Schnellere Approximationsalgorithmen zur Partiell-Dynamischen Berechnung Kürzester Wege[1]

Sebastian Krinninger[2]

Abstract: Ein Algorithmus gilt als dynamisch wenn seine Eingabe mit der Zeit immer wieder Änderungen unterworfen ist und er deshalb das Ergebnis seiner Berechnungen regelmäßig aktualisiert. Das Hauptziel ist es, schneller zu sein als ein naiver Algorithmus, der das Ergebnis nach jeder Änderung von Grund auf neu berechnet. Für dynamische Probleme auf Graphen bestehen die Änderungen in der Regel aus Einfügungen und Löschungen von Kanten. In dieser Arbeit konzentrieren wir uns auf partiell-dynamische Algorithmen, die nur eine Art von Änderungen erlauben; entweder ausschließlich Einfügungen (inkrementeller Algorithmus) oder ausschließlich Löschungen (dekrementeller Algorithmus). Wir entwickeln schnellere, partiell-dynamische Approximationsalgorithmen zur Berechnung annähernd kürzester Wege in Graphen in Bezug auf die Gesamtlaufzeit, also die Summe der Laufzeiten, die jeweils benötigt werden, um das Ergebnis nach einer Änderung zu aktualisieren.

Keywords: Kürzeste Wege, dynamische Graphalgorithmen

1 Einführung und Motivation

Die Berechnung kürzester Wege ist ein fundamentales Problem der Informatik. Wird beispielsweise ein Verkehrsnetz durch einen Graph, bestehend aus Knoten und Kanten, modelliert, so lässt sich die schnellste Verbindung von A nach B als kürzester Weg im Graph berechnen. In möglichst realistischen Modellen wird zugelassen, dass sich der Graph mit der Zeit verändert. Typische Veränderungen des Graphen sind Einfügungen und Löschungen von Kanten. In einem Verkehrsnetz könnte beispielsweise ein temporärer Stau dadurch modelliert werden, dass die entsprechende Kante im Graph gelöscht und später wieder eingefügt wird. Die naive Lösung für ein dynamisches Szenario besteht darin, den kürzesten Weg nach jeder Veränderung im Graph von Grund auf neu zu berechnen. Wünschenswert sind jedoch weitaus effizientere Algorithmen, die nur einen Bruchteil dieser Zeit benötigen. Aus dieser Motivation heraus werden mit dynamischen Graphalgorithmen spezielle Datenstrukturen entwickelt, die Lösungen für fundamentale Graphprobleme nach Veränderungen im Graph schnell wieder aktualisieren. Neben der Berechnung kürzester Wege wurden dynamische Algorithmen bisher auch für zahlreiche andere Probleme entwickelt, so zum Beispiel für Konnektivität, minimaler Spannbaum, größtmögliches Matching, transitive Hülle und starke Zusammenhangskomponenten. Das Ziel dieser Dissertation war, schnellere Algorithmen für dynamische kürzeste Wege in Bezug auf beweisbare obere Schranken der Laufzeit zu finden.

[1] Englischer Titel der Dissertation: „Faster Approximation Algorithms for Partially Dynamic Shortest Paths Problems“

[2] Max-Planck-Institut für Informatik, Department 1: Algorithms and Complexity, Campus E1 4, 66123 Saarbrücken

Ein dynamischer Algorithmus für kürzeste Wege ist eine Datenstruktur, die folgende Operationen für alle Paare von Knoten u und v eines sich verändernden Graphen G mit n Knoten und maximal m Kanten bereitstellt:

- `Insert (`u`, `v`)`: Füge die Kante (u,v) zu G hinzu, falls sie noch nicht existiert.
- `Delete (`u`, `v`)`: Lösche die Kante (u,v) aus G, falls sie existiert.
- `Query (`u`, `v`)`: Gib die Distanz $d_G(u,v)$ von u nach v zurück.

Jede Einfügung oder Löschung wird auch *Update* genannt. Nach jedem Update läuft der Algorithmus für eine gewisse Zeit, um auf die Veränderung im Graph reagieren zu können, sodass nachfolgende Distanz-Queries schnell beantwortet werden können. Die Zeit, die der Algorithmus nach jedem Update benötigt, wird *Update-Zeit* genannt. Sehr oft wird die Update-Zeit über eine Sequenz von Updates amortisiert. Die Zeit, die benötigt wird, um ein Query zu beantworten, wird *Query-Zeit* genannt. Im Folgenden berücksichtigen wir nur kleine Query-Zeiten wie $O(1)$ oder $O(\text{polylog}\, n)$. Alle in dieser Arbeit vorgestellten Algorithmen können ohne großen Mehraufwand als Antwort auf ein Query auch einen Pfad der entsprechenden Länge zurückgeben – in Zeit proportional zur Länge des Pfades.

