Schnelle Algorithmen für zerlegbare Graphen*

Alexander Langer

Lehr- und Forschungsgebiet Theoretische Informatik
RWTH Aachen University
langer@cs.rwth-aachen.de

Abstract: Ein berühmter Satz von Courcelle besagt, dass jedes MSO-definierbare Problem auf Graphen beschränkter Baumweite in Linearzeit gelöst werden kann. Obwohl dies erklärt, warum sich viele Graphprobleme auf solchen Graphen effizient lösen lassen, scheitern in der praktischen Umsetzung von Courcelles Satz die herkömmlichen Verfahren am benötigten Speicherbedarf, der sehr schnell die Grenzen dessen übersteigt, was wir in der Praxis handhaben können. In dieser Arbeit stellen wir einen neuen Ansatz vor, der auf modelltheoretischen Spielen basiert. Mit Hilfe dynamischer Programmierung auf der Baumzerlegung wird in Linearzeit ein Spiel aufgebaut, in welchem sich schnell testen lässt, ob die Formel auf dem Eingabegraphen erfüllt ist. Eine Reihe von Experimenten legt nahe, dass unser Ansatz für manche Problemstellungen als Alternative zur ganzzahligen linearen Optimierung in Betracht kommt.

1 Einführung

Im Jahr 1990 hat Courcelle bewiesen [Cou90], dass jedes Graphproblem, das sich in Monadischer Prädikatenlogik zweiter Stufe (MSO) ausdrücken lässt, auf Graphen mit beschränkter Baumweite in Linearzeit lösbar ist. Dieses Ergebnis wurde wenig später auf viele weitere MSO-definierbare Probleme erweitert, darunter eine große Klasse von Aufzählungs- und Optimierungsproblemen [ALS91, CM93]. Beispiele sind etwa die klassischen Probleme VERTEX COVER, INDEPENDENT SET, DOMINATING SET oder 3-COLORABILITY.

Die Baumweite [Hal76, RS86] eines Graphens ist eine ganzzahlige Zahl, die im Wesentlichen beschreibt, wie ähnlich ein Graph einem Baum ist: Bäume und Wälder haben Baumweite eins, Gitter der Größe $n \times n$ haben Baumweite n; ein vollständiger Graph mit n Knoten hat Baumweite $n - 1$. Ein Graph mit Baumweite k lässt sich rekursiv in kleinere Graphen zerlegen, die ebenfalls nur Baumweite k haben. Eine solche *Baumzerlegung* kann algorithmisch berechnet werden. Probleme lassen sich häufig effizient lösen, wenn wir die in der Baumzerlegung beschriebene Struktur ausnutzen und dynamische Programmierung verwenden. Viele Instanzen aus der täglichen Praxis haben eine kleine Baumweite. Beispielsweise ist die Baumweite der Kontrollflussgraphen von C-Programmen ohne `goto` höchstens sechs [Tho98]; für Java [GMT02] und Ada [BBS04] gibt es ähnliche Schran-

*Englischer Titel der Dissertation: "Fast algorithms for decomposable graphs"

ken. Auch Schienen- oder andere Verkehrsnetze haben in der Regel eine kleine Baumweite. Viele weitere Beispiele finden sich in [Bod93, Bod98, BK08b], auf die wir auch für die formalen Definitionen verweisen.

Den Satz von Courcelle nennen wir auch ein *Meta-Theorem*, weil er nicht nur einige vereinzelte Probleme, sondern direkt eine ganze Klasse von Problemen betrifft. Es gilt jedoch gemeinhin als bekannt, dass der Satz von Courcelle nicht unmittelbar zu einem effizienten Algorithmus führt, der in der Praxis auch tatsächlich nutzbar ist. Beispielsweise schreibt Niedermeier [Nie06] in seinem bekannten Werk über parametrisierte Algorithmen:

> *It must be emphasized, however, that the now described methodology is of purely theoretical interest because the associated running times suffer from huge constant factors and combinatorial explosions with respect to the parameter treewidth.* [...] *After establishing fixed-parameter tractability in this way, as a second step one should then head for a concrete, problem-specific algorithm with improved efficiency* [Nie06, p. 169f].

