Über das Lösen nichtgrundierter Answer-Set-Programme unter Berücksichtigung der Baumweite[1]

Bernhard Bliem[2]

Abstract: Viele wichtige Probleme, die in der Praxis auftreten, sind NP-schwer. Ein etablierter Ansatz, um eine Vielzahl solcher Probleme zu lösen, ist die deklarative Sprache Answer Set Programming (ASP). Allerdings ist dies für einige NP-schwere Probleme trotz der beeindruckenden Effizienz von ASP-Solvern nicht praktikabel. Ein Hoffnungsschimmer ist die Beobachtung, dass in der Praxis auftretende Probleminstanzen oft kleine Baumweite aufweisen. Es konnte nämlich beobachtet werden, dass moderne ASP-Solver effizienter sind, wenn ihre Eingabe kleine Baumweite hat. Leider ist die Eingabe dieser Solver üblicherweise nicht die Probleminstanz selbst, sondern eine durch sogenanntes Grundieren gewonnene Zwischeninstanz, und dieser Prozess kann die Baumweite drastisch vergrößern. Der Einfluss des Grundierens auf die Baumweite wurde bisher nicht hinreichend verstanden.

In der Dissertation klären wir die Frage, unter welchen Umständen grundiert werden kann ohne die Baumweite deutlich zu erhöhen. Dazu definieren wir Klassen von ASP-Programmen und beweisen, dass das Grundieren solcher Programme zusammen mit der Eingabe nicht mit einem übermäßigen Anstieg der Baumweite einhergeht. Können wir ein Problem in einer solchen Klasse ausdrücken, profitieren wir somit automatisch von der augenscheinlichen „Empfindlichkeit“ von ASP-Solvern gegenüber Baumweite. Außerdem präsentieren wir Fortschritte in einer algorithmischen Methodik für das explizite Ausnutzen beschränkte Baumweite. Hier zielen wir insbesondere auf Probleme ab, welche Teilmengenminimierung erfordern, wie es für zahlreiche Probleme auf der zweiten Stufe der Polynomiellen Hierarchie der Fall ist. Schließlich klären wir einige seit langem unbeantwortete Fragen über die Komplexität von Allianzproblemen in Graphen.

1 Einführung

Answer Set Programming (ASP) hat sich zu einem äußerst beliebten Paradigma zum Lösen schwieriger Berechnungsprobleme entwickelt. Es bietet eine leicht verwendbare Sprache, welche prägnante Problemspezifikationen ermöglicht, und kann mit hocheffizienten Systemen aufwarten. Um mit ASP ein Problem zu lösen, spezifiziert man dieses, üblicherweise unabhängig von den Probleminstanzen, als eine Menge von *Regeln*, welche von den Lösungen zu erfüllende Bedingungen formalisieren. Um eine solche Menge von Regeln, genannt *ASP-Programm*, zu lösen, rufen ASP-Systeme üblicherweise zuerst einen *Grounder* auf, der ein äquivalentes *grundiertes*, d. h. variablenfreies, Programm ausgibt. Anschließend wird dieses grundierte Programm an einen *Solver* weitergereicht, der die *Answer Sets*, d. h. die Lösungen, des Programms berechnet. Wir verzichten auf eine Darstellung der Syntax und Semantik und verweisen hierzu auf die Einführung [BET11]. Wir illustrieren die Verwendung von ASP lediglich anhand des folgenden Beispiels.

Beispiel 1. Das Dreifärbbarkeitsproblem lässt sich in ASP wie folgt kodieren:

[1] Originaltitel: *Treewidth in Non-Ground Answer Set Solving and Alliance Problems in Graphs*

[2] TU Wien, bliem@dbai.tuwien.ac.at

```
rot(X) | gruen(X) | blau(X) :- knoten(X).
:- kante(X,Y), rot(X), rot(Y).
:- kante(X,Y), gruen(X), gruen(Y).
:- kante(X,Y), blau(X), blau(Y).
```

Großbuchstaben, wie X und Y in diesem Beispiel, sind Variablen und können durch Konstanten, die in der Eingabe vorkommen, instantiiert werden. In der ersten Zeile raten wir eine Farbe für jeden Knoten. Die *Constraints* in den verbleibenden Zeilen stellen sicher, dass benachbarte Knoten unterschiedliche Farben haben. Wir können dieses Programm zusammen mit einem Eingabegraphen, welcher mithilfe der Prädikate `knoten` und `kante` spezifiziert wird, einem ASP-System geben, welches dann alle gültigen Dreifärbungen des Graphen berechnet.

