Universität Potsdam
Naturwissenschaftliche Fakultät
Informatik

Diplom

Untersuchung des Berechnungsaufwands bei Antwortmengenprogrammierung

Bernd Österholz

Bernd_Oesterholz@gmx.de

1. Dezember 2005

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Die Antwortmengenprogrammierung (answer set programming; kurz ASP) ist ein relativ neues Programmierparadigma. Bei ihr werden die Antwortmengen eines logischen Programms gesucht. Die Berechnung einer Antwortmenge ist NP-vollständig, das heißt, der Berechnungsaufwand nimmt im schlimmsten Fall exponentiell mit der Länge der Eingabe zu.

Was bedeutet dies aber konkret für die Berechnung von Antwortmengen?

Stößt man, wenn mit Antwortmengenprogrammierung gearbeitet wird, irgendwann an eine Schwelle, an der man wegen des exponentiellen Wachstums des Berechnungsaufwands nicht mehr weiter kommt?

Gibt es vielleicht Klassen von Problemen, die leichter zu lösen sind?

Wie kann Wissen darüber, welche Problemklassen wie schwer zu lösen sind, dabei helfen, existierende Antwortmengensolver zu verbessern?

Diese und ähnliche Fragen will ich mit dieser Arbeit untersuchen.

1.1 Lesart

Diese Arbeit ist in Kapitel untergegliedert. Für das Verständnis ist es nicht unbedingt erforderlich alle Kapitel zu lesen. Personen mit entsprechendem Vorwissen können die Kapitel 2 und/oder 3 überspringen.

Kapitel 1 bildet eine kurze Einleitung.

In Kapitel 2 werden die Grundlagen der Antwortmengenprogrammierung, soweit sie für diese Arbeit relevant sind, behandelt.

Kapitel 3 beschäftigt sich mit dem Phasenübergang und dem Berechnungsaufwand. Hier werden vorhandene Resultate über den Phasenübergang und den Berechnungsaufwand aus den Bereichen der aussagenlogischen Erfüllbarkeit (SAT) und der Antwortmengenprogrammierung vorgestellt.

Kapitel 4 beinhaltet theoretische Überlegungen zu Suchbäumen, zum Berechnungsaufwand, zur Erfüllbarkeit und ihrer Zusammenhänge bei logischen Programmen. Außerdem werden Prognosen angestellt, über die Auswirkungen bestimmter Parameter.

In Kapitel 5 werden die Generierungsmodelle für logische Programme beschrieben. Es dient zum schnellen Nachschlagen dieser.

Kapitel 6 behandelt die Ergebnisse meiner praktischen Untersuchungen. Die Resultate der durchgeführten Testreihen werden vorgestellt und Erklärungsansätze über die Gründe des gezeigten Verhaltens aufgestellt, beziehungsweise die Ergebnisse werden mit den in Kapitel 4 gemachten Aussagen abgeglichen.

In Kapitel 7 werden die Ergebnisse kurz zusammengefasst und ein Ausblick auf mögliche, zukünftige Arbeiten gegeben.

Der Großteil der Antwortmengenliteratur liegt in englischer Sprache vor. Um den Leser nicht durch ungenaue Übersetzungen zu verwirren, werden manche Begriffe aus der Antwortmengenprogrammierung in Englisch benutzt. Wenn Begriffe in ihrer Bedeutung verwendet werden, für die es aber einen sehr gebräuchlichen englischen Begriff gibt, wird dieser beim ersten Gebrauch hinter dem deutschen Begriff in Klammern angegeben. Bei aus anderen Arbeiten übernommenen Diagrammen, wurde die originale Beschriftung beibehalten, um Fehler durch ungenaue Übersetzungen zu vermeiden.

Programmteile im Text sind im `Schreibmaschinenstil` geschrieben.

Bei Zahlen wird die amerikanische Schreibweise mit „.“ als Dezimaltrennzeichen verwendet, da viele Diagramme mit dem Tool Gnuplot generiert wurden und die Schreibweise der Zahlen von ihm übernommen wird.

Kapitel 2

Grundlagen der Antwortmengenprogrammierung

Dieses Kapitel gibt eine kurze Einführung in die Grundlagen, die für das Verständnis der Antwortmengenprogrammierung im Bezug auf den Berechnungsaufwand und Erfüllbarkeit hilfreich sind.

2.1 Logische Grundlagen

2.1.1 Normale logische Programme

Antwortmengenprogrammierung ist ein Paradigma, das auf verschiedenen logischen Programmen arbeiten kann. Im folgenden werden zwei Arten von logischen Programmen vorgestellt. Quellen für die Definitionen sind [1], [29], [19] und [2].

A_M ist eine Menge von aussagenlogischen Variablen, seine Elemente A_i sind Atome.

Ein normales logisches (normal logic) Programm Π ist ein Menge von normalen Regeln. Eine Regel r für ein normales logisches Programm, hat die folgende Form:

$$\begin{aligned} A_0 &\leftarrow A_1, ..., A_m, not\ A_{m+1}, ..., not\ A_k \quad k \geq m \geq 0 \\ head(r) &= A_0 \\ body(r) &= \{A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_k\} \\ body^+(r) &= \{A_1, \ldots, A_m\} \\ body^-(r) &= \{A_{m+1}, \ldots, A_k\} \end{aligned}$$

A_i ist ein positives Literal und $not\ A_i$ ein negatives Literal.

Die Funktion $head(r)$ gibt den Kopf der Regel r zurück und $body(r)$ den Körper, wobei die Körperliterale als Elemente einer Menge zurückgegeben werden. In

der Menge zu $body^+(r)$ sind alle Atome die positiv im Körper vorkommen enthalten oder der positive Teil des Körpers. Dagegen sind in der Menge zu $body^-(r)$ alle Atome die negiert im Körper vorkommen enthalten oder der negative Teil des Körpers.

Ein normales logisches Programm Π ist ein positives logisches Programm (definite/basic Program), wenn $body^-(r_i) = \emptyset$ für alle Regeln $r_i \in \Pi$ des Programms ist.
Ein Fakt r ist eine Regel mit $body(r) = \emptyset$.

Eine Integritätsbedingung (constraint) ist eine Regel, deren Körper nicht wahr sein kann, um damit bestimmte Wahrheitswertbelegungen auszuschließen. Sie hat die Form: $\leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_k$
Dies ist die Kurzform für: $F \leftarrow not\ F, A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_k$, das Atom F kommt dabei in keiner anderen Regel vor.

2.1.2 Interpretationen

Wahrheitswerte sind die Werte Wahr (true) und Falsch (false).

Eine Interpretation I ist eine Belegung der Atome eines logischen Programms mit Wahrheitswerten, bzw. eine Menge von Atome eines logischen Programms die wahr sind.

Bei einer dreiwertige Interpretation (3-valued Interpretation) kann jedes Atom drei verschiedene Werte einnehmen Wahr, Falsch und Unbekannt (unknown). Repräsentiert durch das Paar $\langle T, F \rangle$, wobei die Menge T die Atome enthält, die Wahr sind, F die, die Falsch sind, und alle Atome die nicht in einer der beiden enthalten sind, Unbekannt sind. Dabei gilt immer $T \cap F = \emptyset$. Eine dreiwertige Interpretation ist total, wenn $T \cup F = A(\Pi)$ ist, ansonsten ist sie partiell, dabei enthält die Menge $A(\Pi)$ alle Atome des logischen Programms Π.

Ein Modell, ist eine totale Interpretation für ein Programm Π bzw. eine Menge von wahren Atomen, welche Π erfüllt oder wahr macht. Die Menge M ist ein minimales Modell von Π, wenn es für Π kein Modell M' gibt mit $M' \subset M$.

2.2 Aussagenlogische Erfüllbarkeit (SAT)

Forschungen rund um das Gebiet der aussagenlogischen Erfüllbarkeit (SAT) existieren schon länger als die Forschungen zum Feld der Antwortmengenprogrammierung. Bei SAT-Problemen, geht es darum, Modelle einer logischen Formel zu finden, beziehungsweise herauszufinden, ob eine Formel überhaupt erfüllbar (SATisfiable) ist. SAT-Probleme gehören zu der Klasse der NP-vollständigen Probleme.

NP-vollständige Probleme haben die Wissenschaft schon die letzten 35 Jahre beschäftigt ([4], [13]). Das vorläufigen Ergebnis ist, dass es anscheinend keinen Algorithmus gibt, der garantieren kann, NP-vollständige Probleme mit polynomiellen oder auch nur subexponentiellen Berechnungsaufwand zu lösen.

Viele NP-vollständige Problemklassen besitzen jedoch Unterklassen die polynomiell zu lösen sind, ein Beispiel dafür ist z.B. 2-SAT und Horn-SAT [13].

Bei Arbeiten zu SAT ist zu beachten, dass dort, anders als bei der Antwortmengenprogrammierung, Atome als Variablen bezeichnet werden.

2.3 Antwortmengenprogrammierung (ASP)

Eine Antwortmenge eines Programms Π, ist ein minimales Modell von Π, auf das, mithilfe der Regeln von Π, geschlossen werden kann. Das heißt, jedes Atom, welches in der Antwortmenge ist, muss im Kopf einer Regel vorkommen, deren Körper wahr ist, auch wenn das Atom mit falsch belegt wird.

Antwortmengen zu finden oder zu entscheiden, dass es keine gibt, ist der Forschungsgegenstand der Antwortmengenprogrammierung oder kurz ASP (Answer Set Programming). Das finden einer Antwortmenge gehört zu der Klasse der NP-vollständigen Probleme ([2]).

2.3.1 Grundinstanzierung von Programme

Ein logisches Programm P kann im prädikatenlogischen Fall allquantifizierte Variablen enthalten. Durch das so genannten Grundinstanzieren (grounding [2]) werden alle Variablen des logischen Programms P ersetzt, durch die Atome beziehungsweise Instanzen, die sie annehmen können. Voraussetzung dafür ist natürlich, dass der Wertebereich, den die Variablen annehmen können, durch eine endliche Menge darstellbar ist. Dadurch entsteht ein logisches Programm Π ohne Variablen.