Ähnliche Aussagen von weiteren Autoren findet man mitunter in [GPW10b, Gro99]. Dies hat unter anderem die folgenden zwei Gründe: Erstens wird im Standardbeweis von Courcelles Satz ein endlicher Baumautomat konstruiert, der die Baumzerlegung genau dann akzeptiert, wenn die Formel auf dem Eingabegraph erfüllt ist. Zwar ist die Konstruktion eines endlichen Baumautomaten aus einer MSO-Formel in der Theorie sehr gut verstanden – in der Praxis jedoch bleibt dies eine herausfordernde Aufgabe wegen des Problems der „Zustandsexplosion“, siehe z.B. [KMS01, Mar06]. Zweitens gibt es unter der Voraussetzung $\mathsf{P} \neq \mathsf{NP}$ riesige untere Schranken [FG04] für die Konstanten in der Laufzeitabschätzung, so dass bessere Laufzeiten im Allgemeinen auch überhaupt nicht möglich sind. Damit ist die Automatenmethode – zumindest im Hinblick auf die Laufzeitanalyse – beweisbar die effizienteste Möglichkeit.

2 Stand der Forschung

In der Praxis konzentriert man sich daher hauptsächlich darauf, der Platzproblematik beizukommen. Dementsprechend betreffen viele der “*implementation secrets*” [KMS01] der bekannten MONA-Software [KM01] den Speicherbedarf, etwa die Darstellung der Automaten durch BDDs oder die Einführung sogenannter *guides*. Möchte man jedoch die Baumautomaten nach dem Satz von Courcelle konstruieren, scheitert dies selbst bei vielen Trivialproblemen an dem dafür benötigten Platzbedarf – und zwar unabhängig davon, ob man die Algorithmen selbst implementiert oder bestehende Software wie MONA oder `Autowrite` [Dur02, Dur05] dafür verwendet, vgl. [Kep05, Sog08, GPW10b, GPW10a, Mar06, CD12, CE12].

Als Abhilfe schlagen Courcelle und Durand [CD10a, CD10b, CD11, CD12] vor, die problematische Potenzmengenkonstruktion zu vermeiden, indem nicht-deterministische anstelle deterministischer Baumautomaten verwendet werden und indem Quantorenwechsel in der Formel verboten werden (existentielles Fragment von MSO). Dem damit verbunde-

nen Verlust an Ausdrucksstärke der Logik soll teilweise entgegengewirkt werden, indem für häufig verwendete Konstrukte (etwa für Zusammenhang oder für Mengen und Partionen) vordefinierte Prädikate zur Verfügung stehen. Desweiteren ist die von ihnen verwendete Software `Autowrite` in der Lage, die Transitionen des Automaten während der Simulation "*on-the-fly*" zu berechnen, so dass keine vollständige Repräsentation des Automaten im Speicher vorgehalten werden muss. Stattdessen werden sie nur implizit durch eine kleine Anzahl „Meta-Regeln" dargestellt und die jeweils möglichen Transitionen in jedem Schritt daraus abgeleitet. Durch eine syntaktische Erweiterung kann `Autowrite` auch Entscheidungsprobleme lösen, bei denen eine Zahl Teil der Eingabe ist. Leider ist bisher keine Implementierung für Baumzerlegungen verfügbar (verwendet wurde stattdessen das Maß der Cliquenweite), durchgeführte Experimente [CD10b, CD10a, CD12, CE12] deuten jedoch darauf hin, dass ihre Vorgehensweise durchaus Potential hat, wenn man nicht auf die volle Ausdrucksstärke von MSO angewiesen ist.

Einen völlig anderen Weg schlagen Gottlob, Pichler und Wei [GPW10b, GPW10a, Pic11] vor, deren ursprüngliche Motivation in der Antwortmengenprogrammierung (*Answer Set Programming*) liegt: Anstatt einen Baumautomaten zu konstruieren, beschreiben sie eine Übersetzung der MSO-Formel in ein Datalog Programm, in dem nur monadische Prädikate verwendet werden (*Monadic Datalog*). Ähnlich wie die ganzzahlige Optimierung ist Datalog schon seit Jahren Gegenstand intensiver Forschung, so dass bereits eine Reihe hochoptimierter, nachweisbar praxistauglicher Software zur Verfügung steht, deren Stärke man so nutzen kann. Experimente, die in verschiedenen Arbeiten zu verschiedenen Problemstellungen durchgeführt wurden [GPW10b, GPW10a, JPW09, JPRW08], zeigen, dass der Umweg über Datalog in der Tat praktikabel sein kann. Leider steht bisher noch keine Implementierung einer *automatischen* Übersetzung der MSO-Formeln in ein Datalog-Programm zur Verfügung, und so wurden die Datalog-Programme für obige Experimente von den Autoren von Hand geschrieben. Daher bleibt auch offen, ob die nach [GPW10b] automatisch generierten Programme ähnlich effizient lösbar sind.

