Algorithmenentwurfstechniken für parametrisierte Graphmodifikationsprobleme

Jiong Guo*
Institut für Informatik, Friedrich-Schiller-Universität Jena
guo@minet.uni-jena.de

Abstract: Meine Dissertation „Algorithm Design Techniques for Parameterized Graph Modification Problems“ untersucht die Anwendbarkeit von vier Techniken zur Entwicklung parametrisierter Algorithmen für Graphmodifikationsproblemen. Dies sind zwei klassische Techniken, nämlich Datenreduktion und tiefenbeschränkte Suchbäume, und zwei neue Techniken, nämlich iterative Kompression und Parametrisierung bzgl. der Distanz zu einer „schnell“ lösbaren Instanz.

1 Einführung und Überblick

Wie kann man ein gegebenes Objekt so verändern, dass es durch eine kostenoptimale Transformation einen gewünschten Zustand erreicht? Diese Frage spielt nicht nur eine wichtige Rolle, wenn man aus Edelsteinen Schmuck fertigt oder aus Stoffbahnen mit möglichst wenig Verschnitt Kleidung produziert, sondern ist auch Gegenstand der algorithmischen Graphtheorie. Man spricht hier von Graphmodifikationsproblemen. Da Graphen ein sehr universelles Modellierungswerkzeug sind, können sich so viele Anwendungsprobleme aus verschiedensten Gebieten wie z.B. Biologie, Wirtschafts- und Sozialwissenschaften oder der Informatik selbst [Har69, Rob89] als Graphmodifikationsprobleme begreifen lassen.

Betrachten wir als ein konkretes einfaches Beispiel das Problem der „Konfliktauflösung“: In vielen sozialen Prozessen treten verschiedene Interessengruppen auf, welche bezüglich spezifischer Themen gegenläufige Meinungen haben. Nehmen wir nun an, es gibt in einer Gesellschaft sehr viele Interessengruppen. Dann ist es für einen Konfliktmoderator mitunter unmöglich, alle Interessengruppen zu befragen, um darauf basierend an einer Konfliktauflösung zu arbeiten. Deshalb ist es wünschenswert, eine ausgesuchte, möglichst kleine Menge von Interessengruppen zu identifizieren, sodass der Moderator nach deren Kontaktierung das nötige Wissen hat, um alle Konflikte zu beseitigen bzw. zumindest zu glätten. Die Auswahl einer kleiner Menge von Interessengruppen lässt sich nun als ein sehr spezielles Graphmodifikationsproblem modellieren. Dabei identifiziert man jede Gruppe mit einem Knoten und man fügt eine Kante zwischen zwei Knoten

*Finanziert im Rahmen des DFG-Aktionsplans Informatik, NI 369/4 (Kleine Parameter in schwierigen Problemen: Entwurf, Analyse, Implementierung und Anwendung von Festparameteralgorithmen (PIAF)).

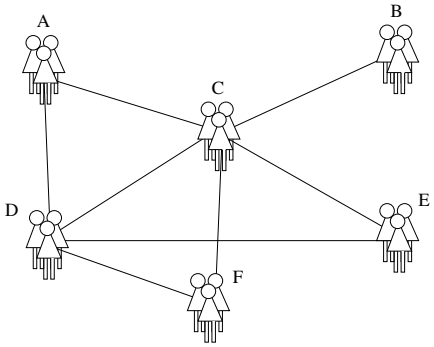

Figure 1: Ein Beispiel für Konfliktauflösung. Gruppen C und D bilden eine optimale Auswahl.

ein, falls die entsprechenden Gruppen widerstreitende Interessen haben. Die gesuchte Auswahl entspricht dann genau der Suche nach einer kleinstmöglichen Knotenmenge, deren Löschung einen kantenfreien Graphen ergibt. Siehe Abbildung 1 für ein Beispiel.

Wie am konkreten Beispiel angedeutet, lassen sich viele Probleme der realen Welt modellieren, in dem man Knoten zur Darstellung von Objekten und Kanten zur Darstellung der Beziehungen zwischen Objekten verwendet. Seit Einführung der graphbasierten Problemmodellierung wurden vielfach auch *Graphmodifikationsprobleme* formuliert. Naheliegende Fragestellungen im Hintergrund sind hier beispielsweise Themen wie Fehlerkorrektur, Konfliktauflösung oder Systemrekonfiguration. In allgemeinster Form lassen sich Graphmodifikationsprobleme als folgende „Minimierungsaufgabe“ beschreiben:

Eingabe: Ein Graph und eine gewünschte Grapheigenschaft Π.
Aufgabe: Führe eine minimale Anzahl von „elementaren Modifikationsoperationen“ derart durch, dass der Graph anschließend die Eigenschaft Π besitzt.