Zu beachten ist, dass P und Π semantisch das gleiche Programm darstellen.

2.3.2 Gelfond-Lifschitz Transformation

Das Redukt (reduct; siehe [14], [2]) oder die Gelfond-Lifschitz Transformation Π^X eines logischen Programms Π bezüglich einer Menge X von Atomen (Interpretation) ist das Programm, welches entsteht, wenn alle Regeln, in denen Atome aus X negiert vorkommen, gelöscht werden und wenn im daraus resultierenden Programm alle negierten Atome aus dem Körper gelöscht werden.

$$\Pi^X = \{ head(r) \leftarrow body^+(r) \mid r \in \Pi \wedge body^-(r) \cap X = \emptyset \}$$

Das entstehende logisches Programm Π^X enthält keine negativen Literale mehr. Daraus folgt, dass das entstandenen Programm Π^X immer ein eindeutiges, minimales Modell besitzt, da es keine Widersprüche in Π^X mehr geben kann.

2.3.3 *Cn*-Operator

Der *Cn*-Operator ist ein wichtiges Konstrukt in der Antwortmengenprogrammierung, er wird über den T_Π-Operator definiert. Π ist ein positives logisches Programm und X eine Menge von Atomen, die direkte Folge T_Π von X bezüglich Π

ist ([19], [2]):

$$T_\Pi(X) = \{head(r) \mid r \in \Pi \wedge body(r) \subseteq X\} \quad (2.1)$$
$$T_\Pi^0(X) = X \quad (2.2)$$
$$T_\Pi^i(X) = T_\Pi(T_\Pi^{i-1}(X)) \quad i > 0 \quad (2.3)$$

Der T_Π-Operator ist monoton, da aus $X \subseteq Y$ folgt, dass $T_\Pi(X) \subseteq T_\Pi(Y)$ ist. Das heißt, wenn $T_\Pi^0(\emptyset)$ ist, ist $T_\Pi^{i-1}(X) \subseteq T_\Pi^i(X)$
Der Cn-Operator ist wie folgt definiert ([19], [2]):

$$Cn(\Pi) = \bigcup_{i \geq 0} T_\Pi^i(\emptyset) \quad (2.4)$$

Die Menge X ist genau dann ein stabiles (stable) Modell, beziehungsweise eine Antwortmenge von einem logischen Programm Π, wenn $Cn(\Pi^X) = X$ ist.

Der Cn-Operator ist ein Fixpunktoperator, da T_Π^i monoton ist und die Menge X, der Atome die von T_Π^i zurückgegeben werden können, eine endliche Menge ist.

Der Cn-Operator ist so realisierbar, dass er in polynomieller Zeit arbeitet ([9]). Da er die R Regeln eines logischen Programms Π mit A Atomen maximal $\min(R, A)$ mal durchgehen muss, denn bei jedem Durchgang muss mindestens ein neues Atom, das im Kopf einer Regel vorkommt, zur Ausgabemenge neu hinzugenommen werden. Wenn mit T_Π^i kein neues Atom, das im Kopf einer Regel vorkommt, mehr zur Ausgabemenge hinzugenommen wird, brauchen keine weiteren T_Π^j mit $i < j$ mehr berechnet werden, da ab T_Π^i alle Ausgabemengen gleich sind. Der Berechnungsaufwand pro Regel r_i steigt dabei linear mit der Anzahl der Literale $(k_i + 1)$ in ihr.
Demnach ist der maximale Berechnungsaufwand B_{max}:

$$B_{max} = const * min(R, A) * \sum_{i=1}^{R} (k_i + 1)$$

2.3.4 Wohlfundierte Semantik (WFS)

Die wohlfundierte Semantik (well-founded semantics) oder WFS eines normalen logischen Programms Π ist charakterisiert durch eine dreiwertige Interpretation, auch als wohlfundiertes Modell (well-founded model) bezeichnet.

Das wohlfundierte Modell $WFS(\Pi)$ eines normalen logischen Programms Π ist definiert durch ([2]):

$$L = lowerbound$$
$$U = upperbound$$

$$L_0 = \emptyset \quad (2.5)$$

$$
\begin{aligned}
U_0 &= A(\Pi) && (2.6)\\
L_i &= L_{i-1} \cup Cn(\Pi^{U_{i-1}}) && (2.7)\\
U_i &= U_{i-1} \cap Cn(\Pi^{L_{i-1}}) && (2.8)\\
lfp(\Pi) &= \bigcup_{i\geq 0} L_i && (2.9)\\
gfp(\Pi) &= A(\Pi) \setminus \bigcap_{i\geq 0} U_i && (2.10)\\
WFS(\Pi) &= \langle lfp(\Pi), gfp(\Pi)\rangle && (2.11)
\end{aligned}
$$

Die Menge $A(\Pi)$ ist die Menge aller Atome von Π. Die Operatoren lfp (lowest fixpoint) und gfp (greatest fixpoint) sind jeweils die Operatoren zur Bildung des kleinsten und größten Fixpunktes von Π. Alle Atome die in $lfp(\Pi)$ enthalten sind, sind auch in allen Antwortmengen von Π enthalten und alle Atome die in $gfp(\Pi)$ enthalten sind, sind in keiner Antwortmenge von Π enthalten.

Die Operatoren, die zur Definition der wohlfundierte Semantik verwendet werden, können bei der Suche nach Antwortmengen genutzt werden.

Soll die dreiwertige Interpretation $I(\Pi, X, Y)$ aller möglichen Antwortmengen AW_i von Π erzeugt werden, von denen X eine Teilmenge ist ($X \subseteq AW_i$) und die kein Element von $A(\Pi) \setminus Y$ enthalten ($A(\Pi) \setminus Y \cap AW_i = \emptyset$), beziehungsweise alle Atome die nicht in Y sind, sind auch in keiner der Antwortmengen, können die Operatoren der Definition des wohlfundierte Modells dafür als Grundlage dienen. Die Definition der angepassten Operatoren lautet dann:

$$
\begin{aligned}
L_0^* &= X && (2.12)\\
U_0^* &= Y && (2.13)\\
L_i^* &= L_{i-1}^* \cup Cn(\Pi^{U_{i-1}^*}) && (2.14)\\
U_i^* &= U_{i-1}^* \cap Cn(\Pi^{L_{i-1}^*}) && (2.15)\\
lfp(\Pi, X) &= \bigcup_{i\geq 0} L_i^* && (2.16)\\
gfp(\Pi, Y) &= A(\Pi) \setminus \bigcap_{i\geq 0} U_i^* && (2.17)\\
I(\Pi, X, Y) &= \langle lfp(\Pi, X), gfp(\Pi, Y)\rangle && (2.18)
\end{aligned}
$$

Der Hauptunterschied ist, dass die L_i und U_i Operatoren mit den Wert $L_0^* = X$ bzw. $U_0^* = Y$ initialisiert werden. Wird $X = \emptyset$ und $Y = A(\Pi)$ gesetzt, so wird das wohlfundierte Modell von Π berechnet.

Wenn es ein $x \in lfp(\Pi, X)$ gibt, für das auch $x \in gfp(\Pi, Y)$ gilt (bzw. $lfp(\Pi, X) \cap gfp(\Pi, Y) \neq \emptyset$), so besitzt Π für $L_0^* = X$ und $U_0^* = Y$ keine Antwortmenge.

Ist $lfp(\Pi, X) \cup gfp(\Pi, Y) = A(\Pi)$, so ist $lfp(\Pi, X)$ eine Antwortmenge von Π.

Wenn $lfp(\Pi, X) \cup gfp(\Pi, Y) \neq A(\Pi)$ ist, können entsprechende Antwortmengen gesucht werden, indem für ein Atom a, das nicht in $lfp(\Pi, X) \cup gfp(\Pi, Y)$ vorkommt, beide Belegungen ausprobiert werden. Dafür kann es einmal mit den Atomen vereinigt werden, die mit $lfp(\Pi, X)$ als zu den Antwortmengen gehörend

gefunden wurden, indem $L_0^* = (lfp(\Pi, X) \cup \{a\})$ gesetzt wird ($I_1 = (\Pi, (lfp(\Pi, X) \cup \{a\}), Y)$) . Andererseits kann es von den Atomen gelöscht werden, die maximal in den Antwortmengen sein können, indem $U_0^* = ((A(\Pi) \setminus gfp(\Pi, Y)) \setminus \{a\})$ gesetzt wird ($I_2 = (\Pi, X, ((A(\Pi) \setminus gfp(\Pi, Y)) \setminus \{a\}))$) .

Das Anwenden der lfp und gfp Operatoren kann als Schließen bzw. Inferenz gesehen werden, da von den vorhandenen Informationen auf neue geschlossen wird. Wenn ein noch nicht festgelegtes Atom aus $A(\Pi) \setminus (lfp(\Pi, X) \cup gfp(\Pi, Y))$ mit L_0^* vereinigt oder aus U_0^* gelöscht wird, wird eine Entscheidung (choice) vollführt.

Da die Menge L_i^* aus der Vereinigung seines Vorgänges mit einer Menge entsteht, ist sie monoton wachsend. Dagegen ist die Menge U_i^* monoton fallend, da sie aus dem Durchschnitt seines Vorgänges mit einer Menge entsteht. Weiterhin ist die Menge der Atome, die in $Cn(\Pi^X)$ enthalten sein können, endlich, denn es können maximal alle Atome des Programms Π enthalten sein. Das heißt, ab einem bestimmten n ist $L_i^* = L_{i-1}^*$ für $i > n$ und für ein m ist $U_i^* = U_{i-1}^*$ für $i > m$. Dann brauchen natürlich keine weiteren L_i^* und U_i^* mehr gebildet werden.

Das wohlfundierte Modell kann in quadratischer Zeit berechnet werden ([20]).