3 Ein spieltheoretischer Ansatz

In dieser Arbeit schlagen wir einen neuen Ansatz vor, der auf einem modelltheoretischen Spiel basiert, das unter dem Namen Model-Checking-Spiel bekannt ist, vgl. [Hin73, Grä07, Grä11]. Das hier vorgestellte Verfahren funktioniert für MSO auf beliebigen Strukturen, wobei nicht nur Entscheidungsprobleme gelöst werden können, sondern wie in [CM93] auch eine große Anzahl Optimierungs- und Aufzählungsprobleme.

Das Model-Checking-Spiel $\mathscr{G} = \mathscr{G}(\varphi, G)$ wird jeweils für eine MSO-Formel φ und einen Graphen $G = (V, E)$ gespielt. Es gibt zwei Spieler, den Verifizierer und den Falsifizierer. Der Verifizierer möchte nachweisen, dass die Formel auf dem Graphen erfüllt ist, der Falsifizierer möchte das Gegenteil beweisen. Das Spielbrett ist ein gerichteter, azyklischer Graph, dessen Positionen (ψ, α) jeweils aus einer MSO-Formel ψ und einer Variablenbelegung α bestehen. Von jeder Position (ψ, α) gibt es eine Menge von Kanten zu Nachfolgepositionen (ψ', α'), jeweils mit einer Unterformel ψ' von ψ samt passender Variablenbelegung α'. Ist beispielsweise $\psi = \forall X \psi'$, dann gibt es für jedes $U \subseteq V$ eine Kante

von (ψ, α) nach $(\psi', \alpha[X/U])$, wobei $\alpha[X/U]$ bedeutet, dass in α nun die Variable X der Formel mit dem Wert U belegt wurde. Ist beispielsweise aber $\psi = \psi_1 \vee \psi_2$, dann gibt es von (ψ, α) zwei ausgehende Kanten zu (ψ_1, α) und (ψ_2, α). Von Positionen mit atomaren Formeln gibt es keine ausgehenden Kanten. Das Spiel beginnt in (φ, ε), wobei ε die leere Variablenbelegung ist. In jeder Position darf genau einer der beiden Spieler ziehen und eine Nachfolgeposition anhand einer ausgehenden Kante auswählen – bei universellen Formeln $(\wedge, \forall)$ der Falsifizierer, bei existentiellen Formeln $(\vee, \exists)$ der Verifizierer. Da der Spielgraph azyklisch und gerichtet ist, endet das Spiel nach einer endlichen Anzahl von Schritten in einer Position (ψ, α), wobei ψ eine atomare Formel ist. Wenn nun die Formel mit der Variablenbelegung α auf G erfüllt ist (kurz $G \models \psi[\alpha]$), gewinnt der Verifizierer, ansonsten der Falsifizierer. Nun kann man durch Induktion über den Formelaufbau zeigen, dass φ auf G genau dann erfüllt ist, wenn der Verifizierer eine Gewinnstrategie für das Spiel $\mathscr{G}(\varphi, G)$ hat. Für weiterführende Informationen verweisen wir auf die Literatur, z.B. [Grä07, Grä11].

Ist das Spiel $\mathscr{G}$ gegeben, kann man effizient testen, welcher der beiden Spieler eine Gewinnstrategie hat, vgl. [Grä07, Grä11]. Jedoch ist die Darstellung von $\mathscr{G}$ bis zu exponentiell größer als die Darstellung von φ und G. Es ist für uns daher nicht sinnvoll, $\mathscr{G} = (G, \varphi)$ zu konstruieren. Zu jedem Model-Checking-Spiel $\mathscr{G}$ gibt es allerdings ein äquivalentes Spiel $\mathscr{G}'$ konstanter Größe. Hier seien $\mathscr{G}$ und $\mathscr{G}'$ *äquivalent*, wenn gilt, dass der Verifizierer genau dann eine Gewinnstrategie für $\mathscr{G}'$ hat, wenn er eine für $\mathscr{G}$ hat. Wenn $G \models \varphi[\varepsilon]$ gilt, ist ein äquivalentes Spiel zu $\mathscr{G}(\varphi, G)$ beispielsweise das Spiel, welches nur die Position $(\mathit{true}, \varepsilon)$ enthält.

