Parametrisierte Ansätze für schwere Graphprobleme: Algorithmen und Experimente

Falk Hüffner*

School of Computer Science
Tel Aviv University
Tel Aviv 69978, Israel
hueffner@tau.ac.il

Abstract: Die Dissertation „Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems“ befasst sich mit Entwurf, Analyse, Implementierung und experimenteller Bewertung von Algorithmen für NP-schwere Graphprobleme. Das Ziel ist es zu belegen, dass das Konzept der *parametrisierten Komplexität*, und insbesondere neuartige algorithmische Techniken, deren Entwicklung auf dieses Konzept zurückgeht, zu einsetzbaren Programmen für die exakte Lösung von Praxisinstanzen führt. Insbesondere untersuchen wir die drei Graphprobleme CLIQUE COVER, BALANCED SUBGRAPH und MINIMUM-WEIGHT PATH, die Anwendungen in der Bioinformatik und anderen Gebieten haben. Zum Einsatz kommen die Techniken Datenreduktion, Iterative Kompression und Farbkodierung. Wir entwickeln eine Reihe neuer Festparameteralgorithmen und geben Implementierungen für vier wichtige Probleme, die nach unserem Stand des Wissens die jeweils schnellsten Löser sind.

1 Einleitung

Viele kombinatorische Probleme aus der Praxis sind *NP-schwer* [GJ79]. Ein Beispiel ist die folgende Aufgabe: Nach einer Reihe von Experimenten wird festgestellt, dass einige Paare von Experimenten widersprüchliche Resultate haben. Wir wissen daher, dass von jedem solchen Paar mindestens ein Experiment fehlerhaft war. Wir wollen also eine kleinstmögliche Anzahl von Experimenten auslassen, sodass es keine Konflikte mehr gibt. Dieses Problem, in abstrakter Form als Graphproblem, ist als VERTEX COVER bekannt: gegeben einen Graphen, lösche eine kleinstmögliche Anzahl Knoten, sodass alle Kanten verschwinden. Es ist inzwischen eine gängige Annahme, dass NP-Schwere eine inhärente kombinatorische Explosion im Lösungsraum impliziert, die zu Laufzeiten führt, die exponentiell mit der Eingabegröße wachsen. Dies bedeutet, dass große Instanzen nicht immer optimal gelöst werden können.

Es gibt mehrere Ansätze, das Problem der NP-Schwere in der Praxis zu umgehen. *Heuristiken* lassen die Forderung nach nützliche Laufzeitgarantien oder Qualitätsgarantien

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

fallen und werden darauf getrimmt, für typische Instanzen schnell gute Ergebnisse zu liefern [MF05]. *Approximationalgorithmen* verzichten auf die Forderung nach Optimalität zugunsten von beweisbar effizienter Laufzeit bei gleichzeitig beweisbar guter Lösungsqualität [ACG+99]. Diese Methoden haben jedoch schwerwiegende Nachteile. In vielen Fällen ist es nicht akzeptabel, dass ein Algorithmus in Grenzfällen extrem lange läuft; und die erzielbaren Approximationsgarantien sind meist recht schwach (eine Garantie wie maximal 10% Fehler ist oft nicht erreichbar [ACG+99]). Parametrisierte Komplexität [DF99, FG06, Nie06] ist ein neuerer Ansatz, der in vielen Fällen einen Ausweg bietet, bei dem man weder auf Optimalität noch auf nützliche Laufzeitschranken verzichten muss.

Parametrisierte Komplexität und Festparameteralgorithmen. Die zentrale Beobachtung, die parametrisierte Komplexität motiviert, ist, dass oft selbst sehr große Praxisinstanzen NP-schwerer Probleme einfach sind. Der Grund ist, dass sie Strukturen enthalten, die ausgenutzt werden können. Die Idee ist, die strukturelle Komplexität mit einem *Parameter* zu messen, der üblicherweise eine positive Zahl k ist. Wir können dann eine zweidimensionale Komplexitätsanalyse vornehmen, bei der das Wachstum der Laufzeit sowohl mit der Eingabegröße n als auch mit k beschrieben wird. Die Hoffnung ist, dass man die kombinatorische Explosion auf den Parameter einschränken kann, sodass Instanzen schnell lösbar sind, solange der Parameter klein ist.