2.3.5 Abhängigkeitsgraph

Gegeben ist ein logisches Programm Π, der Abhängigkeitsgraph G von Π, ist der folgende gerichtete Graph ([20]): Die Knoten des Graphen sind die Atome des Programms. Alle Knoten p, q für die es eine Regel in Π gibt, in der p der Kopf ist und q positiv im Körper der Regel vorkommt, sind durch eine positive Kante von p nach q verbunden. Wenn q negativ im Körper der Regel vorkommt, sind sie durch eine negative Kante von p nach q verbunden.

2.3.6 Negative Zyklen

Ein ungerader (odd) Zyklus im Abhängigkeitsgraphen enthält eine ungerade Anzahl von negativen Kanten. Ein gerader (even) Zyklus enthält eine gerade Anzahl von negativen Kanten. Normale logische Programme die nur gerade Zyklen besitzen, haben immer ein stabiles Modell, und eines der stabilen Modelle kann in polynomieller Zeit berechnet werden ([20]). Weiterhin wird in [20] gezeigt, dass bei Programmen mit einer begrenzten Anzahl von (nicht trivialen) geraden Zyklen alle Antwortmengen in polynomieller Zeit berechnet werden können. Wenn ein Programm keine solchen Zyklen enthält, besitzt es maximal ein stabiles Modell und damit maximal eine Antwortmenge. Ein Programm mit k (nicht trivialen) geraden Zyklen hat maximal 2^k stabile Modelle.

Im allgemeinen generieren gerade Zyklen Antwortmengen und ungerade Zyklen verringern die Zahl der Antwortmengen.

Wie aussagekräftig Zyklen für den Berechnungsaufwand sind, ist allerdings ungeklärt. Dabei sollte bedacht werden, dass alle möglich Abhängigkeitsgraphen und damit alle möglichen Zyklenzusammensetzungen mit normalen logischen Programmen, die nur aus Regeln mit jeweils einen Körperatom bestehen (1-LP), er-

zeugt werden können. Für die Klasse solcher 1-LP Programme kann allerdings mit polynomiellen Aufwand jeweils eine Antwortmengen berechnet werden oder gesagt werden, dass es keine gibt, was für alle normale logische Programme nicht gilt.

2.3.7 Positiver Abhängigkeitsgraph

Sei ein logisches Programm Π gegeben. Der positive Abhängigkeitsgraph G_p von Π, ist der folgende gerichtete Graph: Die Knoten des Graphen sind die Atome des Programms. Alle Knoten p, q für die es eine Regel in Π gibt, in der p der Kopf ist und q positiv im Körper der Regel vorkommt, sind durch eine Kante von p nach q verbunden.

Der positive Abhängigkeitsgraph von Π entspricht also dem Abhängigkeitsgraph von Π, wenn alle negativen Kanten entfernt werden.

2.3.8 Body-head Abhängigkeitsgraph

Der body-head (Körper-Kopf) Abhängigkeitsgraph ist eine Erweiterung des Abhängigkeitsgraphen, in dem auch die Körper von Regeln berücksichtigt werden.

Der body-head Abhängigkeitsgraph eines gegebenen Programms Π, ist ein gerichteter Graph. Die Knoten des Graphen sind die unterschiedlichen Köpfe und Körper des Programms. Alle Kopfknoten p und Körperknoten b für die es eine Regel in Π gibt, in der p der Kopf und b der Körper der Regel ist, sind durch eine Kante von p nach b verbunden. Alle Kopfknoten p und Körperknoten b für die es eine Regel in Π gibt und p Element $body^+(b)$ ist, sind durch eine positive Kante von b nach p verbunden. Wenn p Element $body^-(b)$ ist, sind sie durch eine negative Kante von b nach p verbunden.

Dabei enthält die Atommenge $body^+(b)$ alle positiven und $body^-(b)$ alle negativen Literale des Körpers b.

2.3.9 Loop-Formeln

Wie im Abschnitt 2.3.6 erläutert, sind Zyklen eine wichtige Struktur in logischen Programmen. Sie bekommen noch eine extra Bedeutung mit dem Zeichen „$\leftarrow$“. Während in der klassischen Logik „$a \leftarrow b$“ die Bedeutung „a wenn b“ hat und damit in „$\neg a \vee b$“ übersetzt werden kann, bedeutet „$a \leftarrow b$“ in ASP „aus b folgt a“, damit kann a nur geschlossen werden, wenn b vorher wahr ist. Nicht erlaubt ist dabei a zu schließen, wenn b nur wahr ist, weil a wahr ist, z.B. kann nicht aus „$a \leftarrow a$“ geschlossen werden das a wahr ist. Dies wird auch üblicherweise bei der Suche nach Antworten als Zirkelschluss abgelehnt. Das Ausschließen solcher Zirkelschlüsse ist der einzige Unterschied bei der Antwortmengensuche zur Suche bei SAT.

Eine Menge von Atomen L des logischen Programms Π wird Loop (Kreis) genannt, wenn es für je zwei Atome p und q aus L einen Pfad im positiven Abhängigkeitsgraphen von Π gibt, der sie verbindet. Zirkelschlüsse können durch solche

Loops entstehen. Durch das Einfügen sogenannter Loop-Formeln (loop formulas) können solche Zirkelschlüsse ausgeschlossen werden und so Antwortmengenprobleme direkt mit SAT-Solvern gelöst werden, beziehungsweise Antwortmengenprobleme auf SAT-Probleme zurückgeführt werden. Näheres dazu in [18],[22] und [29]. Allerdings kann die Übersetzung eines Antwortmengenproblems in ein SAT-Problem mithilfe von Loop-Formeln einen exponentiellen Zuwachs der Größe der Problemdarstellung erzeugen.

2.3.10 Nichtmonotonie

Seien $Cons(\Pi)$ die Konsequenzen eines Programmes, das heißt, der Durchschnitt aller Antwortmengen des Programmes, dann ist ASP in dem Sinne nicht monoton, dass aus $\Pi \subseteq \Pi'$ nicht $Cons(\Pi) \subseteq Cons(\Pi')$ bzw. $Cons(\Pi') \subseteq Cons(\Pi)$ folgt.

Bei SAT sind die Konsequenzen $Cons(F)$ der Durchschnitt durch alle Modelle einer Formel F. Sind bei SAT zwei Mengen von Klauseln (Formeln) F und F' gegeben, so folgt aus $F \subseteq F'$ die Beziehung $Cons(F) \subseteq Cons(F')$.

Der Unterschied zwischen ASP und SAT soll an folgendem Beispiel verdeutlicht werden. Gegeben ist das normale logische Program $\Pi = \{a \leftarrow not\ b\}$ und die entsprechende Formel $F = \{a \vee b\}$, sowie die Mengen $\Pi' = \Pi \cup \{b\}$ und $F' = F \cup \{b\}$. Daraus folgt für die Konsequenzen: $Cons(\Pi) = \{a\}$, $Cons(\Pi') = \{b\}$, $Cons(F) = \emptyset$ und $Cons(F') = \{b\}$, das heißt, $Cons(F) \subseteq Cons(F')$, aber weder $Cons(\Pi) \subseteq Cons(\Pi')$ noch $Cons(\Pi') \subseteq Cons(\Pi)$.

2.4 Berechnung von Antwortmengen

Es wird angenommen ([20]), dass der Berechnungsaufwand für die Suche nach einer Antwortmenge im schlimmsten Fall (worst case) exponentiell mit der Anzahl der Atome wächst. Die Idee, die dahinter steckt ist, dass mit jedem zusätzlichen Atom sich die Größe des (möglichen) Suchraums verdoppelt, es aber im Allgemeinen keine Möglichkeit gibt zu garantieren, dass dieser nicht vollständig durchsucht werden muss. Die Suchraumgröße verdoppelt sich mindestens mit jedem weiteren Atom, da die Anzahl der potentiellen Antwortmengen bzw. die Anzahl der Interpretationen $2^{|A(\Pi)|}$ entspricht und der Suchraum mindestens diese enthalten muss, wobei $A(\Pi)$ die Menge der Atome eines Programms Π ist. Im Falle, dass der ganze Suchraum durchsucht werden muss, verdoppelt sich der Berechnungsaufwand mit jedem neuen Atom. Die Abschätzung für den schlimmsten Fall für das Wachstum des Berechnungsaufwands ist demnach $2^{|A(\Pi)|}$.

Dieser Fall kann sicherlich konstruiert werden. Aber die Wahrscheinlichkeit, dass eine konkrete Problemklasse oder ein konkretes Problem diesem Fall entspricht, ist äußerst gering. Meist sind Atome von anderen abhängig und es müssen dann nicht für jedes Atom beide Fälle, dass es wahr ist oder nicht, überprüft werden. Durch ein neues Atom (mit entsprechenden Regeln) wird sich also normalerweise nicht der Berechnungsaufwand verdoppeln. Es reicht für ein exponentielles Wachstum allerdings aus, dass sich im Durchschnitt der Berechnungsaufwand für

ein neues Atom (mit entsprechenden Regeln) um einen Faktor F größer 1 vergrößert. Dann wäre das Wachstum des Berechnungsaufwands $F^{|A(\Pi)|}$.

Unabhängig davon gibt es bei logischen Programmen Unterklassen, die mit polynomiellen Berechnungsaufwand gelöst werden können, z.B. positive logische Programme.

Die meisten Algorithmen zur Suche von Antwortmengen gehen allgemein gesprochen so vor, dass immer zwei Schritte solange wiederholt werden, bis die geforderten Anzahl von Antwortmengen gefunden wurden oder bewiesen wurde, dass keine (weiteren) Antwortmengen existieren. Der erste Schritt ist dabei auf die Wahrheitswerte aller Atome zu schließen, bei denen dies unter den Voraussetzungen möglich ist. Im zweiten Schritt wird ein noch nicht belegtes Atom ausgewählt, eine Wahrheitswertbelegung an ihm ausprobiert und wenn dies nicht zum Ziel führt, auch die andere Wahrheitswertbelegung ausprobiert. Ausprobiert bedeutet dabei, dass für die entstehende partielle Interpretation eine Antwortmenge oder mehrere gesucht werden. Bei diesem Ausprobieren wird das ursprüngliche Problem in zwei kleinere Probleme aufgespalten. Diese Schritte werden solange wiederholt, bis die geforderte Anzahl von Antwortmengen gefunden wurde oder die Schritte nicht mehr ausführbar sind.