Üblicherweise werden in der Literatur folgende Einschränkungen für dynamische Kürzeste-Wege-Probleme betrachtet:

- Anstatt sowohl Einfügungen als auch Löschungen zu erlauben, werden für *partiell-dynamische* Algorithmen nur Updates eines Typs erlaubt. Werden nur Einfügungen erlaubt, so spricht man von *inkrementellen* Algorithmen. Werden nur Löschungen erlaubt, so spricht man von *dekrementellen* Algorithmen. Werden beide Arten von Updates erlaubt, so spricht man von *voll-dynamischen* Algorithmen.
- Anstatt exakter kürzester Wege, berechnen manche Algorithmen nur eine *Approximation*. Ein Algorithmus liefert eine (α,β)-Approximation (wobei $\alpha \geq 1$ und $\beta \geq 0$), falls er auf ein Query mit einer Distanz-Schätzung $\delta(u,v)$ antwortet, für die gilt: $d_G(u,v) \leq \delta(u,v) \leq \alpha d_G(u,v) + \beta$. Eine rein multiplikative Approximation mit $\beta = 0$ bezeichnen wir üblicherweise als α-Approximation statt als $(\alpha,0)$-Approximation. In dieser Arbeit gilt meist $\alpha = 1+\varepsilon$ für ein ε mit $0 < \varepsilon \leq 1$, wobei wir im Folgenden der Übersichtlichkeit halber annehmen, dass ε eine kleine Konstante ist.
- Anstatt die kürzesten Wege zwischen allen Knoten zu berechnen, werden nur die kürzesten Wege von einem ausgewählten Startknoten zu allen anderen Knoten berechnet. Die erste Variante wird als *APSP-Problem* („all-pairs shortest paths“) bezeichnet, die zweite Variante als *SSSP-Problem* („single-source shortest paths“).
- Anstatt nur gewichtete, gerichtete Graphen zu untersuchen, sind auch Einschränkungen auf ungewichtete und ungerichtete Graphen von Interesse.

Hauptsächlich gibt es zwei Motivationen, warum die oben genannten Einschränkungen gemacht werden. Erstens lassen sich die spezialisierten Algorithmen oftmals als Bausteine

zur Lösung der allgemeineren Probleme verwenden. Beispielsweise lässt sich manchmal ein dekrementeller Algorithmus zu einem voll-dynamischen Algorithmus erweitern [HK95]. Zweitens lassen sich Effizienzgewinne über allgemeinere Algorithmen erreichen, indem *Bottlenecks* vermieden werden. Möchte man beispielsweise alle paarweisen Distanzen eines Graphen explizit in einer $n \times n$ Matrix speichern, so ist eine Update-Zeit von $\Omega(n^2)$ unvermeidbar, wenn sowohl Einfügungen als auch Löschungen erlaubt sind, da sich durch ein einziges Update $\Omega(n^2)$ Einträge der Matrix verändern können. Zahlreiche weitere Barrieren wurden kürzlich in Form bedingter unterer Schranken gefunden [AVW14, He15].

In dieser Dissertation wurden mehrere partiell-dynamische Approximationsalgorithmen entwickelt. Meist handelt es sich dabei um dekrementelle Algorithmen, da inkrementelle Algorithmen typischerweise einfacher zu erhalten sind als dekrementelle Algorithmen. Insbesondere können alle im Folgenden besprochenen dekrementellen Algorithmen auch als inkrementelle Algorithmen formuliert werden und liefern die gleichen asymptotischen Laufzeiten wie ihre dekrementellen Gegenstücke. Partiell-dynamische Algorithmen erreichen ihre Laufzeitschranken üblicherweise durch Amortisierung über $\Omega(m)$ viele Updates. Im Folgenden verstehen wir unter der *Gesamtlaufzeit* die Summe der Update-Zeiten für alle Updates.

2 Vorbemerkungen

2.1 Stand der Forschung

Dynamische Graphalgorithmen, insbesondere auch zur Berechnung kürzester Wege, sind seit mehr als drei Jahrzehnten ein aktiver Forschungsgegenstand. Daraus ergibt sich eine reichhaltige Vergleichsliteratur. Wir beleuchten im Folgenden nur den State-of-the-Art für die wichtigsten in dieser Dissertation behandelten Probleme.