Unser Ziel ist es nun, dieses Spiel $\mathscr{G}'$ zu konstruieren. Ignorieren wir für einen Augenblick einmal die Laufzeit und Platzbedarf, kann man dazu prinzipiell wie folgt vorgehen: Wir verwenden dynamische Programmierung auf der Baumzerlegung, die ohne Beschränkung der Allgemeinheit eine gewisse Form hat, die wir „nett“ nennen. Zu jedem Knoten t der Baumzerlegung sei V_t die Menge der Knoten in G, die unterhalb von t in der Baumzerlegung vorkommen. Dann sei $G_t = G[V_t]$ der entsprechende induzierte Untergraph von G, der die gleiche Baumweite wie G hat. Die dynamische Programmierung beginnt in den Blättern t der Baumzerlegung, auf denen die $G[V_t]$ nur konstante Größe haben – damit ist es möglich, das Spiel $\mathscr{G}_t = \mathscr{G}(\varphi, G_t)$ in konstanter Zeit zu konstruieren. Für innere Knoten der Baumzerlegung wird das Spiel $\mathscr{G}_t = \mathscr{G}(\varphi, G_t)$ nun aus den bereits konstruierten Spielen der Kindsknoten von t konstruiert. Ist beispielsweise t ein Knoten der Baumzerlegung, in welchem die Untergraphen G_l und G_r *verschmolzen* werden, geht man grob gesagt so vor, dass für jede Position in $\mathscr{G}_t$ zwei bestimmte Positionen aus $\mathscr{G}_l$ und $\mathscr{G}_r$ auch geeignet miteinander verschmolzen werden. In der Wurzel t der Baumzerlegung gilt $G = G_t$ und somit $\mathscr{G} = \mathscr{G}_t$. Nun testen wir, welcher der beiden Spieler eine Gewinnstrategie hat und erzeugen das gesuchte Spiel $\mathscr{G}'$ mit nur einer einzigen Position – entweder $(\mathit{true}, \varepsilon)$ oder $(\mathit{false}, \varepsilon)$.

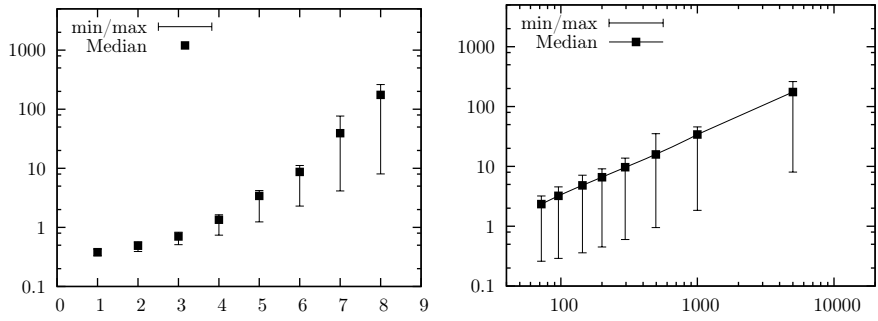
Wenn G ein Graph mit beschränkter Baumweite ist, soll dieses Verfahren aber in Linearzeit möglich und zudem auch noch praxistauglich sein. Dazu führen wir zwei Verbesserungen im Verfahren ein, die in [KLR11] bzw. [LRS11] erstmalig vorgestellt wurden. Die erste Verbesserung besteht im Zusammenfassen von äquivalenten Positionen im Spiel. Die genaue Definition ist sehr technisch, deshalb möchten wir es bei einem ein-

fachen Beispiel zur Veranschaulichung belassen: Wenn $u, v, w \in V$ drei Knoten mit $\{u, v\} \in E$ und $\{u, w\} \in E$ sind, dann kann die Formel $\mathrm{adj}(x, y)$ diese Knotenpaare nicht voneinander unterscheiden. Auch im Spiel sind aus Sicht der Spieler die Positionen $(\mathrm{adj}(x, y), \varepsilon[x/u, y/v])$ und $(\mathrm{adj}(x, y), \varepsilon[x/u, y/w])$ äquivalent, da in beiden Fällen der Verifizierer gewinnen wird. Diese Positionen können also zu einer gemeinsamen Position zusammengefasst werden, ohne den Ausgang des Spiels zu beeinflussen. Fassen wir sukzessive alle äquivalenten Positionen zusammen, erhalten wir irgendwann ein zu $\mathscr{G}$ äquivalentes Spiel $\mathscr{R}$ reduzierter Größe. In einem gewissen Sinne verhält sich das Spiel $\mathscr{R}$ damit zu $\mathscr{G}$ so wie der Bisimulationsquotient eines Transitionssystems zum ursprünglichen Transitionssystem (siehe z.B. [BK08a]). Wir können nun zeigen, dass die Größe von $\mathscr{R}$ ausschließlich von φ und der Baumweite abhängt – sind Baumweite und Formel Konstanten, hat $\mathscr{R}$ also konstante Größe. Bereits allein durch diese erste Verbesserung des Verfahrens ist es möglich, einen spieltheoretischen Beweis für den Satz von Courcelle zu führen [LRS11]. Dieses Verfahren ist allerdings noch nicht praktisch anwendbar: Die Größe von $\mathscr{R}$ entspricht den bekannten unteren Schranken für Courcelles Theorem [FG04] und ist ein exponentieller „Turm“, dessen Höhe von der Formel abhängt.