Hinsichtlich elementarer Modifikationsoperationen wollen wir uns hier auf die wohl einfachsten und natürlichsten konzentrieren, das heißt auf das Einfügen und Löschen von Kanten und das Löschen von Knoten. Zur Verdeutlichung des Konzepts wollen wir nun noch ein zweites Beispielproblem mit Anwendungen in der Bioinformatik und dem Maschinellen Lernen aufgreifen. Das Problem heißt Cluster Editing (und wird in Abschnitt 3 eingehender studiert) und besteht darin, für einen gegebenen Graphen eine kleinstmögliche Anzahl von zu löschenden und einzufügenden Kanten zu finden, sodass die Zusammenhangskomponenten des entstehenden Graphen vollständig verbundene Teilgraphen (englisch: Cliquen) sind. Ein Anwendungsszenario sieht hier etwa folgendermaßen aus: Gegeben seien irgendwelche Einträge aus verschiedenen Datenbanken (also z.B. Namen und Adressen von Forschern); das Ziel ist, Einträge zusammen zu fügen, die zum selben Objekt gehören. Beispielsweise könnte im Falle von Forschern ein und die selbe Person mehrfach in den Datenbanken mit verschiedenen Adressen auftauchen. Stellt man nun die Einträge als Knoten dar und fügt eine Kante zwischen zwei Knoten ein, falls ein sogenannter „Klassifizierer“ glaubt, dass die zwei Knoten das gleiche Objekt repräsentieren, so ergibt sich auf natürliche Weise das Cluster Editing-Problem. Nach Durchführung einer

minimalen Anzahl von Kantenmodifikationen erhält man eine Instanz, in der jede Clique ein Objekt repräsentiert. Weitere Hintergründe zu dieser Daten-Clustering-Fragestellung finden sich in der Arbeit von Bansal et al. [BBC04].

Etwa seit 1970 wurde die Berechnungskomplexität von Graphmodifikationsproblemen intensiv studiert. Leider stellte sich heraus, dass für die meisten Grapheigenschaften die zugehörigen Modifikationsprobleme NP-schwer sind [BBD06, GJ79], das heißt, optimale Lösungen können wohl nur in exponenzieller Laufzeit berechnet werden. In dieser Arbeit gehen wir einer wichtigen Eigenschaft vieler Graphmodifikationsprobleme in ihrer Form als Minimierungsprobleme nach: Typischer Weise ist zu erwarten, dass die Anzahl der gesuchten Modifikationsoperationen (sprich, die Lösungsgröße) relativ klein ist. Wäre diese Anzahl groß, so wären die damit verbundenen Kosten oft zu groß, um noch für die dahinterstehende Anwendung relevant zu sein. Deshalb ist es ein natürlicher Ansatz zum Finden optimaler Lösungsmengen, bei den zugehörigen exakten Algorithmen zu versuchen, die scheinbar nicht vermeidliche exponentielle Laufzeitexplosion allein auf die Lösungsgröße zu beschränken. Beispielsweise wäre ein Algorithmus mit Laufzeit $O(2^k \cdot n^2)$ (für Lösungsgröße k und Knotenzahl n) sicher praktisch nützlicher als ein Algorithmus mit Laufzeit $O(2^n \cdot n^2)$. Lässt sich die exponentielle Laufzeitkomponente wie beschrieben allein auf den Parameter k beschränken, so spricht man von *parametrisierten Algorithmen*. Diese sind zentraler Gegenstand der Dissertation.

An dieser Stelle nun ein paar einführende Worte zur Analyse der parametrisierten Komplexität von Problemen [DF99, Nie06]. Im Kontext dieser Arbeit ist ein parametrisiertes Problem immer ein Tupel (G, k) bestehend aus einem Eingabegraph G und einem Parameter k, einer natürlichen Zahl, die als Beschränkung der Lösungsgröße dient. Durch diese Splittung in zwei Eingabekomponenten lässt sich die parametrisierte Komplexitätstheorie als zweidimensionale Theorie auffassen. In unserem Kontext beschreibt die zweite Komponente, also der Parameter k, immer die Anzahl benötigter Graphmodifikationsoperationen (welche ja möglichst klein gehalten werden soll). Wir nennen ein Graphmodifikationsproblem „*fixed-parameter tractable*“, kurz *FPT*, wenn es einen Algorithmus gibt, der in Laufzeit $O(f(k) \cdot n^c)$ für konstantes c und eine beliebige, nur von k abhängige Funktion f, eine Lösung der Größe höchstens k findet. Man beachte, dass diese Definition für einen konstanten Wert von k eine polynomielle Laufzeit impliziert.