Wenn wir den gerichteten Graphen, der aus lediglich zwei Knoten a, b und einer Kante (a, b) besteht, auf diese Weise spezifizieren und zusammen mit dem obigen Programm einem Grounder vorlegen, könnte dieser folgendes äquivalente grundierte Programm produzieren:

```
knoten(a). knoten(b). kante(a,b).
rot(a) | gruen(a) | blau(a) :- knoten(a).
rot(b) | gruen(b) | blau(b) :- knoten(b).
:- kante(a,b), rot(a), rot(b).
:- kante(a,b), gruen(a), gruen(b).
:- kante(a,b), blau(a), blau(b).
```

Wie in diesem Beispiel führen Grounder in der Praxis diverse Optimierungen durch anstatt auf naive Weise alle Variablen durch alle möglichen Konstanten zu ersetzen. So haben wir die Regel `:- kante(b,b), blau(b), blau(b)` nicht angeführt, weil der Eingabegraph keine Kante (b, b) enthält und `kante` ein *extensionales Prädikat* ist. Das bedeutet, dass dieses Prädikat vom Programm nicht hergeleitet wird sondern nur in der Eingabe vorkommt.

Geben wir dieses grundierte Programm an einen ASP-Solver, so gibt dieser als Answer Sets genau die Dreifärbungen des Graphen aus. (Da die Semantik von ASP eine gewisse Minimalität vorsieht, wird vermieden, dass ein Knoten mehrere Farben bekommt.) △

Trotz der mittlerweile beachtlichen Effizienzfortschritte von ASP-Systemen haben diese weiterhin Schwierigkeiten mit einigen herausfordernden Problemen. Dies ist nicht immer nur eine Frage der Komplexität im Sinne der klassischen Komplexitätstheorie. Interessanterweise kommt es mitunter vor, dass ASP-Systeme auf einem Problem sehr performant sind, während sie für ein anderes Problem von gleicher Komplexität deutlich länger brauchen. Oft ist die klassische Komplexitätstheorie daher nur von beschränkter Nützlichkeit, um die Leistung von ASP-Systemen in der Praxis zu erklären. In solchen Fällen kann es erkenntnisbringend sein, die *parametrisierte* Komplexität der Probleme zu betrachten. Mithilfe dieser Theorie lässt sich die Komplexität eines Problems nicht nur hinsichtlich der Eingabegröße untersuchen, sondern auch in Bezug auf andere Parameter.

In dieser Arbeit interessieren wir uns besonders für die Auswirkung des strukturellen Parameters *Baumweite* auf die Leistung von ASP-Solvern. Die Grundidee: Je kleiner die Baumweite eines Graphen ist, desto mehr ähnelt dieser einem Baum. Es ist altbekannt,

dass viele NP-schwere Graphenprobleme effizient lösbar werden, wenn wir die Eingabe auf Bäume beschränken, und es hat sich herausgestellt, dass dies bei vielen wichtigen Problemen sogar für die allgemeinere Klasse von Graphen beschränkter Baumweite gilt [ALS91].

Tatsächlich sind viele NP-schwere Probleme *FPT* („fixed-parameter tractable“) bzgl. des Parameters Baumweite, d. h. sie können in der Zeit $\mathcal{O}(f(k) \cdot n^c)$ gelöst werden, wobei f eine beliebige berechenbare, nur von der Baumweite k abhängige, Funktion ist, n die Eingabegröße bezeichnet und c eine beliebige Konstante ist. Solche Algorithmen sind üblicherweise für kleine Werte der Baumweite k sehr effizient. Glücklicherweise konnte beobachtet werden, dass in der Praxis auftretende Instanzen üblicherweise kleine Baumweite aufweisen [Bo93]. Baumweite ist nicht nur für Graphenprobleme relevant sondern ebenso auf Instanzen verschiedenartigster Probleme anwendbar, indem man eine passende Repräsentation der Instanz als Graph wählt.

Es gab bereits einige Untersuchungen zu Baumweite in Bezug auf grundiertes ASP. Ein wichtiges Ergebnis ist der Algorithmus von [JPW09], der für grundierte ASP-Programme von beschränkter Baumweite in linearer Zeit entscheiden kann, ob das Programm eine Lösung hat. Dieser Algorithmus verwendet eine Technik namens *Dynamische Programmierung auf Baumzerlegungen*, welche sehr gängig für Algorithmen ist, die kleine Baumweite ausnutzen. Der Algorithmus von [JPW09] wurde auch implementiert und als Solver für grundiertes ASP vorgestellt [Mo10]. Für einige Probleme war dieser auf Dynamischer Programmierung basierende Solver in der Lage, die Leistung moderner ASP-Solver zu übertreffen, solange die Instanzen sehr groß waren und eine sehr kleine Baumweite hatten.