Betrachten wir als Beispiel noch einmal die Erkennung fehlerhafter Experimente (VERTEX COVER). Es ist vernünftig, anzunehmen, dass nur wenige Experimente fehlerhaft sind, denn anderenfalls ist die gesamte Serie unbrauchbar. Wenn wir also die kombinatorische Explosion auf die Anzahl fehlerhafter Experimente k beschränken können, werden selbst große Instanzen lösbar. Dieses Konzept wird formalisiert in der folgenden Definition von Downey und Fellows [DF99].

Definition 1. *Ein parametrisiertes Problem mit Eingabegröße n und Parameter k ist* festparameter-handhabbar*, wenn es einen Algorithmus gibt, der das Problem in $f(k) \cdot n^{O(1)}$ Zeit löst, wobei f eine beliebige Funktion ist, die nur von k abhängt.*

Man beachte, dass Festparameter-Handhabbarkeit eine stärkere Aussage ist als Polynomzeitlösbarkeit für jedes k; ein Algorithmus mit Laufzeit $O(n^k)$ demonstriert, dass ein Problem für jedes k in Polynomzeit lösbar ist, zeigt aber nicht, dass es festparameterhandhabbar ist.

Als Beispiel für einen *Festparameteralgorithmus* betrachten wir VERTEX COVER: Es lässt sich mit einem einfachen Suchbaumalgorithmus lösen, bei dem man für eine Kante $\{a,b\}$ in die zwei Fälle "a löschen" und "b löschen" verzweigt. Dieser Algorithmus hat eine Laufzeit von $O(2^k n)$. Somit können wir auch recht große Instanzen mit z. B. $k = 30$ und $n = 1000$ in akzeptabler Zeit lösen. Es gibt Algorithmen, die VERTEX COVER in $O(1{,}3^k + kn)$ Zeit lösen [Nie06], und somit sogar noch Instanzen mit $k = 120$ und $n = 1000$ lösen können.

Das klassische Buch über parametrisierte Komplexität ist von Downey und Fellows [DF99]. Zwei neuere Bücher legen Schwerpunkte auf algorithmische Aspekte [Nie06] und Komplexitätstheorie [FG06].

Parametrisierte Komplexität ist also ein vielversprechender Ansatz zur Behandlung NP-

schwerer Probleme. Man kann sogar behaupten, dass er sich schon vor seiner Formulierung bewährt hat, denn parametrisierte Komplexität kann den Erfolg vieler in der Praxis eingesetzter Verfahren erklären. Ein gutes Beispiel dafür ist Typinferenz in Compilern für die Programmiersprache ML. Obwohl diese Aufgabe EXPTIME-vollständig ist (und somit noch schwerer als etwa VERTEX COVER), kann der Standardalgorithmus problemlos auch größere Eingaben verarbeiten. Die Erklärung ist, dass der Algorithmus ein Festparameteralgorithmus ist, wobei der Parameter die Verschachtelungstiefe von *let*-Konstrukten ist, die bei normalen Programmen sehr klein ist [DF99].

Es gab jedoch bisher kaum Veröffentlichungen, die sich systematisch mit der Erforschung der praktischen Umsetzung der neuen Ideen befassen. Ziel meiner Arbeit ist es zu belegen, dass das Konzept der parametrisierten Komplexität, und insbesondere auch neuartige algorithmische Techniken, deren Entwicklung auf dieses Konzept zurückgeht, tatsächlich zu einsetzbaren Programmen für die exakte Lösung von Praxisinstanzen führt.

Algorithm Engineering. In letzter Zeit gibt es eine wachsende Kluft zwischen der Algorithmentheorie und Algorithmen, die tatsächlich in der Praxis eingesetzt werden. Grund sind Unterschiede in Methodik und Zielen. In der Algorithmentheorie werden einfache abstrakte Probleme und einfache abstrakte Maschinenmodelle betrachtet. Die Lösungen sind ausgefeilt, und das Hauptziel sind beweisbare Leistungsgarantien. In der Praxis muss man mit komplexen Anforderungen und realen Maschinen umgehen. Im ungünstigsten Fall führen diese Unterschiede in der Sichtweise zu unbrauchbaren Lösungen von den Theoretikern und zu schlecht skalierenden, unzuverlässigen Lösungen von den Praktikern.