Die zweite Verbesserung besteht darin, Positionen aus dem Spiel zu entfernen, wenn klar ist, dass diese sowieso niemals besucht werden oder durch eine äquivalente Position ersetzt werden können. Ist beispielsweise die Formel $\exists v \forall u \psi$, und hat der Falsifizierer eine Gewinnstrategie, sobald im Spiel die Position $p' = (\forall u \psi, \alpha')$ erreicht wird, dann wird der optimal spielende Verifizierer ausgehend von der Position $p = (\exists v \forall u \psi, \alpha)$ niemals auf p' ziehen, es sei denn, er verliert bei *allen* Nachfolgepositionen von p. In beiden Fällen können wir nun Positionen aus dem Spiel entfernen: Entweder löschen wir p' ersatzlos, oder wir ersetzen p durch die äquivalente Position $(\mathit{false}, \alpha')$; nicht mehr erreichbare Nachfolgepositionen werden anschließend ebenfalls entfernt.

Im Verlauf der dynamischen Programmierung ist es jedoch mit dem einfachen Model-Checking-Spiel von oben nicht möglich, zu entscheiden, welche Positionen entfernt werden können, da die Spiele (φ, G) und $(\varphi, G[U])$ für einen Teilgraphen $G[U]$ von G nicht notwendigerweise äquivalent sind – man betrachte etwa die Formel $\exists x(x = x)$ und $U = \emptyset$. Daher führen wir eine Erweiterung des klassischen Model-Checking-Spiels ein, in welchem es möglich ist, in der Variablenbelegung α Platzhalter zu lassen, die in der dynamischen Programmierung erst später belegt werden sollen. In diesem Spiel $\mathscr{E}_t = \mathscr{E}(\varphi, G_t)$ gibt es nun auch die zusätzliche Möglichkeit, dass das Spiel *unentschieden* endet, nämlich genau dann, wenn in einer Position (ψ, α) die Formel eine Variable verwendet wird, die in α nicht belegt ist.

Wir können nun zeigen, dass gilt: Wenn für einen Knoten t der Baumzerlegung einer der beiden Spieler eine Gewinnstrategie für $\mathscr{E}_t$ hat, dann hat er auch eine Gewinnstrategie für $\mathscr{G}$. Wir nennen dies daher auch „frühe Determinierung“. Da diese Eigenschaft beweisbar auch für jede Position des Spiels $\mathscr{E}_t$ gilt, sind wir bereits im Knoten t der Baumzerlegung in der Lage, Positionen (ψ, α) von $\mathscr{E}_t$ zu entfernen bzw. durch (false, α) oder (true, α) zu ersetzen und so die Größe des Spiels $\mathscr{E}_t$ erheblich zu verkleinern. In der Wurzel der Baumzerlegung schließlich entfernen wir alle Positionen mit Platzhaltern in der Variablenbelegung. Dadurch werden alle verbleibenden Positionen in $\mathscr{E}_t$ determiniert, so dass wir schließlich das gesuchte Spiel $\mathscr{G}'$ mit der einzigen Position $(\mathit{true}, \varepsilon)$ bzw.



(a) MINIMUM DOMINATING SET, 5000 Knoten (b) MINIMUM DOMINATING SET, Breite 8

Abbildung 1: Laufzeiten (in Sekunden) unserer MSO-Software für Gitter variabler Breite bei fester Anzahl Knoten sowie variabler Anzahl Knoten bei fester Breite.

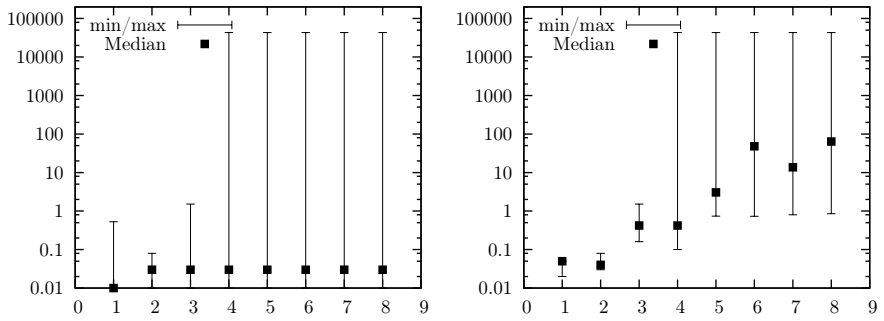
$\mathscr{G}' = (false, \varepsilon)$ erhalten. In diesem können wir den Gewinner des Spiels schnell bestimmen und die Lösung ausgeben.