2 Problemstellung

Obwohl die ermutigenden Ergebnisse von [Mo10] bestätigten, dass kleine Baumweite erfolgreich für ASP-Solving unter „Laborbedingungen“ verwendet werden kann, waren die nötigen Einschränkungen hinsichtlich Problemen und Instanzen, um diesen Ansatz effizient anwenden zu können, zu schwerwiegend für die meisten praktischen Anwendungen. Die größten Hindernisse, diesen Ansatz für eine breite Vielfalt an Problemen nutzbar zu machen, waren die Tatsachen, dass einerseits naive Dynamische Programmierung unter einem enormen Overhead leidet (besonders bezüglich Speicherbedarf), und dass andererseits moderne ASP-Solver dermaßen effizient sind, dass sich die theoretische Überlegenheit des Dynamische-Programmierung-Algorithmus nur bei Instanzen gewaltiger Größe bezahlt macht.

Experimente in [Bl17] wiesen darauf hin, dass moderne ASP-Solver „empfindlich“ für die Baumweite ihrer Eingabe sind insofern als kleinere Baumweite stark mit höherer Performanz korreliert. Diese Beobachtungen lassen interessante Forschungsaufgaben erahnen. Insbesondere zwei Ansätze erscheinen vielversprechend, um kleine Baumweite erfolgreich für ASP-Solving in der Praxis nutzbar zu machen:

1. Die erste Forschungsaufgabe besteht darin, die auf Dynamischer Programmierung basierende Methodik zu verbessern, um deren Overhead und redundante Berechnungen zu vermindern.
   Verglichen mit anderen Problemen sind diese Punkte für das Lösen von grundiertem ASP besonders schwerwiegend, weil die entsprechenden Berechnungsprobleme (unter gängigen komplexitätstheoretischen Annahmen) noch schwieriger sind als NP. (Zu entscheiden, ob ein grundiertes ASP-Programm mit Disjunktionen ein Answer Set besitzt, befindet sich nämlich auf der zweiten Stufe der Polynomiellen Hierarchie.) Diese hohe Komplexität von grundiertem ASP widerspiegelt sich im Dynamische-Programmierung-Algorithmus [JPW09], welcher zuerst eine Brute-Force-Methode anwendet, um alle Modelle aller Teile des zerlegten Programms zu finden, und anschließend Brute Force erneut verwendet, um *für jedes solche partielle Modell* alle möglichen Gegenbeispiele zu finden, die zum Verwerfen des Kandidaten führen.
   Dieses Muster tritt zudem häufig in Dynamische-Programmierung-Algorithmen für andere Probleme auf, wo nach Lösungen gesucht wird, welche eine gewisse Teilmengenminimalität erfüllen (d. h. keine echte Teilmenge darf die Lösungsbedingungen erfüllen). Neben grundiertem ASP ist dies beispielsweise der Fall für das Problem, teilmengenminimale Modelle einer aussagenlogischen Formel zu finden. Im Allgemeinen treten Probleme mit Teilmengenminimierung recht häufig in z. B. der KI auf. Algorithmen für solche Probleme speichern üblicherweise eine große Anzahl redundanter Objekte, da die Teilmengen, die einen Lösungskandidaten entkräften, ihrerseits Lösungskandidaten sind. Weiters enthalten die Spezifikationen solcher Algorithmen selbst Redundanzen, da die potentiellen Gegenbeispiele normalerweise auf nahezu gleiche Weise behandelt werden wie die Lösungskandidaten.

2. Die zweite Forschungsaufgabe besteht darin, ASP zu lösen und dabei nicht Dynamische Programmierung durchzuführen sondern stattdessen kleine Baumweite *implizit* auszunutzen, indem man sich auf die (von Experimenten in [Bl17] gestützte) Annahme verlässt, dass moderne ASP-Solver effizienter sind, wenn man ihnen grundierte Programme von kleiner Baumweite vorlegt.
   Da Probleme üblicherweise in nichtgrundiertem ASP kodiert werden, ist hier das Forschungsziel, zu untersuchen, welche Kodierungstechniken in nichtgrundiertem ASP die Baumweite des grundierten Programms erheblich vergrößern, verglichen mit der Baumweite der Eingabe.