Ziel von *Algorithm Engineering*, dem jüngst das DFG-Schwerpunktprogramm 1307 gewidmet wurde, ist es, diese Kluft zu überbrücken durch eine grundlegend veränderte Methodik der Algorithmenforschung. Klassische Algorithmentheorie arbeitet linear: Aus der Anwendung wird ein abstraktes Modell destilliert, und ein Algorithmus wird entworfen. Dieser Algorithmus wird mathematisch analysiert, was Leistungsgarantien ergibt. An diesem Punkt wird die Arbeit oft als erledigt angesehen; Implementierung und Anwendung wird oft erst nachträglich einbezogen, oder von anderen Gruppen behandelt. Anfänglich ist der Ansatz des Algorithm Engineering gleich: Modell, Algorithmusentwurf und Analyse. Danach wird die Implementierung und experimentelle Bewertung aber nicht als optional angesehen, sondern als integraler Bestandteil des Prozesses. Insbesondere sind echte Eingaben aus der Anwendung zu benutzen. Der Hauptunterschied zur klassischen Algorithmentheorie liegt darin, dass die Resultate der Experimente in einem Kreislauf den Entwurf verbesserter Algorithmen beeinflusst.

Ein erfolgreiches Beispiel von Algorithm Engineering ist Routenplanung in Straßennetzwerken. So benutzten Bast et al. [BFSS07] die Beobachtung, dass man für längere Strecken die Gegend des Startpunkts immer über eine von wenigen wichtigen Kreuzungen verlässt, was in ihren Experimenten Suchen ermöglicht, die bis zu sechs Größenordnungen schneller sind, als mit Dijkstras Algorithmus.

In meiner Arbeit habe ich insbesondere versucht, auf mehreren Gebieten dem Leitfaden des Algorithm Engineering folgend Forschung anzustoßen. Das vielleicht beste Beispiel ist Farbkodierung (Abschnitt 4): Hier wurden für reale Eingaben die Laufzeiten um mehrere

Größenordnungen verbessert, und eine benutzerfreundliche graphische Oberfläche wurde zur Verfügung gestellt. Da Algorithm Engineering jedoch die klassische Algorithmentheorie umfasst und erweitert, ist sie ein ambitioniertes Unternehmen, das normalerweise die Kooperation mehrerer Parteien erfordert; deswegen sollten einige Teile der Arbeit als Einladung für weitere, dem Konzept des Algorithm Engineering folgende Forschungen gesehen werden.

2 Datenreduktion

Datenreduktion ist eine klassische Methode, mit schweren Problemen umzugehen: bevor der eigentliche Lösungsprozess gestartet wird, versucht man, die Größe der Eingabe zu verringern, indem man Teile entfernt oder vereinfacht, zum Beispiel weil man sofort sehen kann, dass sie irrelevant für die Lösung sind. Aufwendige Lösungsalgorithmen brauchen dann nur auf den verbliebenen „harten Kern“ der Instanz angewendet zu werden. Als Beispiel erwähnt Bixby [Bix02] ein großes ganzzahliges lineares Programm, das mit Datenreduktion in einer halben Stunde gelöst werden kann, jedoch ohne Datenreduktion weit davon entfernt ist, praktisch lösbar zu sein.

Einige Datenreduktionsregeln sind einfach und werden leicht entdeckt von jedem, der ein Problem angeht. Andere erfordern tiefere Einsichten in die kombinatorische Struktur des Problems. Sobald eine effektive Reduktionsregel gefunden wird, sollte sie in praktisch jedem Kontext zum Einsatz kommen, sei er heuristisch, approximativ oder exakt.

Datenreduktion wurde bisher meist als heuristische Aufgabe gesehen, denn in der klassischen eindimensionalen Komplexitätstheorie kann man nichts über die Qualität der Datenreduktion aussagen. Parametrisierte Komplexität hingegen stellt zur Einschätzung der Wirkung einer Datenreduktionsregel das Konzept eines *Problemkerns* bereit [DF99, FG06, Nie06, GN07]. Eine Reduktion auf einen Problemkern ist eine in Polynomzeit laufende Datenreduktion, die eine Instanz so verkleinert, dass ihre Größe nur noch vom Parameter k abhängt, und nicht mehr von der ursprünglichen Eingabegröße n. Auf diese Weise ermöglicht das Problemkernkonzept einen fruchtbaren Dialog zwischen Praktikern und Theoretikern: Problemkerne können erklären, warum Reduktionsregeln in der Praxis so gut funktionieren; und die Suche nach Problemkernen kann zu neuen, mächtigen Datenreduktionsregeln führen, die auf tiefen strukturellen Einsichten beruhen. Insbesondere sind Problemkerne auch nützlich, wenn der Parameter nicht klein ist, da sie in Polynomzeit laufen.

