation, die bei der Lösung symmetrischer, diagonal dominanter (SDD) linearer Systeme eine zentrale Rolle gespielt hat. Für einem ungewichteten, ungerichteten Graphen $G = (V,E)$ und einem Spannbaum T von G ist der *Stretch* einer Kante $(u,v) \in E$ in Bezug auf T gegeben durch $\text{stretch}_T(u,v) := \text{dist}_T(u,v)$, wobei $\text{dist}_G(u,v)$ die Länge des kürzesten Weges zwischen u und v in G bezeichnet. Das *durchschnittliche Stretch* über alle Kanten von G in Bezug auf T ist $\text{avg-stretch}_T(G) := \frac{1}{|E|}\sum_{(u,v)\in E}\text{stretch}_T(u,v)$. Ein *Low-Stretch* Spannbaum ist ein Baum mit einem subpolynomialen oder polylogarithmischen durchschnittlichen Stretch in der Anzahl der Knoten n.

Motiviert durch die grundlegende Bedeutung von Low-Stretch Spannbäume sowie deren wichtigen Anwendungen betrachten wir dieses Objekt aus dynamischer Sicht. Wir zeigen, dass es für ungewichtete, ungerichtete Graphen einen voll-dynamischen Algorithmus gibt, der einen Spannbaum mit $n^{o(1)}$ erwartetem durchschnittlichen Stretch und $n^{1/2+o(1)}$ amortisierter Update-Zeit verwaltet. Unser Ergebnis beantwortet eine offene Frage von Baswana et al. [BKS12] und kann erweitert werden, um den durchschnittliche Stretch $O(t)$ und die Update-Zeit $n^{1+o(1)}/t$ zu erreichen, wobei $t \geq \sqrt{n}$. Dies zeigt, dass die $\sqrt{n}$ -Barriere in der Laufzeit nicht inhärent ist, zumindest wenn ein sehr großer Strech tolerierbar ist. Eine der wesentlichen Ideen unseres Algorithmus besteht darin, den Graphen in Cluster mit kleinem Durchmesser zu zerlegen und diese Zerlegung dynamisch verwaltet. Dies wird mit dem Random-Shift-Clustering von Miller, Peng und Xu [MPX13] zusammen mit vielen Anpassungen implementiert, damit es im dynamischen Setting funktioniert. Wir verwenden dann eine dynamische Version der Hierarchie von Clustern mit geringem Durchmesser von Alon, Karp, Peleg und West [Al95], was wiederum einen anspruchsvollen Amortisierungsansatz erfordert, um die Verbreitung von Aktualisierungen innerhalb der Hierarchie zu steuern. Zusätzlich verwendet unser Algorithmus *Sparsifier für dynamische Schnitte*, um das Problem auf wenig dichte Graphen zu reduzieren.

Eine direkte Konsequenz unseres dynamisches Random-Shift-Clusterings ist ein verbesserter dynamischer Algorithmus für Spanners. Ein *k-Spanner H* eines Graphen G ist ein *Subgraph* von G, in dem für alle Knoten u und v gilt, dass $\text{dist}_H(u,v) \leq k \cdot \text{dist}_G(u,v)$. Für $t \geq 1$ und jeden ungewichteten, ungerichteten Graphen G entwickeln wir einen voll-dynamischen Algorithmus, der einen $(2t-1)$-Spanner mit $O(n^{1+1/t}\log n)$ Kanten und $O(t\log^2 n)$ amortisierter Update-Zeit verwaltet. Dieses Ergebnis verbessert den Algorithmus von Baswana et al. [BKS12] um Faktor t in der Größe des Spanners und der Update-Zeit. Die Verbesserung ist besonders relevant, da der Trade-Off zwischen Stretch und Graphgröße von $2t-1$ zu $O(n^{1+1/t})$ unter der bekannten Erdös-Girth-Conjecture bereits bestmöglich ist.