Diese Arbeit liefert nun erste Ergebnisse zu einer systematischen Untersuchung der parametrisierten Komplexität von Graphmodifikationsproblemen. Kernpunkt der Arbeit ist die Entwicklung algorithmischer Techniken zur Gewinnung effizienter parametrisierter Algorithmen. Dies wird nachfolgend etwas genauer beschrieben.

In Abschnitt 2 studieren wir die Technik der iterativen Kompression. Unser zentrales Ergebnis hierbei ist die Beschreibung eines entsprechenden parametrisierten Algorithmus zur Lösung des Feedback Vertex Set-Problems. Dieses Ergebnis löst ein zehn Jahre offenes Problem der algorithmischen Graphtheorie.

In Abschnitt 3 behandeln wir eine der wichtigsten Techniken der parametrisierten Algorithmik: Polynomzeit-Datenreduktion und Reduktion auf einen Problemkern. Hier entwerfen wir einen Problemkern der Größe $O(k^2)$ für das eingangs erwähnte Cluster Editing-Problem und erzielen somit das erste FPT-Ergebnis im Daten-Clustering-Bereich.

In Abschnitt 4 untersuchen wir Suchbaumtechniken in Zusammenhang mit der Charakterisierung von Grapheigenschaften durch verbotene Teilgraphen. Auch hier steht Cluster Editing als Beispiel im Mittelpunkt.

In Abschnitt 5 wenden wir uns schließlich einer neuen Weise der Parametrisierung zu. Anders als vorher beschrieben ist hier nicht die Lösungsgröße der Parameter, sondern es wird eine strukturelle Parametrisierung, die den Abstand von in Polynomzeit lösbaren Spezialfällen misst, eingeführt. In diesem Zusammenhang stellen wir Ergebnisse für das Netzwerkflussproblem Multicommodity Demand Flow in Trees vor.

Das Papier schließt in Abschnitt 6 mit einem kurzen Überblick zur bisherigen Resonanz der Ergebnisse der Dissertation und zu aktuellen und zukünftigen Forschungen.

2 Iterative Kompression

Angenommen, man kennt für ein NP-schweres Optimierungsproblem eine zulässige aber im allgemeinen nicht-optimale Lösung. Hilft die Kenntnis dieser Lösung dann, eine noch bessere Lösung – falls eine soche existiert – „schnell“ zu finden? Die neue Technik der iterativen Kompression zeigt, dass sich dies in manchen Fällen tatsächlich gewinnbringend realisieren lässt.

Grob gesprochen ist die grundlegende Idee der iterativen Kompression wie folgt: Um zu zeigen, dass ein Minimierungsproblem FPT ist, genügt es einen parametrisierten Algorithmus anzugeben, der aus einer gegebenen Lösung der Größe $k+1$ – falls existierend – eine neue Lösung der Größe k konstruiert. Dabei benutzt die Technik der iterativen Kompression zwei zentrale Prozeduren. Die Kompressionsprozedur konstruiert aus einer Lösung der Größe $k+1$ eine der Größe k. Die Iterationsprozedur versorgt die Kompressionsprozedur mit der Lösung der Größe $k+1$; sie ist die Hauptprozedur und benutzt die Kompressionsprozedur als Unterroutine.

Wir illustrieren die Technik der iterativen Kompression anhand des Feedback Vertex Set-Problems:

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine ganze Zahl $k \geq 0$;
Aufgabe: Bestimme eine Knotenmenge $F \subseteq V$ mit $|F| \leq k$ derart, dass die Löschung aller Knoten aus F zu einem kreisfreien Graphen führt.

Die Knotenmenge F heißt „Feedback Vertex Set“. Das Graphmodifikationsproblem Feedback Vertex Set gehört zu den klassischen NP-schweren Problemen [GJ79]. Schon Anfang der neunziger Jahre des letzten Jahrhunderts wurde gezeigt, dass Feedback Vertex Set FPT ist. Danach blieb mehr als zehn Jahre offen, ob ein parametrisierter Algorithmus mit Laufzeit $O(c^k \cdot |V|^{O(1)})$ für konstantes c existiert. Im Rahmen der Dissertation konnte diese Frage dank iterativer Kompression mit „Ja“ beantwortet werden.