Datenreduktion für Clique Cover. Beim CLIQUE COVER-Problem ist die Aufgabe, gegeben einen Graphen, eine kleinstmögliche Menge von Cliquen (vollständigen Teilgraphen) zu finden, die alle Kanten abdeckt. Es hat zahlreiche Anwendungen in der Compileroptimierung, bei geometrischen Berechnungsproblemen und in der angewandten Statistik. Abbildung 1 zeigt ein Beispiel. In diesem Graphen sind zwei Schauspieler miteinander verbunden, wenn sie zusammen in einem Film gespielt haben. Ein Clique Cover (gestrichelt) gibt die einfachste Erklärung: es handelt sich um drei verschiedene Filme. Auf diese

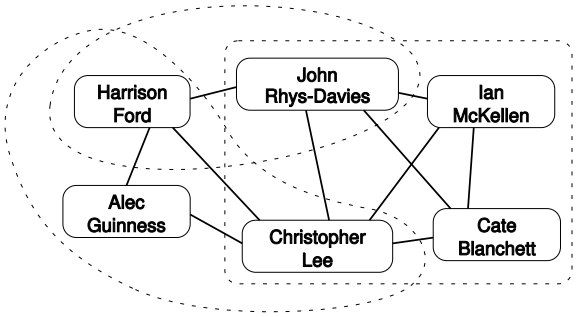


Abbildung 1: Filmgraph

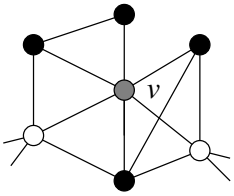


Abbildung 2: Partition der Nachbarn von v. Die zwei weißen Knoten sind Ausgänge, und die schwarzen Knoten sind Gefangene. Da jeder Ausgang einen Gefangenen als Nachbarn hat, ist Reduktionsregel 1 anwendbar, und v kann gelöscht werden.

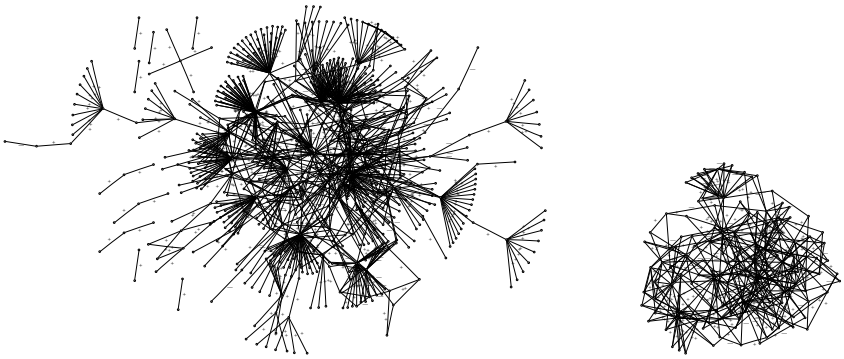
Weise kann man mithilfe von CLIQUE COVER verborgene Strukturen in einem Graphen aufdecken.

CLIQUE COVER ist NP-schwer, und es gibt keine praktisch brauchbaren Approximationsgarantien; deshalb sind bisher eingesetzte Verfahren meist heuristisch. Wir geben eine Reihe von Reduktionsregeln für CLIQUE COVER an, die es ermöglichen, zusammen mit einem simplen Brute-Force-Verfahren viele Praxisinstanzen in wenigen Sekunden zu lösen. Eine einfache Regel besagt, isolierte Knoten (also Knoten ohne Nachbarn) zu löschen. Hier ist die Korrektheit noch offensichtlich. Als Beispiel für eine aufwändigere Reduktionsregel sei die folgende genannt (Abbildung 2).