Dynamische Graphpartitionierung. Wir untersuchen inkrementelle Algorithmen für den globalen minimalen Schnitt und minimalen Schnitt zwischen Knoten s und t, die beide grundlegende Probleme bei der Graphpartitionierung darstellen. Bei Problemen der Graphpartitionierung wird der Eingabegraph in kleinere Komponenten unterteilt, während die Anzahl der Kanten zwischen diesen Komponenten minimiert wird. Diese Probleme

haben in der Vergangenheit einen zentralen Platz beim Verständnis von Netzwerkflüssen, Paketrouting und VLSI-Layout eingenommen. Sie wurden auch in vielen Divide-and-Conquer-Ansätzen zur Lösung von graphbasierten Clustering-Problemen eingesetzt.

Bei einem ungewichteten, ungerichteten Graphen $G = (V,E)$ ist ein *Schnitt* von G eine Partition von V in zwei nicht-leere Teilmengen A und B. Wenn s und t als verschiedene Knoten in G gegeben sind, ist ein *s-t-Schnitt* von G ein Schnitt (A,B), so dass $s \in A$ und $t \in B$. Für einen Schnitt (A,B) in G ist die *Größe* von (A,B) die Anzahl der Kanten mit einem Endpunkt in A und dem anderen in B. Ein *globaler minimaler Schnitt*, der mit λ_G bezeichnet wird, ist ein Schnitt von minimaler Größe. Ein *s-t-minimaler Schnitt*, bezeichnet mit $\lambda_G(s,t)$, ist ein s-t-Schnitt mit einer minimaler Größe.

Wir entwickeln einen *exakten, deterministischen* inkrementellen Algorithmus, der einen globalen minimalen Schnitt λ_G in $O(\log^3 n \log\log n)$ amortisierter Update-Zeit und $O(1)$ Query-Zeit verwaltet. Unser Ergebnis erzielt eine *exponentielle* Beschleunigung der Update-Zeit des bisher besten bekannten Algorithmus von Henzinger [He97] und beantwortet teilweise eine offene Frage von Thorup [Th07]. Darüber hinaus stehen unsere Laufzeitschranken in starkem Kontrast zu einer bedingten Untergrenze für die dynamische Verwaltung von $\lambda(G)$ in *gewichteten* Graphen, die zeigt, dass es keine Algorithmen gibt, die gleichzeitig eine sublineare Update- und Query-Zeit erreichen können. Die Hauptidee unseres Algorithmus besteht darin, eine Sparsifikationsroutine von Kawarabayshi und Thorup [KT19] und den exakten inkrementellen Algorithmus von Henzinger [He97] zu kombinieren. Die Kombination ist allerdings nicht-trivial, da sie Subroutinen beider Arbeiten deutlich erweitert und diese Erweiterungen zur Erlangung unserer Garantien erfordert.

Obwohl das s-t-Minimalschnitt-Problem eines der zentralen Probleme in der algorithmischen Graphentheorie ist, ist nur sehr wenig über dessen dynamische Komplexität für allgemeine Graphen bekannt. Wir machen die ersten positiven Fortschritte, um dieses Problem zu verstehen, indem wir einen inkrementellen Algorithmus mit $\tilde{O}(n^{2/(\ell+1)})$ Update- und Query-Zeit entwerfen, der eine $O(\log^{4\ell} n)$ Approximation von $\lambda_G(s,t)$ für alle $s,t \in V(G)$ erreicht, wobei $\ell \geq 1$. Unser technischer Beitrag ist ein neue Art von Sparsifiern, genannt *lokale Sparsifier*. Diese Sparsifiern sind eine stärkere Version der gut erforschten Knotensparsifier. Ein *Knotensparsifier* H eines Graphen G in Bezug auf eine Menge von *Terminals* $K \subseteq V(G)$ ist ein Graph mit $V(H) \supseteq K$, in dem (i) $|V(H)|$ "klein" ist und (ii) H eine Grapheigenschaft $\mathscr{P}$ bewahrt, die auf die Terminals K in G beschränkt ist (siehe nächste Abschnitt für eine eingehende Behandlung der Knotensparsifiern). Ein *lokaler Sparsifier* ist eine Datenstruktur, die Knotensparsifier generalisert. Bei einem gegebenen Graphen $G = (V,E)$ besteht das Ziel darin, eine Datenstruktur zu erstellen, die die folgenden Operationen unterstützt: (1) PREPROCESS(G): verarbeite den Graphen G vor, (2) QUERYSPARSIFIER(G,K): Berechne einen Knotensparsifier H von G, der eine Eigenschaft $\mathscr{P}$ in Bezug auf K beibehält, und gebe diesen aus. Lokale Sparsifier es uns ermöglichen also, Knotensparsifier für eine beliebige Menge von Terminals in K zu extrahieren. In Operation (2) ist eine sehr wichtige Anforderung, dass K Teil der Eingabe ist, da es $\Theta(2^n)$ verschiedene Terminalmengen K gibt.