- *Exaktes dekrementelles SSSP-Problem:* Der deterministische Algorithmus von Even und Shiloach [ES81] kann einen Baum kürzester Wege eines ungewichteten, ungerichteten Graphen laufend aktualisieren, wenn Kanten aus dem Graph gelöscht werden. Seine Gesamtlaufzeit beträgt $O(mn)$ und jedes Query kann in konstanter Zeit beantwortet werden. Der Algorithmus kann so erweitert werden, dass er innerhalb derselben Laufzeitschranken für ungewichtete, gerichtete Graphen funktioniert [HK95]. Für gewichtete, gerichtete Graphen gibt es eine Modifikation mit der pseudopolynomialen Gesamtlaufzeit $O(mnW)$ [Ki99], wenn die Kantengewichte ganze Zahlen von 1 bis W sind.
- *Approximatives dekrementelles SSSP-Problem:* Bernstein und Roditty [BR11] entwickelten einen dekrementellen $(1+\varepsilon)$-approximativen Algorithmus für SSSP in ungewichteten, ungerichteten Graphen. Der Algorithmus ist randomisiert[3] und hat eine Gesamtlaufzeit von $O(n^{2+o(1)})$ und konstante Query-Zeit.

[3] Hier und im Folgenden sind für randomisierte Algorithmen die Gesamtlaufzeiten in Erwartung zu verstehen. Die Korrektheit der randomisierten Algorithmen gilt zudem mit hoher Wahrscheinlichkeit, das heißt mit Wahrscheinlichkeit mindestens $1 - 1/n^c$ für eine Konstante c. Es wird ferner davon ausgegangen, dass die Sequenz von Updates unabhängig von den Zufallsbits des Algorithmus ist.

- *Approximatives dekrementelles APSP-Problem:* Für gewichtete, gerichtete Graphen gibt es einen randomisierten $(1+\varepsilon)$-approximativen dekrementellen Algorithmus für APSP, der eine Gesamtlaufzeit von $\tilde{O}(mn \log W)$[4] und konstante Query-Zeit besitzt [Be13]. Diesem Ergebnis ging ein Algorithmus von Roditty und Zwick [RZ12] voraus, der die gleichen Garantien für ungewichtete, ungerichtete Graphen bot.

2.2 Grundlegende Techniken

Im Folgenden stellen wir zwei grundlegende Techniken vor, die als Bausteine in einer Vielzahl von dynamischen Algorithmen zur Berechnung kürzester Wege verwendet werden.

Einer dieser Bausteine ist der Algorithmus von Even und Shiloach (bzw. die oben erwähnten Erweiterungen), mit dem ein Baum kürzester Wege laufend aktualisiert werden kann. Seine zentrale Bedeutung ergibt sich daraus, dass man den Algorithmus auch so verwenden kann, dass der Baum nur Pfade bis zu einer festgelegten Distanz D enthält. Die Gesamtlaufzeit hierfür beträgt $O(mD)$, was beispielsweise in ungewichteten Graphen bei entsprechend kleinem D weitaus effizienter sein kann als die Laufzeit von $O(mn)$ für einen vollständigen Baum. Falls zudem eine $(1+\varepsilon)$-approximative Lösung gut genug ist, kann der Algorithmus durch ein geschicktes mehrstufiges Runden der Gewichte so erweitert werden, dass der Baum approximative kürzeste Wege mit bis zu h Kanten („hops“) enthält. Die Gesamtlaufzeit hierfür beträgt $\tilde{O}(mh)$.

Der zweite oft verwendete Baustein ist eine Sampling-Technik die zuerst von Ullman und Yannakakis im Kontext paralleler Algorithmen verwendet wurde [UY91]: Wird jeder Knoten des Graphen bei der Initialisierung unabhängig mit Wahrscheinlichkeit $\Theta((\log n)/h)$ ausgewählt, so enthält man eine Menge S, die in Erwartung aus $\tilde{O}(n/h)$ Knoten besteht. Außerdem enthält jeder kürzeste Weg mit mindestens h Kanten mit hoher Wahrscheinlichkeit einen Knoten aus S. Für partiell-dynamische Algorithmen gilt diese Aussage sogar in allen – höchstens n^2 – Versionen des Graphen, die durch die Updates erzeugt werden.

Diese beiden Beobachtungen lassen sich geschickt gemeinsam nutzen, was beispielsweise zu folgendem exemplarischen Algorithmus führen könnte. Wenn man für jeden Knoten der zufällig gewählten Menge S den Algorithmus von Even und Shiloach bis zur Tiefe $2h$ laufen lässt, so benötigt dies insgesamt eine Laufzeit von nur $\tilde{O}(mh(n/h)) = O(mn)$. Wiederholt man diese Vorgehensweise $\log n$ Mal, wobei h jeweils den Wert der nächsten Zweierpotenz annimmt, so hat man implizit bereits alle kürzesten Wege gespeichert: ein kürzester Weg von u nach v besteht, mit hoher Wahrscheinlichkeit, aus einem Teilpfad von u zu einem Knoten $x \in S$ und dann einem Teilpfad von x nach v. Beide Pfade sind bereits im Even-Shiloach Baum von x gespeichert und man muss sie daher nur noch zusammenfügen.[5] Nahezu alle Beiträge dieser Arbeit zielen darauf ab, eine Verbesserung gegenüber dieser wohl bekannten Kombination aus Even-Shiloach Bäumen und zufälliger Auswahl von Knoten zu erreichen.