Wir skizzieren nachfolgend die zentrale Komponente des Ansatzes, die Kompressionsprozedur, in zwei Schritten:

1. Betrachte alle Partitionen (X, Y) einer Feedback Vertex Set der Größe $k+1$ in zwei

disjunkte Teilmengen X und Y mit $|X| \leq k$. Nimm weiterhin an, dass X vollkommen in der neuen Feedback Vertex Set F' der Größe k enthalten sei, sprich $X \subseteq F'$, und dass $Y \cap F' = \emptyset$.

2. Falls in einer solchen Partition Y einen Kreis in G induziert, so kann dies zu keiner verbesserten Lösung führen und man stoppt hier. Andernfalls werden alle Knoten in X aus G gelöscht. Weiterhin werden Knoten vom Grad 1 bzw. Grad 2 aus dem verbliebenen Graphen gelöscht. Zuletzt wird jede Knotenmenge $S \subseteq V$ mit $S \subseteq V \setminus F$ mit $|S| \leq k - |X|$ dahingehend untersucht, ob die Hinzufügung von S zu X eine Feedback Vertex Set der Größe höchstens k ergibt; gegebenenfalls wird eine entsprechende Lösung $X \cup S$ ausgegeben.

Der beweistechnisch schwierige Teil ist bei obiger Prozedur, dass sie in Laufzeit $O(c^k \cdot |E|)$ durchgeführt werden kann. Der springende Punkt hierbei ist, dass es 2^{k+1} Partitionen von F in X und Y gibt und dass man zeigen kann, dass der Graph nach Löschen von X und Knoten vom Grad 1 und 2 aus nur $O(k)$ Knoten besteht (da sonst keine Feedback Vertex Set der Größe k existiert).

Verglichen mit der Kompressionsprozedur ist die Iterationsprozedur recht einfach: Man betrachtet mit wachsendem i schrittweise die von $\{v_1, \ldots, v_i\}$ induzierten Teilgraphen G_i. Der Fall $i = 1$ ist trivial. Für $i > 1$ gilt, dass eine optimal große Feedback Vertex Set für G_i durch Hinzufügen von v_{i+1} zu einer um höchstens eins zu großen Feedback Vertex Set für G_{i+1} erweitert werden kann. Darauf wird nun wiederum die Kompressionsprozedur angewandt, um eine optimale Lösung zu erhalten. Nach $|V|$ Iterationen dieser Art erhält man somit eine optimale Lösung für den Gesamtgraphen.

Abschließend sei noch bemerkt, dass obiger Algorithmus auch zum *Aufzählen* aller optimalen Lösungen für Feedback Vertex Set benutzt werden kann. Darüber hinaus liefert eine ähnliche Strategie einen Algorithmus mit Laufzeit $O(2^k \cdot |E|^2)$ für das NP-schwere Edge Bipartization-Problem [GGH$^+$06]. Hier geht es darum, eine kleinstmögliche Anzahl von Kanten in einem Graphen zu finden, so dass der entstehende Graph durch deren Löschung bipartit wird. In der Dissertation werden weiterhin noch zwei Beschleunigungstechniken für iterative Kompression vorgestellt, auf die hier nicht eingegangen werden kann.

3 Datenreduktion und Problemkerne

Bevor laufzeitintensive Algorithmen auf NP-schwere Probleme angewandt werden, ist es naheliegend zu versuchen, „einfache Teile“ der Eingabe vorab zu behandeln und somit die Eingabegröße effizient zu reduzieren. Übrig bleibt dann, was man den Problemkern nennt. Erst nach einer solchen Vorverarbeitung sollten laufzeitintensive Algorithmen angewandt werden. Ein aktuelles Beispiel für die Wichtigkeit einer solchen Strategie liefert die Welt des (ganzzahligen) linearen Programmierens, der Kerntechnik der kombinatorischen Optimierung und des mathematischen Programmierens. So beschreibt Bixby [Bix02], welch tragende Rolle Datenreduktionsregeln für das kommerzielle Lösungspaket CPLEX spielen, indem er aufzeigt, wie ein großes lineares Programm nach Vorverarbeitung durch Datenreduktion innerhalb einer halben Stunde gelöst werden kann, jedoch weit von jeder

praktischen Lösbarkeit mit einer „reinen“ CPLEX-Anwendung ist.