Wenn wir diese beiden Techniken miteinander kombinieren, wird in jedem Knoten t der Baumzerlegung ein Spiel $\mathscr{E}'_t$ betrachtet, welches nur konstante Größe hat. Die gesamte Laufzeit des Verfahrens ist daher linear in der Größe der Baumzerlegung und auch des Eingabegraphens. Im Allgemeinen sind die Konstanten in der Laufzeitabschätzung wegen der bekannten unteren Schranken wieder riesig – für konkrete Probleme können wir jedoch beweisen, dass die Laufzeit unseres allgemeinen Verfahrens der Laufzeit spezieller Algorithmen für Graphen mit Baumweite w erstaunlich nahe kommt: Für MINIMUM VERTEX COVER ist diese beispielsweise $O(2^w poly(w)n)$ gegenüber den $O(2^w n)$ des speziellen Algorithmus [AP89]. Für 3-COLORABILITY ist sie $O(3^w poly(w)n)$ gegenüber den $O(3^w n)$ aus [AP89]. Beides ist unter einer bestimmten komplexitätstheoretischen Annahme sogar bis auf den kleinen polynomiellen Faktor $poly(w)$ optimal [LMS11]. Für MINIMUM DOMINATING SET hat unser Algorithmus eine Laufzeit von $O(5^w poly(w)n)$. Dies ist nicht optimal, allerdings nutzen sowohl der $O(3^w poly(w)n)$-Algorithmus aus [vRBR09] als auch der $O(4^w n)$-Algorithmus aus [AN02] eine spezielle Monotonie-Eigenschaft von Dominierungsproblemen aus, die für MSO-definierbare Probleme in der Regel nicht gilt.

4 Experimente

Das hier vorgestellte Verfahren wurde von uns in C++ implementiert und unter dem Namen *Sequoia* als Open-Source-Software veröffentlicht[1]. Um seine Praxistauglichkeit zu ermitteln, haben wir sie in einer Reihe von Experimenten mit IBM ILOG CPLEX [CPL] verglichen, einer bekannten kommerziellen Software für die ganzzahlige und lineare Programmierung. Verwendet wurde CPLEX Academic Research Edition 12.4.0.0. Eine kleine Auswahl der Ergebnisse zeigen die Abbildungen 1 und 2 sowie Tabelle 1.

[1] Verfügbar auf `https://github.com/sequoia-mso`



(a) alle Graphen (b) nur dichter besetzte Graphen

Abbildung 2: Laufzeiten (in Sekunden) von CPLEX für MINIMUM DOMINATING SET, alle Instanzen gegenüber den dichter besetzten Teilgraphen (Kantenwahrscheinlichkeit $p \geq 0.9$) von Gittern mit ungefähr 1000 Knoten bei variabler Breite. Bei der Mehrheit der dünnbesetzten Gittern war bereits die LP-Relaxation optimal oder nahezu optimal. Das Problem wird deutlich schwerer auf dichtbesetzten Instanzen: Obwohl CPLEX sogar beim Erreichen von potentiell nicht-optimalen Lösungen bei einem verbleibenden *integrality gap* von 5% stoppen durfte, benötigt CPLEX deutlich mehr Zeit auf diesen Instanzen als unsere MSO-Software.

			Sequoia		*CPLEX*		
Graph	*Größe*	*Bw.*	*Zeit*	*Lösung*	*Zeit*	*Lösung*	*gap*
Hannover, klein	673	8	$3''$	319	*Time-Out*	327	41 %
Hannover, groß	956	9	$9''$	376	*Time-Out*	385	42 %
Berlin	2599	11	$197''$	1259	*Time-Out*	1342	35 %

Tabelle 1: Laufzeiten für das MINIMUM CONNECTED DOMINATING SET-Problem auf drei Graphen, die aus echten Schienennetzen erzeugt wurden. *Größe:* Anzahl der Knoten; *Bw.:* Baumweite der Instanz; *Lösung:* beste gefundene Lösung innerhalb der angegebenen *Zeit*; *gap:* das verbleibende *integrality gap* von CPLEX; das *Time-Out* ist nach 43 200 Sekunden (12 Stunden).

5 Fazit

In dieser Arbeit haben wir einen neuartigen Beweis für den Satz von Courcelle beschrieben, der zu einem allgemeinen und praxistauglichen Verfahren für Graphprobleme auf Instanzen kleiner Baumweite führt. Überraschenderweise kann die von uns entwickelte Software sogar mit bestehenden kommerziellen Lösungen mithalten, wenn es Optimierungsprobleme auf Graphen mit kleiner Baumweite zu lösen gilt. Der immense Fortschritt in anderen Bereichen wie das Lösen von Entscheidungsproblemen Boolscher Formeln oder der Logikprogrammierung stimmt uns zuversichtlich, dass durch weitere Forschung und neue Ideen die Geschwindigkeit von MSO-Model-Checking-Software noch deutlich weiter gesteigert werden kann.