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

[5] Dies könnte beispielsweise dadurch realisiert werden, dass alle zufällig gewählten Knoten als möglicher Mittelknoten durchprobiert werden, was eine Query-Zeit von $\tilde{O}(n)$ zur Folge hätte.

3 Übersicht der Ergebnisse

Im Folgenden werden die wichtigsten Ergebnisse der Dissertation, sowie die dafür entwickelten algorithmischen Techniken, skizziert.

3.1 APSP in Ungewichteten, Ungerichteten Graphen

Wir erarbeiten zwei dekrementelle Approximationsalgorithmen für das APSP-Problem in ungewichteten, ungerichteten Graphen: Einen randomisierten Algorithmus mit Gesamtlaufzeit $\tilde{O}(n^{2.5})$, der eine $(1+\varepsilon,2)$-Approximation liefert, sowie einen deterministischen Algorithmus mit Gesamtlaufzeit $\tilde{O}(mn)$, der eine $(1+\varepsilon)$-Approximation liefert. Vor dieser Arbeit waren die folgenden beiden Algorithmen als Stand der Technik bekannt: eine randomisierte $(1+\varepsilon)$-Approximation mit Gesamtlaufzeit $\tilde{O}(mn)$ von Roditty und Zwick und ein deterministischer exakter Algorithmus mit Gesamtlaufzeit $\tilde{O}(mn^2)$, bei dem ein Even-Shiloach Baum für jeden Knoten verwendet wird. Somit entspricht die Laufzeit unseres deterministischen Algorithmus der schnellsten bekannten $(1+\varepsilon)$-Approximation und unser randomisierter Algorithmus verbessert die Laufzeit auf Kosten eines zusätzlichen kleinen additiven Fehlers.

3.1.1 Schnellerer Randomisierter Algorithmus

Der randomisierte Algorithmus verwendet den folgenden Ansatz: Zunächst verwenden wir einen dekrementellen Algorithmus, um einen dünn besetzten $(1+\varepsilon,2)$-Emulator H des ursprünglichen Graphen G instand zu halten. Der $(1+\varepsilon,2)$-Emulator H enthält genau die gleichen Knoten wie G sowie gewichtete Kanten, die nicht unbedingt in G enthalten sein müssen, und garantiert dass $d_H(u,v) \leq (1+\varepsilon)d_G(u,v)+2$ für alle Knoten u und v.[6] Zu jeder Zeit kann die Anzahl der Kanten in H durch $\tilde{O}(n^{1.5})$ beschränkt werden, wohingegen der Graph G bis zu $n(n-1)$ viele Kanten enthalten könnte. Auf diesem Graph H wird dann eine Modifikation des Algorithmus von Roditty und Zwick verwendet. Dessen ursprüngliche Laufzeit von $\tilde{O}(mn)$ verringert sich dadurch auf $\tilde{O}(n^{2.5})$. Unser Algorithmus für das Instandhalten des Emulators benötigt eine Gesamtlaufzeit von $\tilde{O}(m\sqrt{n})$ und wird von $\tilde{O}(n^{2.5})$ dominiert. Die finale Approximationsgarantie von $(1+\varepsilon,2)$ ergibt sich aus der Multiplikation der $(1+\varepsilon,2)$-Garantie von H und der $(1+\varepsilon)$-Garantie von Roditty-Zwick, wenn man den Algorithmus mit einem konstant kleineren ε laufen lässt.

Die technische Herausforderung dieses Ansatzes besteht darin, dass, obwohl in G nur Kantenlöschungen vorgenommen werden, in H sowohl Kanten gelöscht als auch hinzugefügt werden könnten, da keine effiziente Methode bekannt ist, um den Emulator H so zu aktualisieren, dass ebenfalls nur Löschungen auftreten. Es ist daher nicht möglich, einen rein dekrementellen Algorithmus – wie etwa jenen von Roditty und Zwick – als „Black Box“ auf dem Emulator H laufen zu lassen. Wir lösen dieses Problem, indem wir den

[6] Ein verwandtes Konzept ist das des k-Spanners: Ein k-Spanner H eines ungerichteten Graphen G ist ein *Subgraph* von G, in dem für alle Knoten u und v gilt, dass $d_H(u,v) \leq k \cdot d_G(u,v)$.

Even-Shiloach Baum, die Hauptkomponente von Roditty-Zwick, so modifizieren, dass er mit den Einfügungen unseres spezifischen Emulators H umgehen kann, ohne die Approximationsgarantie von $(1+\varepsilon, 2)$ zu verletzen. Diese Modifikation nennen wir *monotoner* Even-Shiloach Baum. Führt das Löschen einer Kante in G zum Hinzufügen einer Kante in H, sodass sich die Distanz zweier Knoten in H verringert, so ignoriert unsere Modifikation diese Distanzverringerung in H. Auch wenn unsere Modifikation nahezu trivial ist – reagiere nicht auf Kanteneinfügungen – so ist die Analyse der Approximationsgüte etwas komplexer, da spezielle Eigenschaften von H benötigt werden.