Im Kontext parametrisierter Komplexitätsanalyse sind Datenreduktionsregeln von großer Bedeutung. Hier geht es nicht nur um rein empirische bzw. heuristische Aspekte des Wertes von Datenreduktionen, sondern um *beweisbare* Probleminstanzverkleinerungen, die durch sie erzielt werden können. Formal geht es beim Prozess der Datenreduktion um folgendes: Gegeben sei eine Eingabeinstanz (G, k), dann ist das Ziel eine neue Instanz (G', k') derart zu konstruieren, dass G eine Lösung der Größe k genau dann besitzt, wenn G' eine Lösung der Größe k' besitzt. Dabei muss zudem gelten, dass die Größe von G' durch eine Funktion allein von k beschränkt werden kann, dass $k' \leq k$ ist, und dass die Konstruktion (Transformation) in polynomieller Laufzeit durchgeführt werden kann. Die derart erzielte, reduzierte Instanz (G', k') heißt *Problemkern*, und der ganze Prozess wird *Reduktion auf einen Problemkern* oder auch „*Kernelisierung*“ genannt.

Wir veranschaulichen das Konzept mit Hilfe des Problmes Cluster Editing:

Eingabe: Ein ungerichteter Graph $G = (V, E)$ und eine ganze Zahl $k \geq 0$.
Aufgabe: Bestimme eine Menge P mit $|P| \leq k$ bestehend aus zweielementigen Teilmenge von V derart, dass durch Hinzufügen der Kanten in $P \setminus E$ und durch Löschen der Kanten in $P \cap E$ der resultierende Graph eine disjunkte Vereinigung von vollständigen Teilgraphen (Cliquen) ist.

Wie bereits in der Einleitung dieser Artikels dargestellt, tritt Cluster Editing sowohl in der Bioinformatik als auch beim Maschinellen Lernen auf [SST04, BBC04].

Um zwei Datenreduktionsregeln vorstellen zu können, benötigen wir ein paar Begriffe: Als *gemeinsamen Nachbar* zweier Knoten u und v eines Graphen $G = (V, E)$ bezeichnen wir einen Knoten $z \in V$, für den $\{u, z\} \in E$ und $\{v, z\} \in E$ gilt. Analog ist ein *nicht-gemeinsamer Nachbar* von u und v ein Knoten $z \in V$, sodass entweder $\{u, z\} \in E$ oder $\{v, z\} \in E$ gilt. Nun die zwei Regeln:

Regel 1: Führe folgendes für jedes Paar $u, v \in V$ durch:

(1) Falls u und v mehr als k gemeinsame Nachbarn haben, so füge gegebenenfalls $\{u, v\}$ zu E hinzu.

(2) Falls u und v mehr als k nicht-gemeinsame Nachbarn haben, so lösche gegebenenfalls $\{u, v\}$ aus E.

(3) Falls u und v sowohl mehr als k gemeinsame als auch mehr als k nicht-gemeinsame Nachbarn haben, so kann die gegebene Eingabeinstanz keine Lösung der Größe k besitzen und der Algorithmus kann abbrechen.

Regel 2: Lösche alle Zusammenhangskomponenten, welche Cliquen bilden.

Mit Hilfe obiger zwei Regeln lässt sich folgendes beweisen:

Theorem 3.1. CLUSTER EDITING *besitzt einen Problemkern mit* $O(k^2)$ *Knoten.*

Seit Abschluss der Promotion wurde weltweit, motiviert durch obiges Ergebnis, aktiv an parametrisierten Algorithmen für Cluster Editing geforscht: Protti et al. [PdSS06] verbesserten die Laufzeit der Problemkernreduktion von kubisch auf linear. Fellows et al. [FLRS06] präsentierten einen Problemkern mit höchstens $24k$ Knoten. Letztlich kon-

nte dies weiter verbessert werden zu einem Problemkern mit nur noch $4k$ Knoten [Guo07]. Eine experimentelle Untersuchungen von Cluster Editing-Algorithmen, bezogen auch auf einem im nächsten Abschnitt diskutierten Suchbaumalgorithmus, wurden von Dehne et al. [DLL$^+$06] vorgestellt. Sie stellen deutlich Vorteile gegenüber Approximationsalgorithmen und Heuristiken fest. Abschließend sei noch erwähnt, dass in der Dissertation auch eine technisch deutlich kompliziertere Problemkernreduktion für das NP-schwere Multicut in Trees-Problem vorgestellt wird.