Reduktionsregel 1. *Sei $N(v)$ die Menge der Nachbarknoten von v. Partitioniere die Nachbarschaft eines Knoten v in* Gefangene *p mit $N(p) \subseteq N(v)$ und* Ausgänge *x mit $N(x) \setminus N(v) \neq \emptyset$. Wenn alle Ausgänge mindestens einen Gefangenen als Nachbarn haben, dann lösche v.*

Der Grund für die Korrektheit dieser weniger offensichtlichen Regel ist, dass v zu jeder Clique hinzugefügt werden kann, die einen Gefangenen enthält, und dass auf diese Weise alle an v angrenzenden Kanten abgedeckt werden können. Mit den beiden genannten Regeln ergibt sich das folgende Resultat.

Theorem 1. *Nach Anwendung der Reduktionsregeln hat die* CLIQUE COVER*-Instanz höchstens 2^k Knoten, d. h.* CLIQUE COVER *hat einen Problemkern.*



$n = 690$, $m = 1082$ $n = 144$, $m = 405$

Abbildung 3: Datenreduktion im Hefe-Proteininteraktionsnetzwerk. Links: Originalgraph; rechts: nach Datenreduktion.

Die gefundenen Datenreduktionsregeln wurden zusammen mit einem simplen Brute-Force-Algorithmus auf verschiedenen Instanzen getestet. Von den Testinstanzen aus der Visualisierung von Getreideertragsdaten mit bis zu 124 Knoten und 4 847 Kanten konnten alle in unter einer Sekunde gelöst werden. Mit einem Modell, dass es erlaubt, größere solcher Instanzen zu simulieren, konnten in unter 20 Sekunden noch Instanzen mit bis zu 300 Knoten gelöst werden. Dies deckt alle für diese Anwendung praxisrelevanten Größen ab. Bei Zufallsgraphen konnten selbst Graphen mit 10 000 Knoten noch in 7 min gelöst werden, wenn es relativ wenige Kanten gibt ($m \approx n \ln n$).

Datenreduktion für Balanced Subgraph. Ein Graph mit Kantenbeschriftungen $=$ oder $\neq$ ist *balanciert*, wenn seine Knoten mit zwei Farben so eingefärbt werden können, dass die Relation entlang jeder Kante eingehalten wird (also z. B. bei einer "$=$"-Kante die Endpunkte dieselbe Farbe bekommen). Balancierte Graphen sind eine Verallgemeinerung von bipartiten Graphen. Das NP-schwere BALANCED SUBGRAPH-Problem besteht nun darin, von einem gegebenen Graphen möglichst wenig Kanten zu löschen, sodass er balanciert wird. Es hat Anwendungen in so verschiedenen Gebieten wie Analyse von sozialen Netzwerken, Systembiologie, Festkörperphysik und integriertem Schaltungsentwurf.

Wir zeigen eine neuartige Datenreduktion für BALANCED SUBGRAPH basierend auf kleinen Graphseparatoren und einem neuen Konstruktionsschema für verkleinerte äquivalente Graphkomponenten. Das Schema vereinheitlicht und verallgemeinert eine Vielzahl existierender Reduktionsregeln. Es kann prinzipiell für eine Reihe von Graphproblemen angewendet werden, bei denen eine Färbung oder eine Teilmenge der Knoten gesucht wird. Abbildung 3 zeigt ein Beispiel für die Effektivität der Datenreduktion: nur ca. 21% der Knoten verbleiben. Diese Instanz ist jedoch immer noch zu groß, um sie mit einem naiven Ansatz zu lösen; wir setzen dafür iterative Kompression (Abschnitt 3) ein.

Unsere Implementierung kann eine Reihe biologische Praxisinstanzen exakt lösen, für die bisher nur Approximationen bekannt waren [DESZ07]. Ein Netzwerk mit 688 Knoten und 2 208 Kanten konnte jedoch nicht gelöst werden.