3.1.2 Deterministischer Algorithmus

Unser deterministischer Algorithmus ist eine Derandomisierung des Algorithmus von Roditty und Zwick. Das Herzstück von Roditty-Zwick ist folgende Struktur zur Überdeckung des Graphen, die für mehrere ausgewählte Werte eines Parameters h eingesetzt wird. Zunächst wird durch zufällige Auswahl eine Menge von Knoten S der (erwarteten) Größe $\tilde{O}(n/h)$ generiert, sodass es für jeden Knoten u des Graphen einen Knoten v aus S gibt mit $d_G(u,v) \leq h$, sofern u in einer Zusammenhangskomponente der Größe mindestens h liegt. Dann wird jeder Knoten aus S als Wurzel eines Even-Shiloach Baums mit Tiefe h verwendet. Wir entwickeln eine deterministische Variante dieser Überdeckungsstruktur. Durch eine Greedy-Heuristik lässt sich eine Menge S von Knoten mit den oben genannten Eigenschaften im statischen Fall deterministisch berechnen, wobei die Größe der Menge dadurch beschränkt wird, dass jedem Element von S mindestens $h/2$ Knoten eindeutig zugewiesen werden. Die Herausforderung besteht nun darin, S über die Löschungen in G hinweg aktuell zu halten, da sich im Laufe der Löschungen Zusammenhangskomponenten aufteilen können und somit die Anzahl der zuweisbaren Knoten für einen Knoten aus S unter die Schranke von $h/2$ fallen könnte. Im Algorithmus von Roditty und Zwick taucht dieses Problem aufgrund der Randomisierung nicht auf: die einmal gewählte Menge S hat die gewünschte Eigenschaft mit hoher Wahrscheinlichkeit während aller Kantenlöschungen. Potentiell ergeben sich in unserem deterministischen Ansatz noch zusätzliche Laufzeitverluste dadurch, dass bei Änderungen in S die entsprechenden Even-Shiloach Bäume neu initialisiert werden müssen. Man kann zeigen das jede Verschiebung eines Baums zu einem seiner Nachbarknoten nur mit Kosten $O(m)$ in der Gesamtlaufzeit bestraft wird. Diese Beobachtung kann dann durch geschickt gewählte Regeln zum Aktualisieren der Menge S und Verschieben der Even-Shiloach Bäume so genutzt werden, dass über alle Kantenlöschungen hinweg höchstens $O(n)$ solcher Verschiebungen erforderlich sind. Daher kann der Gesamtaufwand für alle Verschiebungen mit $O(mn)$ abgeschätzt werden, was innerhalb der gewünschten Laufzeit liegt.

3.2 SSSP

Für das partiell-dynamische SSSP-Problem gilt zu beachten, dass es eine bedingte untere Schranke von $\Omega(mn)$ für die Gesamtlaufzeit gibt [RZ11, He15], sowohl für die inkrementelle als auch die dekrementelle Variante und sogar in ungewichteten, ungerichteten

Graphen. Es scheint daher notwendig zu sein, irgendeine Form der Approximation zuzulassen, wenn diese Schranke durchbrochen werden soll. Entsprechende Algorithmen werden im Folgenden vorgestellt.

3.2.1 SSSP in Gewichteten, Ungerichteten Graphen

Für gewichtete, ungerichtete Graphen entwickeln wir einen dekrementellen Algorithmus für eine $(1+\varepsilon)$-Approximation mit der nahezu linearen Gesamtlaufzeit $O(m^{1+o(1)}\log W)$, welche bis auf subpolynomiale Faktoren optimal ist. Dies verbessert die Laufzeit von $O(n^{2+o(1)})$ von Bernstein und Roditty für ungewichtete Graphen [BR11] sowie die Laufzeit von $O(mn\log W)$ des approximativen Even-Shiloach Baums für gewichtete Graphen [Be13].