Wir zeigen, dass eine Variante von *tree cut sparsifiers* von Räcke, Shah und Täubig [RST14] verwendet werden kann, um einen lokalen Sparsifier zu konstruieren, der die Schnittstruktur

des Graphen bis zu polylogarithmische Faktoren bewahrt und $\tilde{O}(m)$ Vorverarbeitungszeit und $\tilde{O}(|K|)$ Query-Zeit erreicht. Wir entwickeln darauf basierend ein Reduktions-Framework, das effiziente lokale Sparsifier in einen inkrementellen Algorithmus umwandelt, was wiederum eine polylogarithmische Approximation des *s-t*-Minimalschnitts impliziert. Zudem impliziert unsere Technik einen inkrementellen Approximationsalgorithmus für das *uniform sparest cut*-Problem mit ähnlichen Garantien.

3 Graphsparsifikation

Ein *Graph Sparsifier* ist eine "komprimierte" Version eines großen Eingabegraphen, der Eigenschaften wie Entfernungs- oder Erreichbarkeitsinformationen, Schnittwert oder das Spektrum des Graphen beibehält. Herkömmlicherweise wurden Graph Sparsifier aus zwei Perspektiven untersucht: (1) *Kantensparsifier*, die die Anzahl der Kanten eines Graphen verringern, und (2) *Knotensparsifier*, die die Anzahl der Knoten verringern. Knotensparsifier wurden erfolgreich angewendet, um die Laufzeit vieler grundlegender graphbasierter Optimierungsprobleme zu verbessern. Die bemerkenswertesten Beispiele sind transitive Reduktionen, Spanners sowie Schnitt- und spektrale Sparsifier.

In dieser Arbeit konzentrieren wir uns auf Knotensparsifier. Ausgehend von einem Graphen $G = (V,E)$ und einer Menge von Knoten $K \subseteq V$, die als *Terminals* bezeichnet werden, besteht das Ziel darin, einen Graphen $H = (V',E')$ zu konstruieren, der die folgenden Eigenschaften erfüllt (i) $V' \supseteq K$ und $|V'|$ ist "klein", idealerweise $|V'| = O(\text{poly}(|K|))$ und (ii) H bewahrt Eigenschaften wie Erreichbarkeit, Distanz oder Schnitte, die zwischen Terminals in K definiert sind (approximativ). Oft ist es wünschenswert, dass H strukturell ähnlich zu G ist, z. B. wenn G planar ist, soll H ebenfalls planar sein. Wenn H eine Eigenschaft approximativ beibehält, wird das Approximationsverhältnis als *Qualität* des Sparsifiers bezeichnet. Dieses Komprimierungswerkzeug ist aus algorithmischer Sicht sehr nützlich: sobald H berechnet ist, kann man algorithmische Aufgaben auch in H anstelle von G ausführen, was wiederum zu Einsparungen bei den Rechen- und Speicherressourcen führt. Neben ihrer praktischen Relevanz haben Knotensparsifier auch Anwendungen in anderen Teilbereichen der theoretischen Informatik gefunden, nämlich in Approximationsalgorithmen und im Netzwerk-Routing.

Im Folgenden werden wir unsere Beiträge zu Knotensparsifiern diskutieren, die strukturell einem Eingabegraphen ähneln und gleichzeitig Distanzen in ungerichteten Graphen oder Informationen zur Erreichbarkeit in gerichteten Graphen beibehalten.