Literatur

[ALS91] S. Arnborg, J. Lagergren und D. Seese. Easy Problems for Tree-Decomposable Graphs. *J. Algorithms*, 12(2):308–340, 1991.

[AN02] J. Alber und R. Niedermeier. Improved tree decomposition based algorithms for domination-like problems. In *Proceedings of the 5th Symposium on Latin American Theoretical Informatics (LATIN)*, number 2286 in Lecture Notes in Computer Science, Seiten 613–627, Cancun, Mexico, 2002. Springer.

[AP89] S. Arnborg und A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.

[BBS04] B. Burgstaller, J. Blieberger und B. Scholz. On the Tree Width of Ada Programs. In *9th Ada-Europe International Conference on Reliable Software Technologies*, number 3063 in Lecture Notes in Computer Science, Seiten 78–90. Springer, 2004.

[BK08a] C. Baier und J.-P. Katoen. *Principles of Model Checking (Representation and Mind Series)*. The MIT Press, 2008.

[BK08b] H.L. Bodlaender und A. M. C. A. Koster. Combinatorial Optimization on Graphs of Bounded Treewidth. *The Computer Journal*, 51(3):255–269, 2008.

[Bod93] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernetica*, 11:1–21, 1993.

[Bod98] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209:1–45, 1998.

[CD10a] B. Courcelle und I. A. Durand. Tractable constructions of finite automata from monadic second-order formulas, 2010. Presented at *Logical Approaches to Barriers in Computing and Complexity*, Greifswald, Germany.

[CD10b] B. Courcelle und I. A. Durand. Verifying monadic second-order graph properties with tree automata. In *3rd European Lisp Symposium*, Seiten 7–21, 2010. Informal proceedings edited by C. Rhodes.

[CD11] B. Courcelle und I. A. Durand. Fly-Automata, Their Properties and Applications. In *Implementation and Application of Automata - 16th International Conference, CIAA 2011*, number 6807 in Lecture Notes in Computer Science, Seiten 264–272. Springer, 2011.

[CD12] B. Courcelle und I. A. Durand. Automata for the verification of monadic second-order graph properties. *J. Applied Logic*, 10(4):368–409, 2012.

[CE12] B. Courcelle und J. Engelfriet. *Graph Structure and Monadic Second-Order Logic: A Language Theoretic Approach*. Number 138 in Encyclopedia of Mathematics and its Applications. Cambridge University Press, Juni 2012.

[CM93] B. Courcelle und M. Mosbah. Monadic second-order evaluations on tree-decomposable graphs. *Theor. Comput. Sci.*, 109(1-2):49–82, 1993.

[Cou90] B. Courcelle. The Monadic Second-Order Theory of Graphs. I. Recognizable Sets of Finite graphs. *Information and Computation*, 85:12–75, 1990.

[CPL] IBM ILOG CPLEX Optimizer, Version 12.4.0.0. `http://www-01.ibm.com/software/integration/optimization/cplex-optimization-studio/`. Aufgerufen am 2012-09-17.

[Dur02] I. Durand. Autowrite: A Tool for Checking Properties of Term Rewriting Systems. In Sophie Tison, Hrsg., *Proceedings of the 13th International Conference on Rewriting Techniques and Applications (RTA)*, number 2378 in Lecture Notes in Computer Science, Seiten 371–375. Springer, 2002.

[Dur05] I. Durand. A Tool for Term Rewrite Systems and Tree Automata. *Electr. Notes Theor. Comput. Sci.*, 124(2):29–49, 2005.

[FG04] M. Frick und M. Grohe. The complexity of first-order and monadic second-order logic revisited. *Annals of Pure and Applied Logic*, 130(1–3):3–31, 2004.

[GMT02] J. Gustedt, O. A. Mæhle und J. A. Telle. The Treewidth of Java Programs. In D. M. Mount und C. Stein, Hrsg., *ALENEX*, number 2409 in Lecture Notes in Computer Science, Seiten 86–97. Springer, 2002.

[GPW10a] G. Gottlob, R. Pichler und F. Wei. Bounded treewidth as a key to tractability of knowledge representation and reasoning. *Artif. Intell.*, 174(1):105–132, 2010.

[GPW10b] G. Gottlob, R. Pichler und F. Wei. Monadic datalog over finite structures of bounded treewidth. *ACM Trans. Comput. Logic*, 12(1):3:1–3:48, 2010.