Die grundlegende Idee unseres Algorithmus ist eine Technik zur Verringerung der zu berücksichtigenden Kantentiefe im Kürzeste-Wege-Baum mit Hilfe eines sogenannten Hopsets. Ein (h,ε)-Hopset eines Graphen $G=(V,E)$ ist eine Menge gewichteter Kanten $F\subseteq V^2$ mit der folgenden Eigenschaft: fügt man F zum ursprünglichen Graph G hinzu, so lässt sich in $G\cup F$ für jedes Paar von Knoten u und v ein Pfad von u nach v finden, dessen Gewicht höchstens $(1+\varepsilon)d_G(u,v)$ beträgt.[7] Hat man ein entsprechendes (h,ε)-Hopset F zur Verfügung, lässt sich auf $G\cup F$ der approximative Even-Shiloach Baum mit Tiefe h anwenden, um eine $(1+\varepsilon)$-Approximation von SSSP mit Gesamtlaufzeit $O(mh)$ zu erhalten. Intuitiv findet durch das Hopset eine „Verdichtung“ des Graphen auf einen kleinen Durchmesser auf Kosten eines relativ kleinen multiplikativen Fehlers statt. Insbesondere unterscheidet sich diese Idee vom „Sparsification“-Ansatz früherer Algorithmen, etwa dem von Bernstein und Roditty [BR11] sowie dem in Abschnitt 3.1.1 vorgestellten. Dies scheint notwendig, da diese Art der Sparsification alleine nicht dazu geeignet ist eine lineare Laufzeit zu erhalten, wenn – grob gesprochen – in der Laufzeit m durch n ersetzt wird.

Hopsets wurden von Cohen zur schnellen parallelen Berechnung von (statischem) SSSP eingeführt [Co00]. Cohens Konstruktion liefert ein $(\text{polylog}\,n,\varepsilon)$-Hopset mit $O(m^{1+o(1)})$ Kanten. Wir verwenden stattdessen ein $(n^{o(1)},\varepsilon)$-Hopset mit $O(m^{1+o(1)})$ Kanten, das auf der Spanner-Konstruktion von Thorup und Zwick [TZ06] basiert. Die Herausforderung für uns ist, das Hopset auch tatsächlich auf dem aktuellen Stand zu halten, wenn Kanten im ursprünglichen Graph gelöscht werden. Dies gelingt uns mit unserem eigens auf diesen Zweck zugeschnittenen Hopset durch Verwendung des monotonen Even-Shiloach Baums aus Abschnitt 3.1.1, wohingegen dies für Cohens Hopset bisher nicht gelungen ist. In Bezug auf die Anwendung für dekrementelles SSSP würde aber auch der etwas bessere h-Parameter von Cohens Hopset keine Verbesserung der asymptotischen Laufzeit liefern.

3.2.2 SSSP in Gewichteten, Gerichteten Graphen

Dieser Algorithmus funktioniert für den allgemeinsten Fall – gewichtete, gerichtete Graphen. Wir erhalten eine $(1+\varepsilon)$-Approximation mit Gesamtlaufzeit $O(mn^{0.9+o(1)})$, wenn das

[7] Zusätzlich muss beachtet werden, dass das Gewicht jeder Kante $(u,v)\in F$ mindestens der Distanz $d_G(u,v)$ entspricht, das heißt, dass $G\cup F$ die ursprünglichen Distanzen nicht unterschätzt.

größte Kantengewicht W durch $2^{log^c n}$ für eine Konstante c beschränkt ist. Es handelt sich hierbei um den ersten Algorithmus, der die $O(mn)$-Barriere durchbricht. Diese Barriere bestand auch für das vermeintlich einfachere dekrementelle *single-source reachability (SSR)* Problem und die Frage, ob ein $o(mn)$-Algorithmus existiert, war lange Zeit offen [Ki08].

Mit unserem Ansatz ergeben sich zwei leicht unterschiedliche Algorithmen für dünn und dicht besetzte Graphen. Die oben erwähnte obere Laufzeit von $O(mn^{0.9+o(1)})$ dient als obere Schranke für beide Algorithmen. In Grenzfällen werden bessere Laufzeiten erzielt. Beispielsweise hat der Algorithmus für das dekrementelle SSR-Problem eine Laufzeit von $O(n^{2+2/3+o(1)})$, wenn $m = \Omega(n^2)$. Außerdem erhalten wir dieselben Laufzeiten wie für SSR auch für das Problem, die starken Zusammenhangskomponenten eines gerichteten Graphen dynamisch auf dem aktuellen Stand zu halten. Dies folgt aus einer Modifikation der Reduktion von Roditty und Zwick [RZ08]. Auch hier ist dies die erste Verbesserung der zuvor bekannten oberen Schranke von $O(mn)$.