Da bei jedem Aufspalten des Problems, zwar ein Atom weniger zu betrachten ist, dafür aber dann zwei, nur um ein Atom kleinere, Probleme zu betrachten sind, kann diese Antwortmengensuche exponentiellen Berechnungsaufwand erfordern.

Wie viele Antwortmengen berechnet werden sollen, hängt von der Problemstellung ab. Meist wird nur eine Antwortmenge gesucht, es können aber auch alle oder eine bestimmte Anzahl von Antwortmengen gesucht werden.

2.4.1 Antwortmengensuche durch Aufspalten und Begrenzen per WFS

Die Strategie, welche mit Aufspalten und Begrenzen (branch and bound; [2]) arbeitet, ist die meist verbreitete Lösungsstrategie für kombinatorische Suchprobleme.

Die Algorithmen arbeiten auf dreiwertigen Interpretationen. Bei der Initialisierung werden zwei leere Mengen als Mengen für die wahren (T) und falschen (F) Atome übergeben und das aussagenlogische Programm Π. Dann werden zwei Schritte solange wiederholt, bis genügend Antwortmengen entstanden sind oder keine mehr erzeugt werden kann. Die Menge T ist eine Antwortmenge, wenn die Interpretation $I = \langle T, F\rangle$ total ist, beziehungsweise die Vereinigung $T \cup F = A(\Pi)$ alle Atome $A(\Pi)$ aus Π enthält. Die Menge T kann durch weitere Berechnungen keine Antwortmenge mehr werden, wenn der Durchschnitt $T \cap F$ nicht leer ist, also ein Atom sowohl wahr als auch falsch sein müsste.

Begonnen wird mit der Interpretation $I = \langle T, F\rangle = \langle \emptyset, \emptyset\rangle$.

Schritt 1: Es wird $I = \langle lfp(\Pi, T), gfp(\Pi, A(\Pi) \setminus F)\rangle$ mithilfe der Operatoren der wohlfundierte Semantik aus Abschnitt 2.3.4 berechnet, wobei $L_0^* = T$ und $U_0^* = A(\Pi) \setminus F$ gesetzt wird. Entsteht dabei eine Antwortmenge, wird diese

zurückgegeben, ist keine mehr möglich bzw. es tritt ein Konflikt auf, wird Falsch zurückgegeben, ansonsten wird zu Schritt 2 übergegangen.

Schritt 2: Dann wird ein Atom ausgesucht, das in $(A(P) \setminus (T \cup F))$ ist. Mit diesem Atom werden zwei neue Interpretationen aus der Interpretation $I = \langle T, F \rangle$ gebildet, wobei bei einer von den neuen Interpretationen das Atom mit T vereinigt wird und bei der anderen mit F. Mit diesem beiden Interpretationen wird jeweils Schritt 1 wiederholt, bis genügend Antwortmengen berechnet worden sind oder es keine weiteren Antwortmengen mehr gibt. Dabei muss eventuell Schritt 1 auch nur mit einer der beiden wiederholt werden.

Die erzeugten Antwortmengen werden zurückgegeben.

Schritt 2 wird dabei choice (Entscheidung) genannt.

Diese Art der Antwortmengensuche kann durch weitere Optimierungen ergänzt werden. Für mehr Details siehe [2].

Im Grunde können in Schritt 1 auch andere Operatoren, als die der wohlfundierte Semantik, eingesetzt werden. Wichtig dabei ist, dass nur etwas geschlossen wird, was logisch folgt. Die eingesetzten Operationen müssen also die Vollständigkeit erhalten.

2.4.2 Systeme

Beide nachfolgend ASP-Systeme arbeiten mit dem Aufspalten und Begrenzen Ansatz auf erweiterten aussagenlogischen Programmen.

Mit dem Programm LParse können dabei Programme mit Variablen in solche erweiterten aussagenlogischen Programme umgewandelt werden. LParse nimmt ein logisches Programm, bildet die aussagenlogischen Version des Programms, löscht einige überflüssige Teile (z.B. doppelte Regeln) und wandelt es in eine für die Systeme leichter lesbare Form um.

2.4.2.1 Smodels

Smodels [27] ist ein Antwortmengensolver von Patrik Simons, welcher auf einer Erweiterung von normalen logischen Programmen arbeitet.

Smodels arbeitet mit dem Aufspalten und Begrenzen mit dem WFS-Ansatz in Abschnitt 2.4.1 auf Seite 11. Zum Einsatz kommen dabei auch eine Heuristik und Operationen welche die Suche beschleunigen sollen, eine solche Operation wäre z.B. das Löschen von überflüssigen Regeln und Literalen. Der Smodels-Algorithmus arbeitet ähnlich wie der DPLL-Algorithmus für SAT-Probleme.

```
1 function smodels(P,I)
2     I := expand(P,I)
3     I := lookahead(P,I)
4     if conflict(P,I) then
5         return false
```

```
6     else
7        if I covers Atoms(P) then
8           return true {T a stable model}
9        else
10          x := heuristic(P,I)
11          if smodels(P,I U {x}) then
12             return true
13          else
14             return smodels(P,I U {not(x)})
15          end if
16       end if
17    end if.
```

Listing 2.1: Smodels

Listing 2.1 übernommen aus [27] zeigt den Hauptalgorithmus von Smodels, dabei ist `P` ein logische Programm und $I = \langle T, F \rangle$ eine dreiwertige Interpretation.

Die Funktionen `expand(P,I)` und `lookahead(P,I)` realisieren Schritt 1 aus Abschnitt 2.4.1. Die `expand(P,I)` Funktion verwendet dabei die Operatoren der wohlfundierte Semantik und die Funktion `lookahead(P,I)` schaut voraus, indem sie Atome versuchsweise mit Wahrheitswerten belegt. Die Funktion `conflict(P,I)` testet, ob die Interpretation I wiedersprüchlich ist (ein Atom ist sowohl wahr als auch falsch). Mit der zusammengesetzten Funktion `I covers Atoms(P)` wird geprüft, ob in der Interpretation I alle Atome des Programms belegt sind, beziehungsweise ob I total ist. Mit der Funktion `heuristic(P,I)` wird ein Atom für Schritt 2 aus Abschnitt 2.4.1 ausgesucht. Die Vereinigung von Mengen wird durch die Funktion `U` realisiert.

Für mehr Details sei auf [27] verwiesen.

2.4.2.2 Nomore++

Bei dem am Institut Informatik, Lehrstuhl Wissensverarbeitung, an der Universität Potsdam entwickelten Antwortmengensolver NoMoRe [1] handelt es sich um einen auf Graphen basierten Ansatz. Ein Solver, der auf diesem Ansatz aufbaut, ist der am gleichen Lehrstuhl entwickelte Nomore++ Solver.

Grundlage dafür ist der body-head Abhängikeitsgraph eines logischen Programms. Über Einfärbungen werden die einzelnen Knoten mit Werten belegt. Wenn alle Kopfknoten mit einem Wahrheitswert Wahr oder Falsch belegt wurden, wurde eine Antwortmenge gefunden. Wenn dies aufgrund von Widersprüchen nicht möglich ist, gibt es keine Antwortmenge. Das System Nomore++ arbeitet mit einem erweiterten Aufspalten und Begrenzen Ansatz aus 2.4.1. Zusätzlich zu den Belegungen der Atome, beziehungsweise der Köpfe, können auch den Körpern der Regeln Werte zugeordnet werden. Dadurch gibt es nicht nur die Möglichkeit an Köpfen aufzuspalten, beziehungsweise einen choice zu machen, sondern auch für Körper. Der potentielle Suchraum wird dadurch natürlich größer, aber er kann

auch stärker eingeschränkt werden. Für eine genaue Beschreibung von Nomore++ siehe [1].

Kapitel 3

Phasenübergang und Berechnungsaufwand

Der Phasenübergang beschreibt eine Region bezüglich eines Parameters, in der ein anderer Parameter eine zu seinem sonstigen Verhalten relativ starken Sprung in seinen Werten durchmacht. Ein Beispiel ist der Phasenübergang des Wassers von fest zu flüssig. Während die Temperatur allmählich steigt, nimmt beim Schmelzpunkt die Festigkeit des Wassers plötzlich stark ab. Phasenübergänge wurden auch in anderen Bereichen außerhalb der Physik gefunden, z.B. der Mathematik, Chemie oder Informatik.

Oft haben diese Phasenübergänge auch einen Einfluss auf andere Parameter. Während beim Wasser innerhalb der Phasen die Energiemenge, die nötig ist, um eine Erwährmung um ein Grad zu erzielen, ungefähr konstant ist, nimmt diese Energiemenge bei den Phasenübergängen plötzlich rapide zu.

Ähnliches ist auch häufig für Phasenübergänge bei Berechnungsproblemen für den Berechnungsaufwand zu beobachten. Im Bereich von Phasenübergängen (z.B. von lösbaren zu nicht lösbaren Problemen) ist der Berechnungsaufwand oft höher.