Neben der Nutzung von Baumweite für das Lösen von ASP sind wir darüber hinaus an einigen Varianten eines Graphenproblems namens SECURE SET [BDH07] interessiert. Es gehört zur Klasse der sogenannten *Allianzprobleme*, welche nach einer Gruppe von Knoten fragen, die einander auf eine bestimmte Weise aushelfen können. Zu praktischen Anwendungen von Allianzproblemen zählen das Finden von Gruppen von Webseiten, die Gemeinschaften bilden [Fl02] oder das Aufteilen von Ressourcen in einem Computernetzwerk sodass gleichzeitige Anfragen erfüllt werden können [HHH03]. Wir nennen eine Menge S von Knoten eines Graphen *gesichert*, wenn jede Teilmenge von S mindestens so viele Nachbarn in S hat wie Nachbarn außerhalb von S. (Formal: Die Ungleichung $|N[X] \cap S| \geqslant |N[X] \setminus S|$ muss für jede Teilmenge X von S gelten, wobei $N[X]$ die *geschlossene Nachbarschaft* von X ist, d. h. die Knoten in X und deren Nachbarn.) Das SECURE

SET Problem fragt, ob ein gegebener Graph eine gesicherte Knotenmenge von höchstens einer gegebenen Größe enthält.

Der Grund, warum wir uns mit SECURE SET beschäftigen, ist, dass dieses Problem besonders für die ASP-Forschung sehr interessante Eigenschaften aufweist: Versuche, dieses Problem in ASP auszudrücken, führten zu äußerst komplizierten Spezifikationen, welche darauf hinweisen, dass SECURE SET womöglich die volle Ausdrucksstärke von ASP benötigt [Ab15]. Es ist jedoch leider unklar, ob dies tatsächlich der Fall ist, da die Komplexität des Problems ungeklärt geblieben ist, obwohl dieses bereits 2007 vorgestellt wurde [BDH07].

Eine der Varianten von SECURE SET, welche wir in der vorliegenden Arbeit behandeln, ist das DEFENSIVE ALLIANCE Problem. Hier suchen wir nach Knotenmengen S, wo für jedes *Element* $v \in S$ die Ungleichung $|N[v] \cap S| \geqslant |N[v] \setminus S|$ erfüllt ist. Dieses Problem hat in der Fachliteratur beachtliche Aufmerksamkeit genossen [FRV14]. Es ist als NP-vollständig bekannt, doch seine Komplexität parametrisiert durch Baumweite ist bislang offen geblieben.

3 Forschungsergebnisse

Unsere Ergebnisse können in drei Gruppen eingeteilt werden: Erstens präsentieren wir Fortschritte bei der Dynamischen Programmierung; zweitens definieren wir Klassen von nichtgrundiertem ASP, von denen wir zeigen, dass das Grundieren hier beschränkte Baumweite der Eingabe erhalten kann; drittens stellen wir Komplexitätsresultate und Algorithmen für Allianzprobleme in Graphen vor.

3.1 Fortschritte bei der Dynamischen Programmierung

Wir stellen eine fortgeschrittene Variante der Dynamischen Programmierung vor, die auf Teilmengenminimierung beinhaltende Probleme abzielt. Genauer gesagt formalisieren wir, wie für jedes Problem P, dessen Lösungen exakt die teilmengenminimalen Lösungen eines Grundproblems G sind, ein Dynamische-Programmierung-Algorithmus für G automatisch in einen Dynamische-Programmierung-Algorithmus für P umgewandelt werden kann. Wir beweisen, dass die Laufzeit des resultierenden Algorithmus linear auf Instanzen beschränkter Baumweite ist, sofern dies für den Grundalgorithmus der Fall ist. Weiters zeigen wir, dass der resultierende Algorithmus korrekt ist, wenn der Grundalgorithmus korrekt ist und, sinngemäß, ausschließlich Teillösungen berechnet, die keine Entscheidungen, welche weiter unten in der Baumzerlegung getroffen worden sind, „rückgängig macht“. Der resultierende Algorithmus hat zwei Vorteile verglichen mit einem naiven Dynamische-Programmierung-Algorithmus, der P direkt löst: Erstens ist er üblicherweise einfacher zu spezifizieren, weil wir lediglich einen Algorithmus für das Grundproblem entwerfen und uns nicht um die Teilmengenminimierung kümmern müssen. Zweitens ist er unter Umständen effizienter, da er weniger redundante Objekte speichert.