Die technische Neuerung hinter diesem Algorithmus ist folgender *double sampling* Ansatz: Durch Zufallsauswahl erhalten wir zwei Mengen von Knoten A und B, die mit hoher Wahrscheinlichkeit folgende Eigenschaften erfüllen: (1) Jeder kürzester Weg des Graphen mit ausreichend vielen Kanten enthält einen Knoten aus A. (2) Jedes Paar von Knoten u und v aus A besitzt entweder eine Verbindung über einen Knoten aus B mit wenigen Kanten oder die Anzahl der Knoten auf allen kurzen Pfaden von u nach v ist klein. Eigenschaft (1) wurde bereits in früheren Algorithmen verwendet (siehe Abschnitt 2.2). Eigenschaft (2) wurde bisher hingegen noch nicht ausgenutzt und erlaubt uns die Anzahl der zu berücksichtigenden Knoten durch die Wahrscheinlichkeit für die Zufallsauswahl zu steuern. Der Algorithmus verwendet nun für jeden Knoten aus B einen „eingehenden“ und einen „ausgehenden“ Even-Shiloach Baum, mit dem für jedes Paar u und v von Knoten aus A festgestellt werden kann, ob es noch eine kurze Verbindung von u nach v über einen Knoten aus B gibt. Sobald dies nicht mehr der Fall ist, konstruieren wir einen Graph der alle kurzen Wege von u nach v enthält und lassen eine Instanz des Even-Shiloach Baums auf diesem Graph laufen. Durch Eigenschaft (2) kann garantiert werden, dass dieser Graph nur wenige Knoten enthält, wodurch sich ein Effizienzgewinn ergibt. Durch geeignete Wahl der Parameter lassen sich hiermit approximative kürzeste Wege zwischen allen Knoten aus A auf dem aktuellen Stand halten. Mit bekannte Techniken lässt sich dies dann auf einen approximativen SSSP-Algorithmus erweitern.

3.2.3 Inkrementeller SSSP Algorithmus für Ungewichtete, Ungerichtete Graphen

Zuletzt stellen wir einen $(1+\varepsilon)$-approximativen inkrementellen Algorithmus für SSSP in ungewichteten, ungerichteten Graphen mit Gesamtlaufzeit $O(m^{3/2}n^{1/4}\log n)$ vor. Dieselbe Idee führt außerdem zu sowohl einen inkrementellen also auch einen dekrementellen dynamischen Algorithmus im synchronisierten verteilten Modell, den wir aus Platzgründen nicht besprechen.

Die Hauptidee des Algorithmus ist, einen Baum für kürzeste Wege bewusst „nachlässig“ zu aktualisieren. Der inkrementelle Algorithmus hat zwei Parameter k und Δ und teilt die

Updates in Phasen ein, die aus jeweils k Updates bestehen. Zu Beginn jeder Phase wird ein Baum kürzester Wege T vom Startknoten s aus, zusammen mit den zugehörigen Distanzen $\delta(s,\cdot)$, durch Breitensuche berechnet. Wird eine Kante (u,v) in den Graph eingefügt, so unterscheiden wir zwei Fälle.[8] (1) Falls, $\delta(v)-\delta(u)>\Delta$, so startet der Algorithmus vorzeitig eine neue Phase. (2) Falls $\delta(v)-\delta(u)\leq\Delta$, so fügen wir die Kante (u,v) in den Baum T ein, indem wir v als Kind von u verbinden und die Kante zum vorherigen Elternknoten von v entfernen. Jedes Mal wenn Fall (2) eintritt, erhöht sich der Fehler unsere Distanz-Schätzung $\delta(s,\cdot)$ vom Beginn der Phase um einen additiven Fehler von höchstens Δ. Das heißt, dass nach k solchen Einfügungen der additive Fehler höchstens $k\Delta$ beträgt. Umgekehrt finden wir in Fall (1) einen Knoten dessen Distanz zum Startknoten sich um mindestens Δ verringert. Da die maximale Distanz zum Startknoten in einem ungewichteten Graph höchstens n beträgt, kann dies für jeden Knoten höchsten n/Δ Mal passieren, also insgesamt n^2/Δ Mal. Da der Graph aber zusätzlich noch ungerichtet ist, verringert sich in diesem Fall auch die Distanz zu s für alle Knoten, die nahe genug an v liegen. Es lässt sich zeigen, dass aufgrund dieser Beobachtung der erste Fall höchstens $O(n^2/\Delta^2)$ Mal auftreten kann. Bei q Kanteneinfügungen lässt sich die Anzahl der Phasen des Algorithmus somit durch $O(q/k+n^2/\Delta^2)$ beschränken. Die Gesamtlaufzeit des Algorithmus wird von der Breitensuche zu Beginn jeder Phase dominiert. Somit lässt sich ein additiver Fehler von $k\Delta$ mit einer Gesamtlaufzeit von $O((q/k+n^2/\Delta)\cdot m)$ erreichen. Hieraus erhält man eine $(1+\varepsilon)$-Approximation, wenn man für kleine Distanzen zusätzlich einen inkrementellen Even-Shiloach Baum bis zur Tiefe $k\Delta/\varepsilon$ verwendet. Die oben erwähnte Gesamtlaufzeit erhält man nun durch eine geeignete Wahl der Parameter k und Δ.