Distance Approximation Minors. Wir untersuchen Knotensparsifier, die Minoren des Eingabegraphen sind und die die Distanzen zwischen den Terminals approximieren. Ein Minor ist ein Graph, der durch das Löschen von Kanten und Knoten und durch Kantenkontraktion erzeugt wurde. Minoren sind besonders vorteilhaft, da sie strukturelle Eigenschaften des Eingabegraphen bewahren, z. B. ist ein Minor eines planaren Graphen ebenfalls ein planarer Graph. Wenn ein Graph $G = (V,E)$ und Terminals $K \subseteq V$ gegeben sind, ist ein α-*Distance Approximating Minor* von G ein Graph $H = (V',E')$, sodass (i) $V' \supseteq K$, (ii) H ein Minor von G ist, der alle Terminals enthält und bei dem keine Kanten von Terminals kontraktieren

wurden, und (iii) der kürzeste Weg zwischen zwei Knoten der Untermenge K liegt innerhalb eines α-Faktors ihres kürzesten Weges in G. Knoten in $V' \setminus K$ werden als *Steiner-Knoten* bezeichnet.

Krauthgamer, Nguyen und Zondiner [KNZ14] untersuchten *distance preserving minors*, d. h. $\alpha = 1$. Sie zeigten, dass allgemeine Graphen distance preserving minors mit $O(|K|^4)$ Steiner-Knoten besitzen. Es stellt sich natürlicherweise die Frage, in welchem Verhältnis sich Qualität der Approximation und die Anzahl der Steiner-Knoten zueinander befinden. Für diese Frage erhalten wir neue unteren und oberen Schranken. Unsere untere Schranke zeigt, dass es für unendlich viele $k \in \mathbb{N}$ und einen Graphen mit k Terminals gibt, die kein $(\alpha - \varepsilon)$-distance preserving minor mit $\Omega(k^\gamma)$ Steiner-Knoten zulässt, für alle $\varepsilon > 0$, wobei $\alpha \geq 2$ und $\gamma = \gamma(\alpha)$. Zum Beispiel folgt aus unseren Resultaten, dass jeder $(2-\varepsilon)$-distance preserving minor mindestens $\Omega(k^2)$ Knoten besitzen muss.

Um die untere Schranke zu beweisen, führen wir eine neue Black-Box-Reduktionstechnik ein, die die untere Schranken für das auf $V' = K$ beschränkte Distance Approximating Minor-Problem umwandelt, d. h. $|V' \setminus K| = 0$, in superlineare untere Schranken für $|V' \setminus K|$ für Distance Approximating Minors mit der gleichen Qualität. Das Herzstück unserer Graphkonstruktionen sind Varianten von *Steiner Systems*, welche vor allem im kombinatiorischen Design untersucht wurden. Unsere obere Schranke zeigt, dass die Klasse der planaren Graphen $(1+\varepsilon)$-distance approximation minors mit $|V'| = O(|K|^2 \varepsilon^{-2} \log^2 |K|)$ zulässt. Durch eine leichte Qualitätssteigerung kann unser Sparsifier daher die Resultate von Krauthgamer et al. [KNZ14] um einen quadratischen Faktor verbessern. Der Schlüssel zu diesem Ergebnis ist das sogenannte *Terminal Path Cover*. Ein solches Cover ist eine Menge von kürzesten Wegen im Graphen, deren Vereinigung (1) die Terminalmenge enthält und (2) kürzeste Distanzen zwischen Terminals approximiert. Wir zeigen, dass *Distance Oracles* für planare Graphen erweitert werden können, um Terminal Path Covers für planare Graphen zu konstruieren. Nun erhalten wir die gewünschten Garantien, indem wir diese Ideen zusammen mit einem Argument zur Anzahl der Verzweigungen auf den kürzesten Wegen kombinieren.

Reachability Preserving Minors. Die Kantensparsifikation von *gerichteten* Graphen ist für viele Probleme deutlich schwieriger als in ungerichteten Graphen. Etwa kann man für Probleme wie Erreichbarkeit zeigen, dass es für gerichtete Graphen keine nicht-trivialen Kantensparsifier geben kann. Wir komplementieren dieses negative Ergebnis für Kantensparsifier, indem wir positive Resultate für Knotensparsifier präsentieren. In diesem Problem ist die Eingabe ein gerichteter Graph $G = (V, E)$ und eine Menge von Terminals $K \subseteq V$. Ein *Reachability Preserving Minor* von G ein gerichteter Graph $H = (V', E')$, der die folgenden Eigenschaften erfüllt: (i) $V' \supseteq K$, (ii) H ist ein Minor von G, und (iii) für jedes Paar von Knoten $u, v \in K$ gibt es einen gerichteten Weg von u nach v in H, wenn es einen gerichteten Weg von u nach v in G gibt.