In der Tat hat sich empirisch gezeigt, dass diese Methodik zu einer deutlichen Effizienzsteigerung für verschiedene Probleme führt [Bl16]. Eine verbesserte Version des klassischen Dynamische-Programmierung-Algorithmus für grundiertes ASP ist mithilfe dieser Ideen implementiert worden [Fi17] und hat sich als erheblich schneller erwiesen als der Algorithmus aus [JPW09]. Unser Ergebnis formalisiert das gemeinsame Schema, das diesen Algorithmen zugrunde liegt. Auf diese Weise stellen wir einen formalen Rahmen bereit, der es ermöglicht, die erwähnten Optimierungen einfach auf andere Probleme zu übertragen. Dadurch machen wir die eindrucksvollen Effizienzsteigerungen, von denen in [Bl16, Fi17] berichtet worden ist, für Personen zugänglich, die an verwandten Problemen arbeiten. Das ist in erster Linie für Probleme auf der zweiten Stufe der Polynomiellen Hierarchie nützlich, da Teilmengenminimierung ein wiederkehrendes Thema vieler solcher Probleme ist.

3.2 Baumweitenerhaltende Klassen von nichtgrundiertem ASP

Wir definieren Klassen von nichtgrundierten ASP-Programmen, welche so grundiert werden können, dass die beschränkte Baumweite der Eingabe erhalten bleibt. Durch Einschränken der Syntax von nichtgrundiertem ASP definieren wir zwei Programmklassen, nämlich *bewachte* („guarded“) und *verknüpft bewachte* („connection-guarded“) Programme [Bl17]. Bewachte Programme garantieren, dass die Baumweite nach dem Grundieren immer dann klein ist, wenn die Baumweite der Eingabe klein ist. Wir beweisen diese Eigenschaft formal und zeigen, dass bewachte Programme trotz ihrer Einschränkungen weiterhin Probleme ausdrücken können, die vollständig für die zweite Stufe der Polynomiellen Hierarchie sind.

Verknüpft bewachte Programme sind sogar noch ausdrucksstärker als bewachte Programme. Wir zeigen, dass für verknüpft bewachte Programme die Baumweite nach dem Grundieren immer dann klein ist, wenn die Baumweite *und der maximale Knotengrad* der Eingabe (repräsentiert als Graph) klein ist.

Mit diesen Ergebnissen nähern wir uns dem Ziel, von der Empfindlichkeit, die moderne ASP-Solver augenscheinlich gegenüber Baumweite aufweisen, implizit zu profitieren, da sie uns Einblick in die Veränderung der Baumweite der Eingabe durch das Grundieren gewähren. So können wir, indem wir ein Programm in bewachtem ASP schreiben, sicher sein, dass moderne Grounder die Beschränkung der Baumweite nicht zerstören. Im Fall von verknüpft bewachtem ASP gilt das gleiche für die Kombination von Baumweite und maximalem Knotengrad.

Beispiel 2. Das ASP-Programm in Listing 1 kann verwendet werden, um zu entscheiden, ob eine gegebene Knotenmenge S in einem gegebenen Graphen gesichert ist. Es rät eine Teilmenge X von S und verwendet sogenannte *schwache Constraints*, sodass die Kosten jedes Answer Sets exakt $|N[X] \cap S| - |N[X] \setminus S|$ betragen. (Wenn ein Programm schwache Constraints enthält, werden nur Answer Sets ausgegeben, die die Summe der verletzten schwachen Constraints minimieren.) Falls es eine Teilmenge von S gibt, die weniger Nachbarn in S als Nachbarn außerhalb von S hat, dann gibt es ein Answer Set mit negativen Kosten. Wir können auf diese Weise entscheiden, ob S gesichert ist, indem wir überprüfen, ob dieser minimale Wert negativ ist.