4 Zusammenfassung und Ausblick

In dieser Arbeit wurden mehrere offene Probleme in Bereich dynamischer Graphalgorithmen gelöst. Randomisierung war oft ein essentieller Bestandteil der resultierenden Algorithmen. Daher stellt sich die Frage, ob sich die Ergebnisse weiter konsolidieren lassen, indem sie mit deterministischen Algorithmen „nachgebaut“ werden. In dieser Dissertation wurde dies beispielsweise mit der Derandomisierung des Algorithmus von Roditty und Zwick [RZ12] vorgemacht. Eine weitere interessante Frage ist, ob die in dieser Dissertation entwickelten Techniken auch für nicht-dynamische Probleme nützlich sein könnten. So konnte beispielsweise eine etwas einfachere Version des in Abschnitt 3.2.1 erwähnten Hopsets dazu verwendet werden, einen nahezu optimalen *verteilten* Algorithmus für approximative kürzeste Wege zu entwickeln [HKN16]. Da dynamische Algorithmen oft nur *lokale* Berechnungen durchführen, könnten unsere Techniken noch weitere Anwendung für parallele und verteilte Algorithmen finden.

Literaturverzeichnis

[AVW14] Abboud, Amir; Vassilevska Williams, Virginia: Popular conjectures imply strong lower bounds for dynamic problems. In: Symposium on Foundations of Computer Science (FOCS). S. 434–443, 2014.

[8] Da der Graph ungerichtet ist, müssen die beiden Fälle zusätzlich mit vertauschten Rollen von u und v getestet werden.

[Be13] Bernstein, Aaron: Maintaining Shortest Paths Under Deletions in Weighted Directed Graphs. In: Symposium on Theory of Computing (STOC). S. 725–734, 2013.

[BR11] Bernstein, Aaron; Roditty, Liam: Improved Dynamic Algorithms for Maintaining Approximate Shortest Paths Under Deletions. In: Symposium on Discrete Algorithms (SODA). S. 1355–1365, 2011.

[Co00] Cohen, Edith: Polylog-Time and Near-Linear Work Approximation Scheme for Undirected Shortest Paths. Journal of the ACM, 47(1):132–166, 2000.

[ES81] Even, Shimon; Shiloach, Yossi: An On-Line Edge-Deletion Problem. Journal of the ACM, 28(1):1–4, 1981.

[He15] Henzinger, Monika; Krinninger, Sebastian; Nanongkai, Danupon; Saranurak, Thatchaphol: Unifying and Strengthening Hardness for Dynamic Problems via the Online Matrix-Vector Multiplication Conjecture. In: Symposium on Theory of Computing (STOC). S. 21–30, 2015.

[HK95] Henzinger, Monika; King, Valerie: Fully Dynamic Biconnectivity and Transitive Closure. In: Symposium on Foundations of Computer Science (FOCS). S. 664–672, 1995.

[HKN16] Henzinger, Monika; Krinninger, Sebastian; Nanongkai, Danupon: A Deterministic Almost-Tight Distributed Algorithm for Approximating Single-Source Shortest Paths. In: Symposium on Theory of Computing (STOC). 2016.

[Ki99] King, Valerie: Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. In: Symposium on Foundations of Computer Science (FOCS). S. 81–91, 1999.

[Ki08] King, Valerie: Fully Dynamic Transitive Closure. In: Encyclopedia of Algorithms. 2008.

[RZ08] Roditty, Liam; Zwick, Uri: Improved Dynamic Reachability Algorithms for Directed Graphs. SIAM Journal on Computing, 37(5):1455–1471, 2008.

[RZ11] Roditty, Liam; Zwick, Uri: On Dynamic Shortest Paths Problems. Algorithmica, 61(2):389–401, 2011.

[RZ12] Roditty, Liam; Zwick, Uri: Dynamic Approximate All-Pairs Shortest Paths in Undirected Graphs. SIAM Journal on Computing, 41(3):670–683, 2012.

[TZ06] Thorup, Mikkel; Zwick, Uri: Spanners and emulators with sublinear distance errors. In: Symposium on Discrete Algorithms (SODA). S. 802–809, 2006.

[UY91] Ullman, Jeffrey D.; Yannakakis, Mihalis: High-Probability Parallel Transitive-Closure Algorithms. SIAM Journal on Computing, 20(1):100–125, 1991.

Sebastian Krinninger wurde 1986 in Gräfelfing geboren. Er studierte von 2005 bis 2008 im Bachelor-Studiengang Informatik an der Universität Passau sowie von 2008 bis 2011 im Master-Studiengang Computational Intelligence an der Technischen Universität Wien, wo er eine Abschlussarbeit im Bereich der mathematischen Fuzzy-Logik schrieb. Von 2011 bis 2015 verfasste er seine Doktorarbeit über dynamische Graphalgorithmen zur Berechnung kürzester Wege unter der Betreuung von Monika Henzinger an der Universität Wien. Das Herbstsemester 2015 verbrachte er als Fellow am Simons Institute for the Theory of Computing in Berkeley. Seit Anfang 2016 forscht er als Postdoc am Max-Planck-Institut in Saarbrücken.