4 Verbotene Teilgraphen und Suchbäume

Kehren wir nochmals kurz zum Feedback Vertex Set-Problem zurück. Dort geht es letztlich darum, eine kleinstmögliche Knotenmenge zu finden, sodass durch deren Löschung alle Kreise im Graphen zerstört werden. Ein naiver Ansatz zur Lösung des Problems wäre nun, immer nach einem Kreis im Graphen zu suchen und alle Fälle der Löschung eines Kreisknotens auszuprobieren. Dies führt zu einer rekursiven Suchbaumstrategie, denn eine der betrachteten Möglichkeiten *muss* zu einer optimalen Lösung führen. Leider hat diese Strategie hier einen Haken: Wenn wir einen Kreis wie oben beschrieben bearbeiten, so kann es sein, dass der Kreis „beliebig“ groß ist (und auch kein kurzer zu finden ist) und dadurch die angedeutete Fallunterscheidung entsprechend (zu) viele Fälle zu betrachten hätte. Der Suchbaum hätte dann auf jeden Falle eine Größe, die exponentiell in der Größe des gegebenen Graphen ist.

Für Feedback Vertex Set führt eine Suchbaumstrategie wie oben beschrieben – anders als iterative Kompression – offenbar nicht zu einem effizienten parametrisierten Algorithmus. Dennoch können wir daraus etwas Positives mitnehmen: Die grundsätzliche Vorgehensweise beruhte auf dem Suchen nach einem *verbotenen Teilgraphen*, in diesem Fall dem eines Kreises. Bei Feedback Vertex Set ist das Problem, dass die Größe der Teilstruktur zu groß sein kann. Bei anderen Problemen ist das nicht so – hier gibt es konstant große verbotene Teilgraphen. Wir illustrieren das am einfachen Beispiel des Cluster Editing.

Leicht überlegt man sich, dass ein gegebener Graph $G = (V, E)$ eine disjunkte Vereinigung von Cliquen ist, genau dann wenn es keine drei Knoten $u, v, w \in V$ gibt, die einen Pfad (P_3) in G induzieren, d.h. einen Teilgraphen mit drei Knoten und zwei Kanten. Hiermit können also die Zielgraphen beim Cluster Editing durch verbotene Teilgraphen, nämlich P_3-Strukturen, charakterisiert werden. Eine einfache Suchbaumstrategie würde nun schlicht in drei Fälle verzweigen: Entweder eine der zwei vorhandenen Kanten löschen oder die dritte, fehlende Kante einfügen. Jeder dieser drei Kantenmodifikationen zerstört die verbotene Teilstruktur und mindestens einer der drei Fälle *muss* zu einer optimalen Lösung führen. Suchen wir eine Lösung der Größe k, so erhalten wir hiermit die Suchbaumgröße $O(3^k)$. Die obige Suchbaumstrategie könnte jedoch noch deutlich verbessert werden, und jüngste experimentelle Untersuchungen der Jenaer Bioinformatikgruppe (Prof. Böcker) bestätigen die praktische Nützlichkeit der Verbesserung. Insgesamt wird in der Dissertation folgender Satz bewiesen:

Theorem 4.1. CLUSTER EDITING *kann in* $O(2, 27^k + |V|^3)$ *Zeit gelöst werden.*

Abschließend sei hierzu noch erwähnt, dass eine rechnerbasierte Suche nach Verzweigungsstrategie die Suchbaumgröße noch von $O(2, 27^k)$ auf $O(1, 92^k)$ verbessern konnte [GGHN04]. In der Dissertation wurde auch noch ein allgemeineres Problem hinsichtlich verbotener Teilgraphen und Suchbaumstrategie erfolgreich untersucht: Closest 3-Leaf Power. In der Arbeit werden die grundlegenden Ideen skizziert und eine umfassende Beschreibung wurde parallel publiziert [DGHN06].