List. 1: Ein bewachtes ASP Programm, das prüft, ob eine gegebene Menge S (deklariert mittels des Predikats `s`) in einem gegebenen Graph (deklariert mittels der Predikate `knoten` und `kante`) gesichert ist.

```
% Rate eine Teilmenge X von S.
x(S) | nx(S) :- s(S).
% Nachbarn von X sind "gut" wenn sie in S sind, ansonsten "boese".
nachbar(V) :- x(X), kante(X,V).
nachbar(X) :- x(X), knoten(X).
gut(V)     :- nachbar(V), s(V).
boese(V)   :- nachbar(V), knoten(V), not s(V).
% Hat X mehr boese Nachbarn als gute, so ist S nicht gesichert.
% Die folgenden schwachen Constraints stellen dies durch Summieren fest.
:~ knoten(V), gut(V).    [1,V]   % +1 fuer jeden guten Nachbarn.
:~ knoten(V), boese(V). [-1,V]   % -1 fuer jeden boesen Nachbarn.
```

Das Programm in Listing 1 ist bewacht. Man beachte, dass es alternativ auch ohne schwache Constraints möglich ist, zu überprüfen, ob eine Knotenmenge gesichert ist. Beispielsweise können wir die schwachen Constraints durch den „starken“ Constraint `:- #sum{ 1,G : gut(G); -1,B : boese(B)} >= 0` ersetzen. (Dieser Constraint enthält ein sogenanntes *Aggregat* – ein fortgeschrittenes ASP-Konstrukt, mit dem summiert werden kann.) Dieser neue Constraint ist jedoch nicht bewacht. Das bedeutet, dass das ursprüngliche Programm in Listing 1 im Allgemeinen zu Grundierungen von wesentlich kleinerer Baumweite führt und daher höhere Effizienz verspricht. △

In der Dissertation präsentieren wir ebenfalls eine Komplexitätsanalyse von Berechnungsproblemen, welche diesen Programmklassen entsprechen, und betrachten als Parameter die Baumweite der Eingabe, den maximalen Knotengrad der Eingabe, und eine Kombination dieser beiden Parameter. Die Ergebnisse dieser Analyse zeigen, dass ASP-Solving für jedes fixe bewachte ASP-Programm FPT ist, wenn wir die Baumweite der Eingabe als Parameter betrachten; darüber hinaus ist ASP-Solving für jedes fixe verknüpft bewachte Programm FPT, wenn der Parameter die Kombination aus Baumweite und maximalem Knotengrad ist. Diese Resultate sind nicht offensichtlich, da unsere ASP-Klassen schwache Constraints sowie Aggregate unterstützen, welche jeweils nicht von den FPT Algorithmen [JPW09, Fi17] für grundiertes ASP unterstützt werden. Desweiteren beweisen wir Schwereresultate, welche zeigen, dass für verknüpft bewachtes ASP *sowohl* Baumweite *als auch* maximaler Grad beschränkt sein müssen, um FPT zu erreichen. Zu diesem Zweck präsentieren wir eine verknüpft bewachte ASP-Kodierung eines Problems, das sogar für fixe Baumweite NP-schwer ist, und wir präsentieren eine bewachte Kodierung eines Problems, das sogar für fixen Knotengrad Σ_2^P-schwer ist.

Als Nebenprodukt dieser Untersuchungen erhalten wir *Metatheoreme* zum Beweisen von FPT-Resultaten. Mit anderen Worten: Unsere Ergebnisse zu bewachtem ASP ermöglichen es uns, zu beweisen, dass ein durch Baumweite parametrisiertes Problem FPT ist, indem wir dieses Problem einfach in bewachtem ASP ausdrücken. Wir vergleichen dieses Metatheorem mit dem verbreiteten Ansatz, FPT zu beweisen, indem man das Problem in

monadischer Prädikatenlogik zweiter Stufe ausdrückt und den wohlbekannten Satz von Courcelle anwendet. Auf ähnliche Weise können wir zeigen, dass ein durch Baumweite gemeinsam mit maximalem Knotengrad parametrisiertes Problem FPT ist, indem wir dieses Problem in verknüpft bewachtem ASP ausdrücken. Dieses Ergebnis ist ansprechend, da uns keine Metatheoreme bekannt sind, die es ermöglichen, FPT-Resultate für die Kombination von Baumweite und Grad als Parameter zu erhalten.

3.3 Allianzprobleme in Graphen

Wir führen eine Komplexitätsanalyse von Allianzproblemen in Graphen durch, sowohl im klassischen komplexitätstheoretischen Rahmen als auch parametrisiert durch Baumweite. Zunächst klären wir die Komplexität des SECURE SET Problems, indem wir zeigen, dass das Problem, sowie unterschiedliche Varianten, Σ_2^P-vollständig (und damit auf der zweiten Stufe der Polynomiellen Hierarchie) ist.