Wie in [13] bemerkt, können viele NP-vollständige Probleme durch einen Parameter charakterisiert sein, der angibt wie stark das Problem die Lösungsmöglichkeiten einschränkt. Wenn die Problemklassen bezüglich dieses Parameters betrachtet werden, gibt es einen Phasenübergang von meist erfüllbar zu meist nicht erfüllbar. In vielen Fällen kann ein (Skalierungs-) Parameter gefunden werden, der durch eine Skalierungsfunktion aus anderen Parametern gebildet wird, so dass die Kurven für die Erfüllbarkeit sich bezüglich diesem in einem Bereich schneiden und dort einen Phasenübergang haben. Wenn sich Kurven in einem Punkt überschneiden, wird dieser Überschneidungspunkt (crossover point) genannt. Bezüglich eines solchen Skalierungsparameters sind auch Verläufe von Kurven für andere Parameter (z.B. Zeit) ähnlich, das heißt z.B., dass die Kurven bezüglich des Skalierungsparameters ungefähr bei dem gleichen Wert ein Maximum aufweisen.

3.1 Graphenprobleme

Bei der Suche nach Hamiltonschen Zyklen in ungerichteten Graphen mit per Zufall generierten Kanten ist der Skalierungsparameter durch die Skalierungsformel $\frac{E}{N \log N}$ bestimmt [13], wobei E die Anzahl der Kanten (edges) im Graph ist und N die Anzahl der Knoten (nodes).

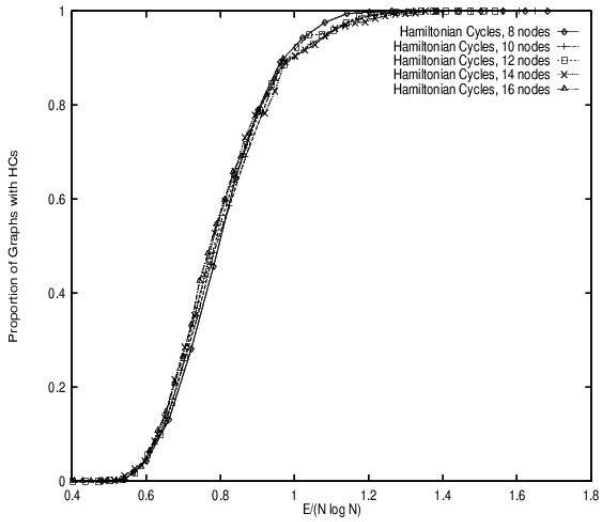

Abbildung 3.1: Erfüllbarkeit bei Hamiltonschen Zyklen in Zufallsgraphen

Bezüglich dieses Parameters gibt es einen Überschneidungspunkt bei der Erfüllbarkeitskurve, wie in Abbildung 3.1 aus [13] Seite 3 für verschiedene Anzahlen von Knoten N (nodes) zu sehen. Die y-Achse zeigt den Anteil der Graphen die einen Hamiltonschen Zyklus besitzen. Die Daten von [13] wurden mithilfe eines von Jeremy Frank und Charles U. Martel selbst implementierten Aufspalten und Begrenzen (branch-and-bound) Algorithmus erhoben. Deutlich erkennbar ist, dass mit Zunahme der Anzahl der Kanten bei rund 0.7, die Anzahl der Graphen, die Hamiltonsche Zyklen besitzen, stark zunimmt, von einem Bereich (Phase 1), in dem so gut wie kein Graph einen Hamiltonschen Zyklus hat, zu einem Bereich (Phase 2), wo fast alle Graphen Hamiltonsche Zyklen besitzen.

In der Nähe dieses Phasenübergangs wurde auch das Maximum des Berechnungsaufwandes gefunden. Dies ist in Abbildung 3.2 aus [13] zu sehen. Auf der y-Achse sind die Backtracks des in [13] verwendeten Algorithmus skaliert aufgetragen, dabei wurden zur Skalierung die Werte der Testreihen zu den einzelnen Knotenanzahlen jeweils durch die Maximalwerte der Testreihen dividiert (skalierter Testreihenwert=(Testreihenwert)/(maximaler Testreihenwert)) .

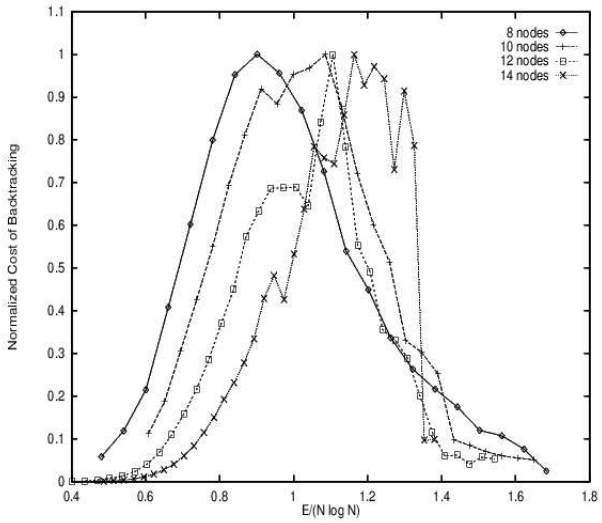

Abbildung 3.2: Backtracks bei Hamiltonschen Zyklen in Zufallsgraphen

3.2 Aussagenlogische Erfüllbarkeit (SAT)

Der Berechnungsaufwand und die Verteilung von logischen Formeln mit und ohne Modellen bei SAT-Solvern wurde schon mehrfach untersucht, beispielsweise in [24], [6], [25], [5] und [12]. In [12] sind einige Wahrscheinlichkeitsuntersuchungen hinsichtlich des Berechnungsaufwands bei SAT-Problemklassen zu finden. Das Ergebnis dieser Untersuchungen ist, dass große Bereiche des Parameterraums Probleme darstellen, die im Durchschnitt mit einer Wahrscheinlichkeit gegen 1 mit polynomiell wachsenden Aufwand zu lösen sind.

In [24], [6], [25] und [5] wurden statistische Untersuchung anhand verschiedener Modelle gemacht.

Beim fixed clause-length (feste Klausellänge) oder k-SAT Modell, besitzt jede Klausel eine feste Anzahl k von Literalen. Die Literale werden aus der Menge der Variablen zufällig ausgewählt und mit einer Wahrscheinlichkeit von 0.5 negiert. Beim constant-probability (konstante Wahrscheinlichkeit) Modell, hat jede Variable eine konstante Wahrscheinlichkeit in einer Klausel aufgenommen zu werden, wobei die Wahrscheinlichkeit, dass sie negiert aufgenommen wird, wieder 0.5 ist. Die Auswahl geschieht natürlich immer zufällig. Eine Erweiterung zum constant-probability Modell ist das εk-SAT Modell, bei dem trivial zu lösende Klauseln verboten sind (z.B. die Klausel (a) oder $(a \vee not\ a)$). Eine andere Erweiterung ist das $[k, l]$-SAT Modell, bei dem die Klausellänge gleichmäßig zwischen der Länge k und l verteilt ist.

Die restlichen Parameter beider Modelle sind Anzahl der Klauseln und Variablen für die Probleme bzw. Formeln. Für eine Testreihe in [24], [25] und [5] wurde eine Anzahl von Testpunkten mit jeweils mehreren Problemen (in der Größenord-

nung von 1000) mit festen Parameterwerten generiert, wobei sich die verschiedenen Testpunkte in einem Parameterwert (meist der Klauselzahl) unterschieden.

Zur Berechnung der Modelle wurde der DP- und der DPLL-Algorithmus verwendet. Der DPLL-Algorithmus ist der von Davis, Logemann und Loveland in [7] beschriebene Algorithmus. Er baut auf dem in [8] beschriebenen ursprünglichen DP-Algorithmus von Davis und Putnam auf. Beim ursprünglichen DP-Algorithmus werden Variablen einer Formel nacheinander durch Resolution eliminiert, wobei alle möglichen Resolventen einer gewählten Variable gebildet und alle Klauseln, welche die gewählte Variable enthalten, gelöscht werden.

Beim DPLL-Algorithmus wurde der Eliminationsschritt durch einen Aufspalten-Schritt (bzw. choice bei dem beide Variablenbelegungen ausprobiert werden) ersetzt. Auf diese Weise entstehen für eine gewählte Variable zwei kleinere Unterprobleme anstatt eines großen erweiterten Problems. Der entstehende Algorithmus macht eine tiefen-zuerst (depth-first) Suche mit Backtracking (Zurückverfolgen) durch die (Teil-) Wahrheitswertbelegungen. Der Algorithmus ist in [5] Seite 3 zu finden.

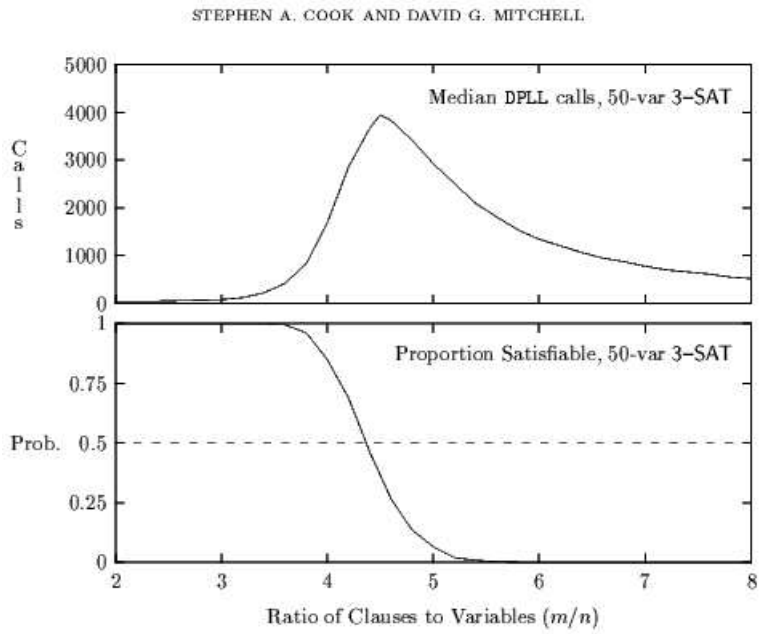

Abbildung 3.3: Performanz des DPLL-Algorithmus und Erfüllbarkeit für 3-SAT mit 50 Variablen

In Abbildung 3.3, übernommen aus [5], ist ein typisches Ergebnis zu sehen. Verwendet wurde dabei das fixed clause-length Modell mit 3 Literalen (3-SAT) und 50 Variablen. Auf der x-Achse ist dabei der Parameter $m/n = Klauseln/Variablen$ aufgetragen, da sich dieser als charakteristischer Skalierungsparameter gezeigt hat.