3 Iterative Kompression

Iterative Kompression ist ein recht neuer Ansatz zur Lösung von NP-schweren Problemen [RSV04]. Die grundlegende Idee – Induktion – ist jedoch recht alt: Wir nehmen an, dass wir ein Teilproblem bereits gelöst haben. Dies kann man ohne Beschränkung der Allgemeinheit tun, weil man auf diese Weise irgendwann bei trivialen Instanzen landet, die sich einfach lösen lassen. Iterative Kompression lässt sich (wie bereits erwähnt) auf BALANCED SUBGRAPH anwenden. Hier nehmen wir an, dass wir für die Eingabeinstanz minus eine Kante e bereits eine Lösung X' haben. Dann ist $X' \cup \{e\}$ offensichtlich eine Lösung für unsere Instanz – nur eventuell keine optimale. Aufgabe ist es also, diese Lösung zu komprimieren, und die Kunst liegt darin, dies möglichst effizient zu tun.

Mithilfe von iterativer Kompression entwerfen wir die schnellsten zur Zeit bekannten Festparameteralgorithmen für eine Reihe von Problemen. Das erste betrachtete Problem ist CLUSTER VERTEX DELETION, das Problem, eine minimale Anzahl von Knoten aus einem Graphen zu löschen, sodass danach jede Zusammenhangskomponente eine Clique ist. Wir geben einen Algorithmus mit Laufzeit $O(2^k \cdot k^9 + n^3)$ an, wobei n die Anzahl Knoten im Eingabegraphen und k die Lösungsgröße ist. Dies ist die erste nichttriviale Anwendung von iterativer Kompression für ein Problem, das kein Kreiszerstörungsproblem ist, und eine der ersten Anwendungen, die auch mit gewichteten Daten umgehen können. Als nächstes betrachten wir das FEEDBACK VERTEX SET-Problem in Turniergraphen, also das Problem, einen Turniergraphen durch Knotenlöschungen azyklisch zu machen, und erhalten eine Laufzeit von $O(2^k \cdot n^2(\log n + k))$. Im Kern benutzt der Algorithmus auf überraschende Weise eine Prozedur zur Bestimmung einer längsten gemeinsamen Teilzeichenkette.

Wir zeigen weiter, dass das EDGE BIPARTIZATION-Problem festparameter-handhabbar ist, indem wir einen auf iterativer Kompression beruhenden Algorithmus mit Laufzeit $O(2^k \cdot m^2)$ angeben, wobei m die Anzahl der Kanten im Eingabegraphen ist. Dieses Resultat kann auf BALANCED SUBGRAPH erweitert werden. Desweiteren entwickeln wir einige heuristische Beschleunigungen, die sich in Experimenten als sehr effektiv herausstellen. Als nächstes geben wir einen neuartigen Beweis für einen auf iterativer Kompression beruhenden Algorithmus für VERTEX BIPARTIZATION und verbessern die Laufzeit um einen Faktor von k auf $O(3^k \cdot mn)$. Mit einer heuristischen Beschleunigung kann unsere Implementierung alle Instanzen einer Problemsammlung aus der Bioinformatik in Minuten lösen, während konventionelle Methoden nur etwa die Hälfte der Instanzen überhaupt in akzeptabler Zeit lösen können. Weitere Experimente mit generierten und zufälligen Eingaben werden gezeigt.

4 Farbkodierung

Farbkodierung ist eine randomisierte Methode zum Finden kleiner Teilgraphen der Größe k in einem Graph [AYZ95]. Sie wurde bereits benutzt, um Kandidaten für Signalpfade in Proteininteraktionsnetzwerken zu finden [SIKS06], also eine Folge unterschiedlicher Proteine, bei der jedes mit dem vorigen stark wechselwirkt. Dies kann als das NP-schwere

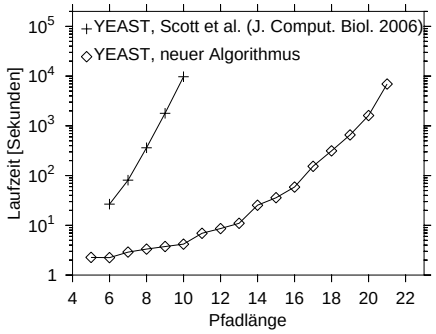


Abbildung 4: Vergleich der Laufzeiten für das Hefenetzwerk von der Implementation von Scott et al. [SIKS06] und unserem Programm

MINIMUM-WEIGHT PATH-Problem modelliert werden: finde in einem kantengewichteten Graphen einen einfachen Pfad einer gegebenen Länge mit minimalem Gewicht. Beispielsweise suchen wir im Proteininteraktionsnetzwerk der Hefe mit 4 400 Knoten und 14 300 Kanten nach Pfaden der Länge $k = 5$ bis 15.