Dann widmen wir uns der Komplexität von SECURE SET und DEFENSIVE ALLIANCE wenn beide Probleme durch Baumweite parametrisiert sind. Wir verdeutlichen den Nutzen unserer ASP-Klassen als FPT-Klassifizierungswerkzeuge, indem wir einfache Kodierungen von Allianzproblemen präsentieren. Indem wir das NP-vollständige DEFENSIVE ALLIANCE Problem in verknüpft bewachtem ASP ausdrücken, erhalten wir auf einfache Weise das bereits bekannte Resultat, dass dieses Problem parametrisiert durch die Kombination von Baumweite und maximalem Knotengrad FPT ist. Von größerer Wichtigkeit ist jedoch, dass wir ein neues Resultat für das co-NP-vollständige Problem, zu entscheiden ob eine gegebene Knotenmenge in einem gegebenen Graph gesichert ist, erhalten: Wir zeigen, dass dieses Problem FPT für den Parameter Baumweite ist, indem wir das Problem in bewachtem ASP ausdrücken.

Wir liefern auch einige negative Resultate. So zeigen wir etwa, dass (unter weitverbreiteten komplexitätstheoretischen Annahmen) weder DEFENSIVE ALLIANCE noch SECURE SET FPT sind, wenn der Parameter die Baumweite ist. Diese Fragen sind seit der Vorstellung der Probleme in den Jahren 2002 bzw. 2007 unbeantwortet geblieben und wurden explizit als offene Probleme in [KO17] (bezüglich DEFENSIVE ALLIANCE) und [HD09] (bezüglich SECURE SET) genannt.

Trotz der parametrisierten Schwere von SECURE SET können wir zumindest ein leicht positives Ergebnis vermelden: Wir zeigen, dass das SECURE SET Problem immerhin in polynomieller Zeit gelöst werden kann, wenn die Instanzen beschränkte Baumweite haben, obwohl der Grad des Polynoms von der Baumweite abhängt.

4 Publikationen

Ein Großteil der Resultate wurde auf Konferenzen vorgestellt oder in Fachzeitschriften publiziert:

- Die Fortschritte in der Dynamischen Programmierung für Probleme mit Teilmengenminimierung wurden am AAAI-Workshop *Beyond NP 2016* vorgestellt und eine erweiterte Version in der Zeitschrift *Fundamenta Informaticae* veröffentlicht [Bl16].
- Die Klasse von verknüpft bewachten ASP-Programmen, wo Grundierungen unter Beibehaltung von beschränkter Baumweite möglich sind solange der maximale Knotengrad ebenfalls beschränkt ist, wurde auf der *IJCAI 2017* präsentiert [Bl17]. Der dort vorgestellte Artikel enthielt weder die gründliche Komplexitätsanalyse, die in der Dissertation durchgeführt wurde, noch die Arbeit über die Klasse der bewachten Programme, welche attraktiv sein kann, da hier der Knotengrad nicht beschränkt sein muss. Diese Zusätze sind derzeit für eine Konferenz unter Begutachtung. Ein Zeitschriftenartikel ist in Planung.
- Das Σ_2^P-Vollständigkeitsresultat für das SECURE SET Problem wurde auf der Konferenz *WG 2015* vorgestellt [BW16]. Eine erweiterte Version dieses Artikels, welche zusätzlich die Ergebnisse zur parametrisierten Komplexität des Problems enthält, wurde in der Zeitschrift *Algorithmica* veröffentlicht [BW17]. Die vorliegende Dissertation enthält zusätzlich die parametrisierte Komplexitätsanalyse des DEFENSIVE ALLIANCE Problems, wozu ein Zeitschriftenartikel derzeit unter Begutachtung ist.

Die Forschung am Dissertationsthema hat darüber hinaus zu zahlreichen Nebenprodukten geführt, die unter anderem auf den Konferenzen *IJCAI 2016* (zwei Artikel), *ECAI 2016*, *FoIKS 2016*, *COMMA 2016* und *JELIA 2014* vorgestellt wurden. Außerdem sind zwei Artikel im *Journal of Logic and Computation (JLC)* erschienen.

Literaturverzeichnis

[Ab15] Abseher, Michael; Bliem, Bernhard; Charwat, Günther; Dusberger, Frederico; Woltran, Stefan: Computing Secure Sets in Graphs Using Answer Set Programming. J. Logic Comput., 2015. Zur Publikation angenommen.

[ALS91] Arnborg, Stefan; Lagergren, Jens; Seese, Detlef: Easy Problems for Tree-Decomposable Graphs. Journal of Algorithms, 12(2):308–340, 1991.