Im oberen Diagramm sind auf der y-Achse der Median der rekursiven Aufrufe (calls) des DPLL-Algorithmus aufgeführt, wodurch der Berechnungsaufwand reflektiert wird. Bei einer sortierten Reihe von Messwerten ist der Median der Wert, der an der mittleren Position steht. Die y-Achse des unteren Diagrammes zeigt den Anteil der lösbaren Probleme des Testpunktes an. Im unterem Diagramm ist deutlich ein Phasenübergang zu erkennen, von einem Bereich mit überwiegend lösbaren, zu einem Bereich mit überwiegend nicht lösbaren Problemen. Dieser Phasenübergang liegt zwischen den m/n Werten 4 und 5.

Im oberen Diagramm des Berechnungsaufwands ist ein leicht-schwer-leicht Muster (easy-hard-easy pattern) zu erkennen. Wobei das Maximum ungefähr bei $m/n = 4.5$ liegt, dort wo die Kurve im unterem Diagramm den 0.5 Wert erreicht.

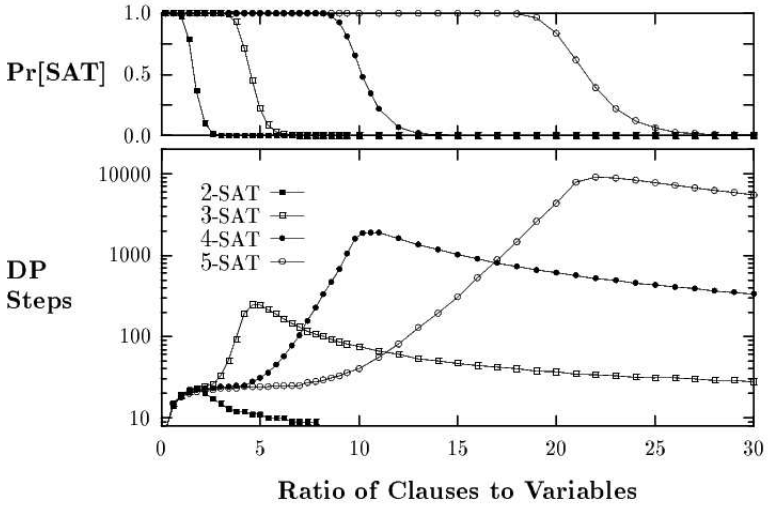

Abbildung 3.4: Median steps des DP-Algorithmus für k-SAT mit ausgewählten Werten von k

In Abbildung 3.4, übernommen aus [24] Seite 4, sind die Diagramme mit den Kurven für $n = 25$ Variablen und verschiedene Werte für die Anzahl k der Literale pro Klausel zu sehen. Dabei zeigt diesmal das untere Diagramm den Median des Berechnungsaufwandes in Form von logarithmisch aufgetragenen Median der DP steps, der DP-Algorithmus ist in [24] auf Seite 2 zu finden. Der verwendete DP-Algorithmus entspricht dabei dem DPLL-Algorithmus, ohne die „pure literal“ Regel, ist also ein DPLL-Algorithmus und nicht der ursprüngliche Resolutions DP-Algorithmus.

Auch hier tauchen wieder die gleichen Muster auf. Weiterhin ist zu erkennen, dass mit zunehmenden k auch das Maximum des Berechnungsaufwandsmedians und der Kurvendurchgang durch den 0.5 Wert im oberen Diagramm nach rechts wandern. Dies entspricht einer Zunahme des m/n Parameters. In dem unteren Diagramm ist weiterhin eine Zunahme des Maximums zu sehen.

Die 2-SAT Probleme bilden eine Ausnahme. Wie Beispielsweise in [24] dargestellt, sind 2-SAT Probleme in $O(n + m)$ Zeit zu lösen.

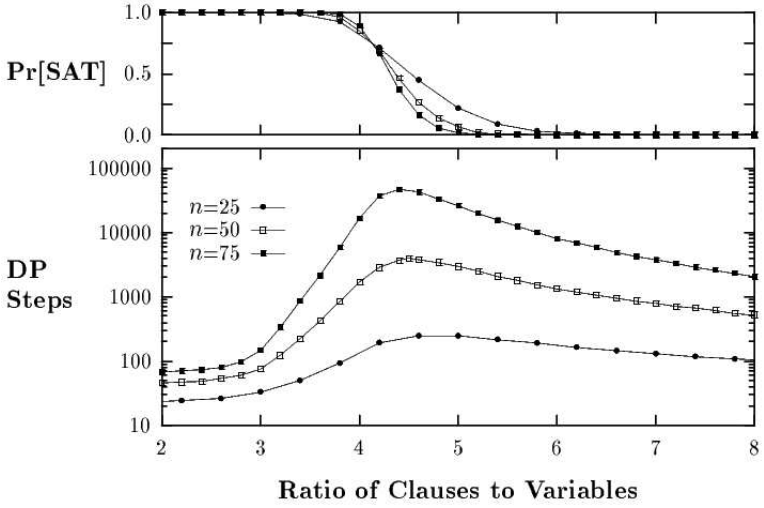

Abbildung 3.5: Steps des DP-Algorithmus für 3-SAT mit ausgewählten Werten von Variablen n

In Abbildung 3.5, übernommen aus [24] Seite 6, sind die Diagramme mit den Kurven für verschiedene Anzahlen von Variablen ($n = 25; 50; 75$) und 3 Literalen pro Klausel (3-SAT) zu sehen. Die Art der Diagramme entspricht dabei denen in Abbildung 3.4 . Ersichtlich hierbei ist, dass sowohl das Maximum beim Berechnungsaufwand, als auch der Phasenübergang bei der Erfüllbarkeit bei ungefähr den gleichen m/n Werten liegen. In beiden Diagrammen werden die Kurven mit zunehmenden n steiler. Desweiteren wächst der Berechnungsaufwand mit zunehmenden n exponentiell.

In [25] wird vermutet, dass mit dem constant-probability (konstant Wahrscheinlichkeit) Modell keine schweren Probleme generiert werden können und es damit für den Test von SAT-Solvern ungeeignet ist.

In [24] wird das εk-SAT Modell untersucht, das ein ähnliches Verhalten zeigt wie das k-SAT Modell, abgesehen davon, dass die generierten Probleme wesentlich einfacher sind. Auch andere Modelle zeigen ein zum k-SAT Modell ähnliches Verhalten.

Das leicht-schwer-leicht Muster wird in [25] damit begründet, dass in Formeln, mit wenigen Klauseln pro Variable, die legalen Variablenbelegungen wenig beschränkt werden und so die meisten Belegungen Lösungen sind. Bei vielen Klauseln pro Variable, sind die legalen Variablenbelegungen zu sehr beschränkt und es kann schnell ermittelt werden, dass es keine erfüllenden Belegungen gibt. Dazwischen, im schweren Bereich, müssen viele Variablenbelegunge geprüft werden, um herauszufinden, ob es eine Lösung gibt. Da bei der Erfüllbarkeitswahrscheinlich-

keit von 0.5 am schlechtesten vorausgesagt werden kann, ob die Formel erfüllbar ist oder nicht, liegt dort der schwierige Bereich. In [24] wird dazu noch die Ergänzung gemacht, dass das Maximum sich durchaus etwas verschieben kann, wenn z.B. der Algorithmus dazu ausgelegt wurde erfüllbare Formeln schneller zu lösen.

Bei der Frage, ob zufällig generierte Probleme als Test für SAT-Solver benutzbar sind, gehen die Meinungen auseinander. Hauptargument dafür, ob sie benutzbar sind, ist anscheinend, dass schwere Probleme generiert werden können. Während in Mitchell's Arbeiten davon ausgegangen wird, dass zumindest der schwere Bereich bei k-SAT für $k > 2$ dafür durchaus geeignet ist, wurden in anderen Arbeiten keine passenden Parameter dafür gefunden.

3.3 Antwortmengenprogrammierung (ASP)

In [29] Kapitel 4 und [21] wurden von Yuting Zhao Phasenübergänge bei der Antwortmengenberechnung untersucht. Darin werden drei ASP-Systeme benutzt, Smodels 2.27, DLV (Mai 16, 2003 Version) und ASSAT 2.0, der mit dem SAT-Solver Chaff2 arbeitet.

Bei seinem fixed bodylength (feste Körperlänge) Modell k-LP haben die Regeln die Länge k, das heißt $(k-1)$ Körperatome. Das Modell ist an das fixed clauselength Modell k-SAT bei SAT angelehnt.

Yuting Zhao verwendet, angelehnt an SAT, L als Nummer der Regeln und N als Nummer der Atome.

Diagramm 3.6 ist von [29] Seite 82 übernommen. Es handelt sich dabei um sein 3-LP Modell mit 150 Atomen. Auf der x-Achse ist der L/N (= $Regeln/Atome$ = R/A) Faktor aufgetragen, $L/N = 5$ heißt z.B. die Programmklasse mit $N = 150$ Atomen und $L = 150 * 5 = 750$ Regeln. Auf der linken Seite ist die Wahrscheinlichkeit „pro“ (=probability) aufgetragen, mit der in der Klasse die Programme erfüllbar sind, beziehungsweise mindestens eine Antwortmenge haben. Dazu gehört die mit „pro“ bezeichnete Kurve. Auf der rechten Seite ist die Zeitachse in Sekunden aufgetragen. Sie dient den restlichen Kurven, welche die unterschiedlichen Durchschnittszeiten zeigen. Die Testläufe wurden mit drei verschiedenen ASP-Solvern durchgeführt, DLV, Smodels und ASSAT. Die Kurven, deren Legendenbeschriftung mit „D“ beginnt gehören zum DLV-Solver, „S“ steht für Smodels und „A“ für ASSAT. Die Endung „total“ bedeutet, dass es sich um alle generierten Probleme handelt. Die „no“ Kurven beinhalten nur Daten von Problemen, ohne Antwortmengen (unerfüllbare Probleme). Die „has“ Kurve bezieht sich auf Probleme mit Antwortmengen (erfüllbare Probleme), sie zeigt die Zeit, die benötigt wurde um eine Antwortmenge zu finden.