Bei Farbkodierung wird jeder Knoten des Graphen zufällig mit einer von k Farben eingefärbt. Dabei hofft man, dass die k Knoten im optimalen Pfad alle unterschiedliche Farben erhalten (der Pfad *bunt ist*), denn unter dieser Annahme lässt sich mittels dynamischen Programmierens MINIMUM-WEIGHT PATH erheblich schneller lösen. Da dies aber meist nicht der Fall ist, wiederholt man diese Versuche, bis mit hoher Wahrscheinlichkeit mindestens einmal der Pfad tatsächlich bunt war. Das Verfahren liefert deswegen einen Festparameteralgorithmus, weil sowohl die Laufzeit der Suche nach einem bunten Pfad als auch die Anzahl Versuchswiederholungen, um eine vorgegebene Fehlerwahrscheinlichkeit ε zu erreichen, im exponentiellen Anteil nur von k abhängen.

Zunächst verbessern wir die Laufzeit der Farbkodierungsmethode für MINIMUM-WEIGHT PATH von $O(5.44^k \cdot m \cdot (-\ln \varepsilon))$ auf $O(4.32^k \cdot m \cdot (-\ln \varepsilon))$. Die Idee ist, anstelle von k Farben $k+x$ zu verwenden. Zwar wird dadurch jeder einzelne Versuch langsamer, aber man braucht weniger Versuche, weil es wahrscheinlicher ist, dass ein Pfad bunt wird. Für $x \approx 0.3k$ ergibt sich ein Optimum der Laufzeit. Eine weiter Verbesserung ergibt sich dadurch, dass man das dynamische Programmieren als Zustandsraumsuche auffasst (also als Kürzeste-Wege-Problem in einem implizit definierten Graphen). Mit heuristische Evaluierungsfunktionen (einer Technik aus der Künstlichen Intelligenz) ist es dann möglich, den Suchraum einzuschränken und die Suche zu leiten (z. B. mit dem A*-Algorithmus).

Unsere Implementierung, die sorgfältig abgestimmte Datenstrukturen benutzt, zeigt eine Beschleunigung von mehreren Größenordnungen gegenüber vorherigen Ergebnissen (Abbildung 4). Insbesondere kann sie für praktisch alle für die Suche nach Signalpfaden relevanten Parametereinstellungen Resultate in Sekunden liefern, was die interaktive Untersuchung solcher Pfade ermöglicht. Für diese Aufgabe haben wir die graphische Benutzeroberfläche FASPAD (Abbildung 5) entwickelt. Hier kann der Benutzer zu einem beliebigen Netzwerk und vorgegebener Pfadlänge die besten Pfadkandidaten finden und graphisch anzeigen