[BDH07] Brigham, Robert C.; Dutton, Ronald D.; Hedetniemi, Stephen T.: Security in Graphs. Discrete Appl. Math., 155(13):1708–1714, 2007.

[BET11] Brewka, Gerhard; Eiter, Thomas; Truszczyński, Mirosław: Answer Set Programming at a Glance. Communications of the ACM, 54(12):92–103, 2011.

[Bl16] Bliem, Bernhard; Charwat, Günther; Hecher, Markus; Woltran, Stefan: D-FLATˆ2: Subset Minimization in Dynamic Programming on Tree Decompositions Made Easy. Fund. Inform., 147(1):27–61, 2016.

[Bl17] Bliem, Bernhard; Moldovan, Marius; Morak, Michael; Woltran, Stefan: The Impact of Treewidth on ASP Grounding and Solving. In (Sierra, Carles, Hrsg.): Proceedings of IJCAI 2017. AAAI Press, S. 852–858, 2017.

[Bo93] Bodlaender, Hans L.: A Tourist Guide through Treewidth. Acta Cybernet., 11(1-2):1–21, 1993.

[BW16] Bliem, Bernhard; Woltran, Stefan: Complexity of Secure Sets. In (Mayr, Ernst W., Hrsg.): Revised Papers of WG 2015. Jgg. 9224 in LNCS. Springer, S. 64–77, 2016.

[BW17] Bliem, Bernhard; Woltran, Stefan: Complexity of Secure Sets. Algorithmica, 2017. Im Druck.

[Fi17] Fichte, Johannes Klaus; Hecher, Markus; Morak, Michael; Woltran, Stefan: Answer Set Solving with Bounded Treewidth Revisited. In (Balduccini, Marcello; Janhunen, Tomi, Hrsg.): Proceedings of LPNMR 2017. Jgg. 10377 in LNCS. Springer, S. 132–145, 2017.

[Fl02] Flake, Gary William; Lawrence, Steve; Giles, C. Lee; Coetzee, Frans: Self-Organization and Identification of Web Communities. IEEE Computer, 35(3):66–71, 2002.

[FRV14] Fernau, Henning; Rodríguez-Velázquez, Juan A.: A Survey on Alliances and Related Parameters in Graphs. Electron. J. Graph Theory Appl. (EJGTA), 2(1):70–86, 2014.

[HD09] Ho, Yiu Yu; Dutton, Ronald D.: Rooted Secure Sets of Trees. AKCE Int. J. Graphs Comb., 6(3):373–392, 2009.

[HHH03] Haynes, Teresa W.; Hedetniemi, Stephen T.; Henning, Michael A.: Global Defensive Alliances in Graphs. Electron. J. Combin., 10, 2003.

[JPW09] Jakl, Michael; Pichler, Reinhard; Woltran, Stefan: Answer-Set Programming with Bounded Treewidth. In (Boutilier, Craig, Hrsg.): Proceedings of IJCAI 2009. AAAI Press, S. 816–822, 2009.

[KO17] Kiyomi, Masashi; Otachi, Yota: Alliances in Graphs of Bounded Clique-Width. Discrete Appl. Math., 223:91–97, 2017.

[Mo10] Morak, Michael; Pichler, Reinhard; Rümmele, Stefan; Woltran, Stefan: A Dynamic-Programming Based ASP-Solver. In (Janhunen, Tomi; Niemelä, Ilkka, Hrsg.): Proceedings of JELIA 2010. Jgg. 6341 in LNCS. Springer, S. 369–372, 2010.

Bernhard Bliem wurde 1988 geboren und studierte an der TU Wien Informatik (Studiengänge *Software & Information Engineering* und *Computational Intelligence*) sowie Philosophie an der Universität Wien. Seine Diplomarbeit wurde mit dem *Distinguished-Young-Alumnus*-Preis der Fakultät für Informatik der TU Wien ausgezeichnet. Das Doktorat in Informatik absolvierte er unter der Betreuung von Prof. Stefan Woltran an der TU Wien. Nach Abschluss seines Doktorats begann Bernhard Bliem als Postdoc an der Universität Helsinki zu forschen. Seine Forschungsinteressen umfassen verschiedene Themen aus der Künstlichen Intelligenz wie etwa Answer Set Programming, SAT Solving, Logikprogrammierung und Wissensrepräsentation, sowie Algorithmik und Komplexitätstheorie.