5 Stukturelle Parametrisierung

Es ist eine häufig zu machende Beobachtung, dass NP-schwere Probleme auf speziellen Eingabeinstanzen effizient gelöst werden können. Beispielsweise sind viele auf allgemeinen Graphen NP-schwere Probleme auf Bäume eingeschränkt leicht handhabbar. Es ist ein zentrales Anliegen der Algorithmik, ein tieferes Verständnis für die Ursachen der Schwierigkeit von Problemen zu gewinnen. Ein natürlicher Ansatz in diese Richtung ergibt sich auf folgende Weise: Finde einen „strukturellen Parameter“, der den Abstand einer gegebenen Instanz eines NP-schweren Problems von einer bekannter Maßen effizient lösbaren misst. Danach untersuche, ob das Problem bezüglich dieses Parameters FPT ist. Ist dem so, so hat man auf diese Weise eine klare „Ursache“ für die Problemschwierigkeit gefunden. Als Paradebeispiel möge hier das berühmte Konzept der Baumzerlegung eines Graphen dienen, welches mit dem Abstandsmaß Baumweite verknüpft ist. Ein Graph mit Baumweite 1 ist nichts anderes als ein Baum; ergo sind hierfür viele Probleme effizient lösbar. Der strukturelle Parameter Baumweite – eine Kernkonzept der algorithmischen Graphtheorie – misst also den Grad der Baumartigkeit eines Graphen und viele NP-schwere Probleme sind FPT bezüglich des Parameters Baumweite.

Die Dissertation versucht, diese Beobachtung unter dem Begriff „Abstand von einer Trivialität“ möglichst allgemein zu fassen. Das zugehörige Vorgehen gliedert sich in zwei Schritte:

1. Identifiziere trivial lösbare (in der Regel heißt das in Polynomzeit lösbare) Spezialfälle eines gegebenen Problems und bestimme ein gewisses Maß, das den Abstand einer beliebigen Eingabeinstanz zu einem solchen Spezialfall misst. Dieses Maß ist der strukturelle Parameter.

2. Entwerfe einen parametrisierten Algorithmus bezüglich des strukturellen Parameters.

Wir veranschaulichen diese Vorgehensweise durch Angabe eines parametrisierten Algorithmus für das NP-schwere Multicommodity Demand Flow in Trees-Problem (MDFT):

Eingabe: Ein Baum $T = (V, E)$, wobei jede Kante e eine positive ganze Zahl $c(e)$ als Kapazitätswert hat, und eine Liste von Knotenpaaren $F = \{f_i \mid f_i := (u_i, v_i), u_i \in V, v_i \in V, u_i \neq v_i\}$. Die Elemente $f \in F$ heißen „Fluss“ und sind verknüpft mit einer ganzzahligen Kapazitätsanforderung und einem reellwertigen Profitwert $p(f)$.
Aufgabe: Finde eine Teilmenge $F' \subseteq F$, so dass die Flüsse in F' alle „realisierbar“ sind und zugleich der Profit $\sum_{f \in F'} p(f)$ maximiert wird. Dabei heißt $F' \subseteq F$ realisierbar, falls die entsprechenden Flüsse mit ihren gegebenen Kapazitätsanforderungen alle gleichzeitig ohne eine Kapazitätsüberschreitung irgendeiner Kante durch das Baumnetzw-

erk T geschickt werden können.

MDFT hat viele Anwendungen, z.B. in der Telekommunikation oder beim Verkehr. Der von uns betrachtete strukturelle Parameter für MDFT ist $k := \max_{v \in V} |F_v|$, wobei F_v die Menge aller über den Knoten v führenden Flüsse aus F ist. Die entsprechende triviale Instanz mit $k = 1$ bedeutet, dass alle Flüsse disjunkt sind und MDFT auf simple Art gelöst werden kann. Der parametrisierte Algorithmus bezüglich des Parameters k beruht auf dynamischen Programmieren. In der Dissertation wurde so folgendes Ergebnis erzielt:

Theorem 5.1. *MDFT kann in $O(2^k \cdot |F| \cdot |V|)$ Zeit gelöst werden.*

Ein weiteres Ergebnis der Dissertation zu struktureller Parametrisierung beruht ebenso auf dynamischer Programmierung. Es zeigt, dass das NP-schwere Problem Weighted Multicut in Trees FPT ist. Hierbei ist die Eingabe ein Baum $T = (V, E)$ mit positiv reellwertigen Kantengewichten und eine Menge F von Knotenpaaren, und die Aufgabe ist eine Kantenmenge $E' \subseteq E$ mit kleinstmöglichem Gesamtgewicht so zu finden, dass alle Verbindungspfade zwischen den Knotenpaaren in F durch Löschung der Kanten in E' unterbrochen werden.