Wie zu sehen ist, unterscheiden sich die drei Solver deutlich in der Zeit, um eine Antwortmenge zu finden oder zu entscheiden, dass es keine gibt. Der DLV-Solver schneidet am schlechtesten ab, er ist aber auch dafür ausgelegt, mit der mächtigsten Sprache der drei Solver zu arbeiten. ASSAT schneidet am besten ab, da es von den Erfahrungen, die bisher mit SAT-Solvern gemacht wurden, profitiert. Allerdings

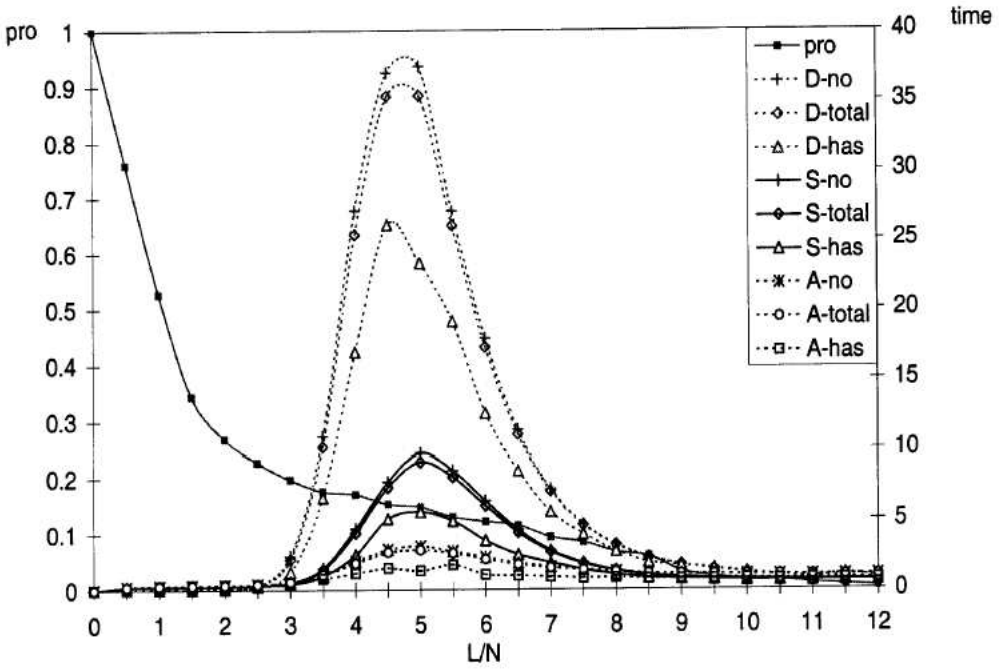

Abbildung 3.6: Erfüllbarkeit und durchschnittliche Zeit für 3-LP bei 150 Atomen

ist es mit dieser Version von ASSAT, im Gegensatz zu den anderen Solvern, noch unmöglich weitere Antwortmengen zu finden. Wenn aber von den konkreten Berechnungszeiten abgesehen wird, ähneln sich die Kurven der drei Solver doch sehr. Eine logarithmische Einteilung der Zeitskale hätte die Ähnlichkeiten im Verlauf wohl noch besser sichtbar gemacht.

Interessant ist, dass alle drei Solver ein leicht-schwer-leicht Muster aufweisen und ungefähr an der gleichen Stelle bei $L/N = 5$ ein Maximum besitzen. Die Schwierigkeit der Problemlösung liegt anscheinend in den Problemen selbst und ist weniger von den Solvern abhängig.

In [29] wird weiterhin bemerkt, dass das Maximum der Zeitkurve in einer Region liegt, in der die meisten Probleme nicht lösbar sind (im Bereich in dem annähernd 10% bis 20% der Probleme lösbar sind, wie an der „pro“ Kurve des Diagramms zu sehen) und nicht wie bei SAT, in der Übergangsregion von lösbaren zu nicht lösbaren Problemen. Der Grund wird von Yuting Zhao in der Nichtmonotonität von ASP gesehen.

Im Vergleich zu dem Diagramm 3.5, für die entsprechende Klasse von SAT-Problemen, fällt auf, dass das Maximum sich bei ähnlichen L/N Werten befindet. Während sich bei SAT die Kurve für den Anteil der erfüllbaren Probleme bei wenigen Klauseln pro Atom in der Nähe von 1 aufhält, sinkt die entsprechende Kurve bei ASP gleich stark ab. In diesem Sinne gibt es bei ASP anscheinend keinen Pha-

senübergang bei der Erfüllbarkeit, da es nur eine Phase der größtenteils unerfüllbaren Probleme gibt.

Aufgrund der Ähnlichkeit zu den SAT-Problemen kann aber angenommen werden, dass sich auch hinter dem Maximum bei den ASP-Zeitkurven ein versteckter Phasenübergang eines Parameters verbirgt, der nicht erfasst oder abgebildet wurde.

Weiterhin ist in diesem Modell schwerer herauszufinden, dass Probleme unerfüllbar sind, als das Probleme erfüllbar sind.

In [29] wird festgestellt, dass mit steigender Anzahl von Atomen die Kurven wie bei SAT ausgeprägter werden. Die Erfüllbarkeitskurven fallen beim Phasenübergang stärker ab und die Berechnungsaufwandskurven haben einen größeren Bogen im schweren Bereich. Auch dies ist unabhängig vom verwendeten Solver.

Beim Mischen von Regeln mit den Körperlängen 2 und 3 in einem Verhältnis 1 zu 1, wiederholen sich die Ergebnisse ([29]), wenn von den konkreten L/N Werten abgesehen wird. Die Kurven sind in x-Achsen Richtung (L/N Achse) nur gedehnt (beziehungsweise gestaucht).

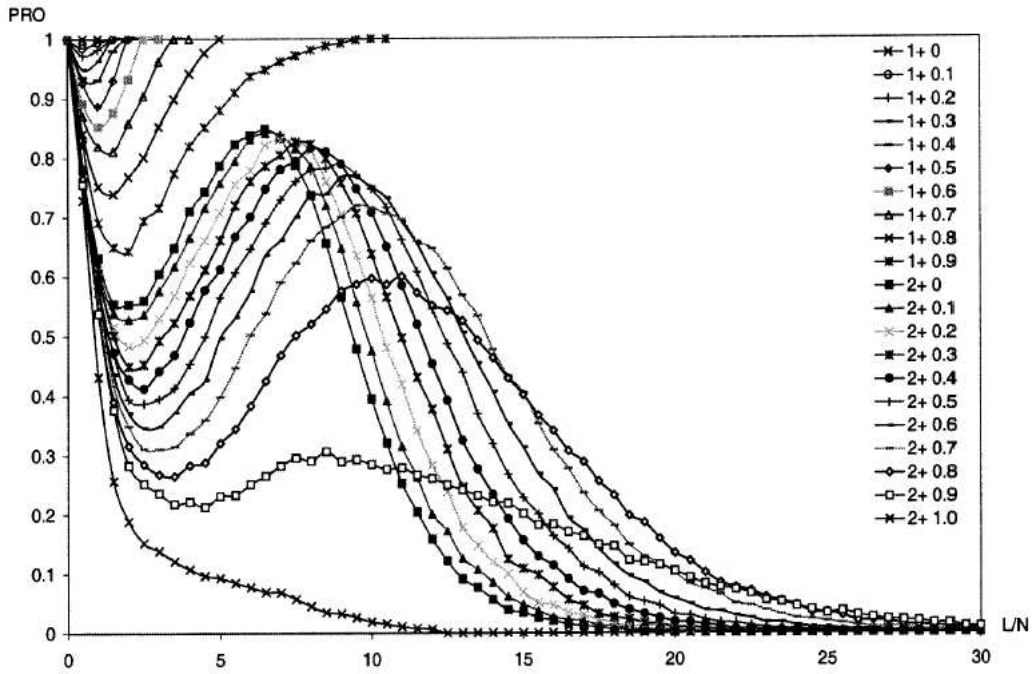

Abbildung 3.7: Erfüllbarkeit von 1 + p-LP und 2 + p-LP bei 100 Atomen

Ein weiteres Modell ist das $k+p$-LP Modell, wobei der Anteil p der Regeln aus Regeln mit der Körperlänge $(k-1)$ und der Rest aus Regeln mit der Körperlänge k besteht. Grafik 3.7 (aus [29] Seite 94) gibt die Erfüllbarkeitswahrscheinlichkeiten bei 100 Atomen mit unterschiedlichen k und p Werten an.

Alle Kurven, außer der für 2 + 1-LP, zeigen ein nicht monotones Verhalten. Der Grund für den dargestellten Kurvenverlauf liegt wohl in der Nichtmonotonität von ASP.

Das Verhalten bei dem 1 + p-LP Modell bei hohen L/N Werten beruht wahrscheinlich darauf, dass Fakten nicht widerlegt werden können. Wenn ein Programm also genug Fakten beinhaltet, um mögliche Widersprüche außer Kraft zu setzen, ist es erfüllbar. Die Wahrscheinlichkeit, dass ein Atom ein Fakt ist, nimmt in dem 1 + p-LP Modell mit steigendem L/N Faktor zu.

Das fallen-steigen-fallen (drop-rise-drop) Muster bei den 2 + p-LP Modellen wird mit steigender Anzahl von Atomen ausgeprägter ([29] Seite 90).