Wir initiieren die Erforschung solcher Sparsifier und erhalten die ersten nicht-trivialen Garantien für das Problem. Wir zeigen, dass planare Graphen Reachability Preserving Minors mit $|V'| = O(|K|^2 \log |K|)$ besitzen. Für allgemeinen Graphen sind unsere oberen Schranken nur um einen weiteren Faktor $|K|$ schlechter. Weiterhin beweisen wir neue untere Schranken, die zeigen, dass für unendlich viele $k \in \mathbb{N}$ ein gerichteter planarer Graph G mit

k Terminals existiert, so dass jeder Reachability Preserving Minor $\Omega(k^2)$ Steiner-Knoten verwenden muss. Für planare Graphen sind die untere und die obere Schranke bis auf polylogarithmische Faktoren in k scharf.

Literaturverzeichnis

[Al95] Alon, N.; Karp, R.; Peleg, D.: A Graph-Theoretic Game and Its Application to the k-Server Problem. SIAM J. Computing, 24(1):78–100, 1995.

[BKS12] Baswana, S.; Khurana, S.; Sarkar, S.: Fully dynamic randomized algorithms for graph spanners. ACM Transactions on Algorithms, 8(4):35:1–35:51, 2012.

[Go19] Goranci, G.: Dynamic Graph Algorithms and Graph Sparsification: New Techniques and Connections. Dissertation, University of Vienna, 2019.

[He97] Henzinger, M. R.: A Static 2-Approximation Algorithm for Vertex Connectivity and Incremental Approximation Algorithms for Edge and Vertex Connectivity. J. Algorithms, 24(1):194–220, 1997.

[KNZ14] Krauthgamer, R.; Nguyen, H. L.; Zondiner, T.: Preserving Terminal Distances Using Minors. SIAM J. Discrete Math., 28(1):127–141, 2014.

[KS16] Kyng, R.; Sachdeva, S.: Approximate Gaussian Elimination for Laplacians - Fast, Sparse, and Simple. In: Symposium on Foundations of Computer Science. S. 573–582, 2016.

[KT19] Kawarabayashi, Ken-ichi; Thorup, Mikkel: Deterministic Edge Connectivity in Near-Linear Time. J. ACM, 66(1):4:1–4:50, 2019.

[LRT79] Lipton, R.; Rose, D.; Tarjan, R.: Generalized Nested Dissection. SIAM Journal on Numerical Analysis, 16(2):346–358, 1979.

[MPX13] Miller, G.; Peng, R.; Xu, S. C.: Parallel Graph Decompositions Using Random Shifts. In: Symposium on Parallelism in Algorithms and Architectures. S. 196–203, 2013.

[RST14] Räcke, Harald; Shah, Chintan; Täubig, Hanjo: Computing Cut-Based Hierarchical Decompositions in Almost Linear Time. In: Symposium on Discrete Algorithms. S. 227–238, 2014.

[ST04] Spielman, D.; Teng, S.: Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In: Symposium on Theory of Computing. S. 81–90, 2004.

[Th07] Thorup, M.: Fully-Dynamic Min-Cut. Combinatorica, 27(1):91–127, 2007.

Gramoz Goranci wurde am 27. Juni 1990 in Gjakova geboren. Er studierte von 2008 bis 2011 im Bachelor-Studiengang Mathematik-Informatik an der Universität Prishtina sowie von 2012 bis 2014 im Master-Studiengang Informatik an der Technischen Universität München, wo er eine Abschlussarbeit im Bereich Graphalgorithmen unter der Betreuung von Harald Räcke schrieb. Von 2015 bis 2019 verfasste er seine Doktorarbeit über dynamische Graphalgorithmen und Graphsparsifikation unter der Betreuung von Monika Henzinger an der Universität Wien. Das Herbstsemester 2017 vebrachte er als Gastwissenschaftler am Georgia Institute of Technoloy. Seit Anfang 2020 forscht er als PostDoc an der Universität Toronto.