[Grä07] E. Grädel. Finite Model Theory and Descriptive Complexity. In *Finite Model Theory and Its Applications*, Seiten 125–230. Springer, 2007.

[Grä11] E. Grädel. Back and Forth Between Logics and Games. In K. R. Apt und E. Grädel, Hrsg., *Lectures in Game Theory for Computer Scientists*, Seiten 99–145. Cambridge University Press, 2011.

[Gro99] M. Grohe. Descriptive and Parameterized Complexity. In J. Flum und M. Rodríguez-Artalejo, Hrsg., *Proceedings of the 13th international Workshop on Computer Science Logic (CSL)*, number 1683 in Lecture Notes in Computer Science, Seiten 14–31. Springer, 1999.

[Hal76] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976.

[Hin73] J. Hintikka. *Logic, Language-Games and Information: Kantian Themes in the Philosophy of Logic*. Clarendon Press, 1973.

[JPRW08] M. Jakl, R. Pichler, S. Rümmele und S. Woltran. Fast Counting with Bounded Treewidth. In *Proceedings of the 15th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning*, number 5330 in Lecture Notes in Computer Science, Seiten 436–450. Springer, 2008.

[JPW09] M. Jakl, R. Pichler und S. Woltran. Answer-Set Programming with Bounded Treewidth. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, Seiten 816–822, 2009.

[Kep05] S. Kepser. Using MONA for Querying Linguistic Treebanks. In *Proceedings of HLT/EMNLP 2005*. The Association for Computational Linguistics, 2005.

[KLR11] J. Kneis, A. Langer und P. Rossmanith. Courcelle's Theorem – A Game-Theoretic Approach. *Discrete Optimization*, 8(4):568–594, 2011.

[KM01] N. Klarlund und A. Møller. *MONA Version 1.4 User Manual*. BRICS, Dept. of Comp. Sc., University of Aarhus, January 2001. Verfügbar auf `http://www.brics.dk/mona/`.

[KMS01] N. Klarlund, A. Møller und M. I. Schwartzbach. MONA Implementation Secrets. In *Proc. of CIAA'00*, Seiten 182–194. Springer-Verlag, 2001.

[Lan13] A. Langer. *Fast algorithms for decomposable graphs*. Dissertation, RWTH Aachen University, 2013.

[LMS11] D. Lokshtanov, D. Marx und S. Saurabh. Slightly Superexponential Parameterized Problems. In D. Randall, Hrsg., *Proceedings of the 22nd ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Seiten 760–776. SIAM, 2011.

[LRS11] A. Langer, P. Rossmanith und S. Sikdar. Linear-Time Algorithms for Graphs of Bounded Rankwidth: A Fresh Look Using Game Theory (Extended Abstract). In *TAMC'11*, number 6648 in LNCS, Seiten 505–516. Springer, 2011.

[Mar06] H. Maryns. On the Implementation of Tree Automata: Limitations of the Naive Approach. In *Proc. 5th Int. Treebanks and Linguistic Theories Conference (TLT 2006)*, Seiten 235–246, 2006.

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

[Pic11] R. Pichler. Exploiting Bounded Treewidth with Datalog (A Survey). In *Datalog Reloaded - First International Workshop*, number 6702 in Lecture Notes in Computer Science, Seiten 88–105. Springer, 2011.

[RS86] N. Robertson und P. D. Seymour. Graph minors II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7:309–322, 1986.

[Sog08] D. Soguet. *Génération automatique d'algorithmes linéaires*. Doctoral dissertation, University Paris-Sud, 2008.

[Tho98] M. Thorup. All Structured Programs have Small Tree-Width and Good Register Allocation. *Inf. Comput.*, 142(2):159–181, 1998.

[vRBR09] J. M. M. van Rooij, H. L. Bodlaender und P. Rossmanith. Dynamic Programming on Tree Decompositions Using Generalised Fast Subset Convolution. In *ESA*, number 5757 in Lecture Notes in Computer Science, Seiten 566–577. Springer, 2009.

Alexander Langer hat von 2001–2008 an der RWTH Aachen Informatik studiert und war von 2008–2012 wissenschaftlicher Mitarbeiter am Lehr- und Forschungsgebiet Theoretische Informatik der RWTH Aachen. Nach seiner Diplomarbeit, in der es bereits um MSO-definierbare Probleme für Graphen mit kleiner Baumweite ging, hat er im Rahmen seiner Promotion, die er 2013 abschloss, an der Entwicklung und Implementierung eines praxistauglichen Verfahrens für solche Probleme geforscht.