6 Zusammenfassung und Ausblick

Graphmodifikationsprobleme spielen wichtige Rolle in vielen Anwendungsfelden. Meist sind sie NP-schwer. Der Ansatz der parametrisierten Algorithmik ist besonders für solchlerlei Probleme realistisch und vielversprechend. Die Dissertation bietet eine erste systematische Aufarbeitung der Techniken des parametrisierten Algorithmenentwurfs für Graphmodifikationsprobleme. Die Arbeit stellt sowohl Ansatzpunkte für den Algorithmenentwurf als auch konkrete Einzelergebnisse für wichtige Graphmodifikationsprobleme vor.

Wie bereits erwähnt, wurden die meisten Ergebnisse der Arbeit bereits von diversen internationalen Arbeitsgruppen aufgegriffen, sowohl von theoretischer als auch von praktischer Seite aus. Auch eigene Forschung führte nach Abschluss der Promotion zu wichtigen weiteren Erkenntnissen, am bemerkenswerten hier ist vielleicht der lineare Problemkern für Cluster Editing [Guo07]. Die große Zahl verschiedener praxisrelevanter Graphmodifikationsprobleme verspricht zahlreiche interessante Herausforderungen für zukünftige Forschungen. Vielleicht kann diese Arbeit als Einstiegshilfe hierzu dienen.

References

[BBC04] N. Bansal, A. Blum und S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

[BBD06] P. Burzyn, F. Bonomo und G. Durán. NP-completeness results for edge modification problems. *Discrete Applied Mathematics*, 154(13):1824–1844, 2006.

[Bix02] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50:3–15, 2002.

[DF99] R. G. Downey und M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

[DGHN06] M. Dom, J. Guo, F. Hüffner und R. Niedermeier. Error compensation in leaf power problems *Algorithmica*, 4(44):363–381, 2006.

[DLL+06] F. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw und Y. Zhang. The Cluster Editing problem: implementations and experiments. In *Proc. 2nd IWPEC*, Band 4196 in LNCS, Seiten 13–24. Springer, 2006.

[FLRS06] M. R. Fellows, M. A. Langston, F. Rosamond und P. Shaw. Polynomial-time linear kernelization for Cluster Editing. Manuskript zur Veröffentlichung eingereicht, 2006.

[GGH+06] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier und S. Wernicke. Compression-based fixed-parameter algorithms for Feedback Vertex Set and Edge Bipartization. *Journal of Computer and System Sciences*, 72(8):1386–1396, 2006.

[GGHN04] J. Gramm, J. Guo, F. Hüffner und R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.

[GJ79] M. R. Garey und D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[Guo07] J. Guo. A more effective linear kernelization for Cluster Editing. In *Proc. 1st ESCAPE*, LNCS, Springer, Apr. 2007.

[Har69] F. Harary. *Graph Theory*. Addison-Wesley, 1969.

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

[PdSS06] F. Protti, M. D. da Silva und J. L. Szwarcfiter. Applying modular decomposition to parameterized bicluster editing. In *Proc. 2nd IWPEC*, Band 4169 in LNCS, Seiten 1–12. Springer, 2006.

[Rob89] F. Roberts, Hrsg. *Applications of Combinatorics and Graph Theory to the Biological and Social Sciences*. Springer, 1989.

[SST04] R. Shamir, R. Sharan und D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004.

Jiong Guo wurde am 28. November 1970 in Chengdu, V. R. China, geboren, wo er 1988 mit Abschluss Hochschulreife abging. Von September 1988 bis Juli 1992 studierte er Informatik an der University of Electronic Science and Technology of China in Chengdu. Das Studium schloss er mit einem Bachelor of Science im Juli 1992 ab. Anschließend arbeitete er bis September 1995 bei der Industrial and Commercial Bank of China in Chengdu als Softwareentwickler. Danach war er als Softwareentwickler in der Guotai Securities Ltd. in Chengdu beschäftigt. Im August 1996 kam er nach Deutschland und studierte, nach dem Erlernen der deutschen Sprache, von Oktober 1996 bis Februar 2002 Informatik mit dem Nebenfach Betriebswirtschaftslehre an der Universität Tübingen. Im Februar 2002 verlieh ihm die Universität Tübingen ein Diplom in Informatik (Prädikat: „sehr gut“). Anschließend arbeitete er bis September 2004 als wissenschaftlicher Angestellter im Fach Informatik an der Universität Tübingen. Im Oktober 2004 wechselte er an die Universität Jena. Im Februar 2006 schloss er seine Promotion mit Auszeichnung in Jena ab und ist dort seither als PostDoc tätig.