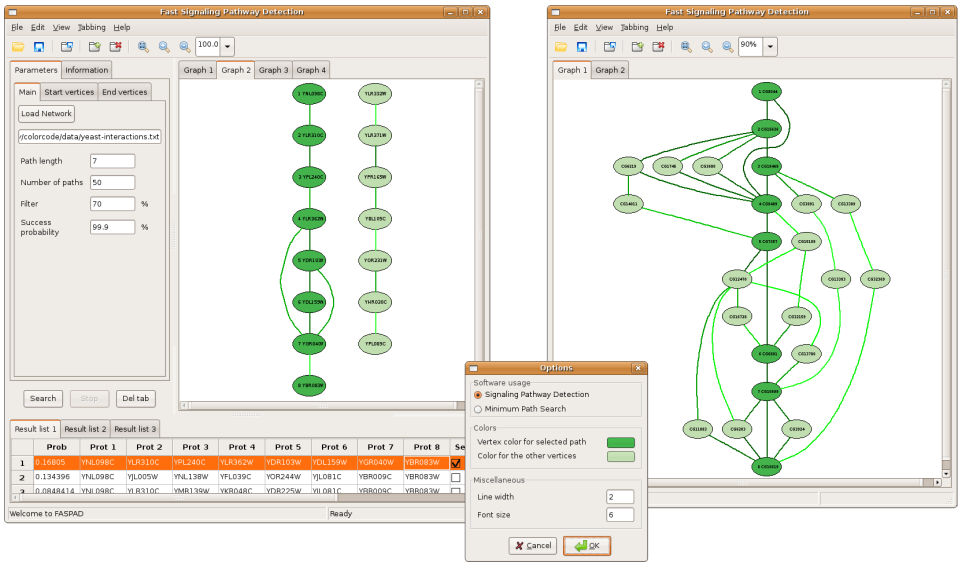


Abbildung 5: Links: Zwei Signalpfad-Kandidaten im Proteininteraktionsnetzwerk des Hefepilzes, die von FASPAD gefunden wurden. Sie entsprechen dem Zellwandintegritätspfad und dem Pfad für faserartigen Wuchs, die aus der Literatur bekannt sind. Rechts: Pfad im Netzwerk der Fruchtfliege mit Kontext.

lassen, wobei auch Überlappungen der Pfade und ihre Umgebung im Netzwerk angezeigt werden können.

5 Zusammenfassung

Wir haben eine Reihe neuer Festparameteralgorithmen entwickelt und Implementierungen für vier Probleme angegeben, die nach unserem Stand des Wissens die jeweils schnellsten Löser sind. Dies unterstreicht die praktische Brauchbarkeit des parametrisierten Ansatzes, und insbesondere seine Nützlichkeit als Leitfaden für die Entwicklung neuartiger algorithmischer Techniken wie iterativer Kompression oder Farbkodierung. Basierend auf diesen Erkenntnissen findet sich in der Dissertation eine Schritt-für-Schritt-Anleitung, wie man mit den Methoden der parametrisierten Komplexität effektiv zu optimalen Ergebnissen für NP-schwere Probleme kommt.

Die Implementierungen für CLIQUE COVER, BALANCED SUBGRAPH, VERTEX BIPARTIZATION und MINIMUM-WEIGHT PATH sind als freie Software unter der *GNU Public License* erhältlich bei http://theinf1.informatik.uni-jena.de/~hueffner/.

Literatur

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi et al. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer, 1999.

[AYZ95] N. Alon, R. Yuster und U. Zwick. Color-coding. *Journal of the ACM*, 42(4):844–856, 1995.

[BFSS07] H. Bast, S. Funke, P. Sanders und D. Schultes. Fast Routing in Road Networks with Transit Nodes. *Science*, 316(5824):566, 2007.

[Bix02] R. E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[DESZ07] B. DasGupta, G. A. Enciso, E. D. Sontag und Y. Zhang. Algorithmic and Complexity Results for Decompositions of Biological Networks into Monotone Subsystems. *Biosystems*, 90(1):161–178, 2007.

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

[FG06] J. Flum und M. Grohe. *Parameterized Complexity Theory*. Springer, 2006.

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

[GN07] J. Guo und R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[MF05] Z. Michalewicz und D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, 2005.

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[RSV04] B. Reed, K. Smith und A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[SIKS06] J. Scott, T. Ideker, R. M. Karp und R. Sharan. Efficient Algorithms for Detecting Signaling Pathways in Protein Interaction Networks. *Journal of Computational Biology*, 13(2):133–144, 2006.

Falk Hüffner wurde am 25. Februar 1976 in Oldenburg geboren und erwarb 1995 das Abitur am Lothar-Meyer-Gymnasium Varel. Nach seinem Zivildienst am Institut für Vogelforschung in Wilhelmshaven studierte er von Oktober 1996 bis März 2004 Informatik mit Nebenfach Mathematik an der Eberhard-Karls-Universität Tübingen. Nachdem er dort zunächst auch als wissenschaftlicher Angestellter im Fach Informatik bei der Gruppe von Dr. Rolf Niedermeier begann, wechselte er zusammen mit der Gruppe im September 2004 an die Friedrich-Schiller-Universität Jena. Im Dezember 2007 schloss er dort seine Promotion ab und arbeitete bis März 2008 als Postdoc. Seitdem ist er als Postdoc an der Tel Aviv University im Bereich Computational Genomics in der Gruppe von Prof. Ron Shamir tätig. Neben der algorithmischen Forschung engagiert sich Falk Hüffner bei Free-Software-Projekten wie dem Debian-Projekt oder dem GNU-Compiler gcc.