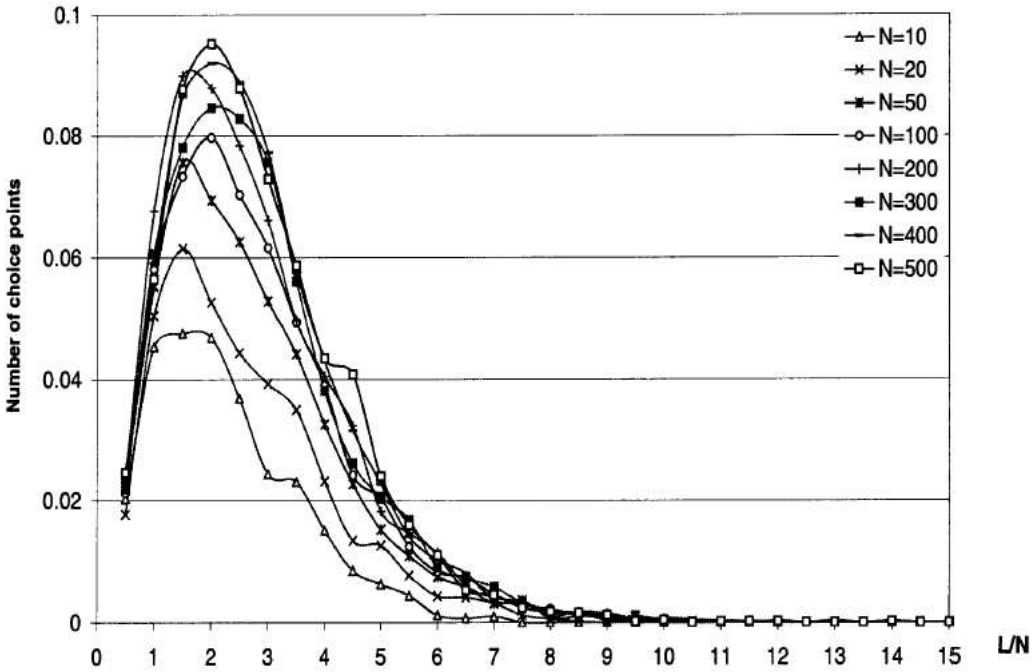

Abbildung 3.8: Durchschnittliche choices für 2-LP mit ausgewählten Anzahlen N von Atomen

Abbildung 3.8 aus [29] Seite 92 zeigt die durchschnittliche Anzahl von choices bei dem 2 + 0-LP Modell aus [29]) bei unterschiedlichen Werten für die Anzahl der Atome N. Auch hier zeigt sich das leicht-schwer-leicht Muster, dabei liegt das Maximum um $L/N = 2$. Allerdings liegt das Maximum aller Kurven bei nur unter durchschnittlich 0.1 choice points, dass heißt unter durchschnittlich 1 choice point auf 10 Problemen, die Probleme sind also trivial.

Kapitel 4

Theoretische Überlegungen

Dieses Kapitel beinhaltet einige allgemeine theoretische Überlegungen, die für die Auswertung und das Verständnis der Testergebnisse nützlich sind. Komplexe mathematische Theorien sind jeweils nur auf bestimmte Antwortmengenklassen anzuwenden, zum Beispiel auf zufällig erzeugte Programme mit fester Körperlänge. Es ist aber eine allgemeine Charakterisierung des Begriffs „Suchaufwand“ sinnvoll.

Viele der Überlegungen dürften unter Fachleuten bekannt sein, sind aber trotzdem kaum in der Literatur zu finden.

4.1 Suchbaum

Im Nachfolgenden sind theoretische Überlegungen aufgeführt, welche die Berechnungsaufwand der vollständigen Aufspalten und Begrenzen (branch and bound) Algorithmen, im Hinblick auf deren Suchbaum, beleuchten. Dabei wird mit einfachen Überlegungen begonnen, die dann erweitert werden.

Ausgegangen wird von einem logisches Programm Π mit A Atomen. Damit ist die Anzahl der möglichen Interpretationen $|I| = 2^A$.

Der Suchbaum (siehe Suchbaumbeispiel in Abbildung 4.1) ist ein Baum, dessen Knoten, ohne die Blätterknoten der Ebene A, für einzelne Atome stehen, über die, bei der Suche an dieser Stelle, Aussagen gemacht werden. Die Wurzel des Suchbaums wird als oben bezeichnet und oben dargestellt. Der Wurzelknoten symbolisiert das erste Atom, über das bei der Suche eine Aussage getroffen wird. Die Aussagen werden durch die abgehenden Kanten symbolisiert. Eine abgehende Kante eines Knotens, ist eine Kante die den Knoten enthält und nicht in Richtung des Wurzelknotens geht. Der Einfachheit halber kann die Vorstellung verwendet werden, dass die Kante, die nach links von einem Knoten abgeht, bedeutet, dass das zugehörige Atom mit Falsch (false) belegt wird, und die Kante, die nach rechts abgeht, bedeutet, dass das Atom wird mit Wahr (true) belegt wird. Über jedes Atom, über das eine Aussage getroffen wurde, kann keine weitere Aussage getroffen werden. Ein Atom das mit Falsch belegt wurde, kann nicht mehr mit Wahr belegt werden und umgekehrt. So taucht jedes Atom von der Wurzel bis zu einem Blatt

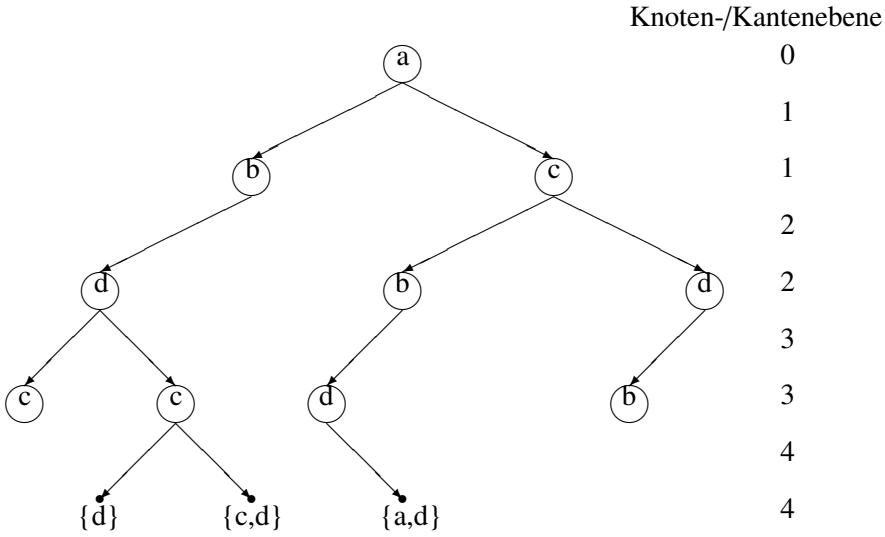

Abbildung 4.1: Suchbaumbeispiel

höchstens einmal auf.

Die Knotenebene n im Suchbaum enthält alle Knoten, die n Kanten vom Wurzelknoten entfernt sind. In der Kantenebene n im Suchbaum sind die Kanten, die über $n-1$ Kanten mit dem Wurzelknoten verbunden sind, bzw. die Kanten, die zur Knotenebene n führen. Die mit einem Knoten assoziierte Belegung besteht aus den Aussagen zu den Atomen, die durch den Pfad vom Wurzelknoten zu dem Knoten repräsentiert werden. Die Blätter der Ebene A repräsentieren die Antwortmengen des Programms Π.

Von jedem Knoten, der ein Atom repräsentiert, können maximal zwei Kanten ausgehen, da über ein Atom nur die Aussage, das es wahr bzw. falsch ist, getroffen werden kann. Wenn ein Antwortmengensuchprogramm schließen kann, dass eine Aussage nicht getroffen werden kann, gibt es auch die entsprechende Kante im Baum nicht, die Kante wird ausgeschlossen, und damit gibt es auch nicht den von der Kante induzierten Unterbaum.

Es ist leicht erkennbar, dass es für eine geringe Suchbaumgröße besser ist, einen Kante in der Ebene n auszuschließen, als zwei Kanten in der Ebene $n+1$. Daraus folgt, dass einen Kante in der Ebene n auszuschließen besser ist, als 2^i Kanten in der Ebene $n+i$. Ausschließen einer Kante bedeutet dabei, dass die Teilbelegung eines Knotens, zu dem die Kante führen würde, nicht weiter betrachtet wird, weil es für diese Teilbelegung keine Antwortmengen gibt.

Nachfolgend ist $K(n)$ die Anzahl der Kanten in Ebene n und $K = \sum_{n=1}^{A} K(n)$ die Anzahl der Kanten im gesamten Suchbaum.

Im Folgenden geht es um das Wachstum des Berechnungsaufwands und nicht um konkrete Zahlen, die den realen Berechnungsaufwand wiederspiegeln. Deshalb kann als Größe für den Berechnungsaufwand die Anzahl der Kanten im Suchbaum herangezogen werden und damit auch die Anzahl der Knoten, deren Zahl genau

um Eins größer ist, als die der Kanten. Dies ist realistisch, da für jede Aussage Zeit und damit Berechnungsaufwand benötigt wird. Wenn eine untere Abschätzung für einen Algorithmus getätigt werden soll, kann die Anzahl der Kanten des Suchbaums mit der minimalen Zeit oder dem minimalen Berechnungsaufwand, die das Suchprogramm für eine Kante benötigt, multipliziert werden. Im Folgendem wird deshalb die Anzahl der Kanten und die Höhe des Berechnungsaufwands synonym gebraucht. Für obere Abschätzungen des Berechnungsaufwands, die genauer als 2^A sind, ist dieser Ansatz allerdings nur bedingt zu gebrauchen.

Der Gesamtberechnungsaufwand t_{ins} ergibt sich aus dem Mittelwert der Berechnungsaufwände für die Kanten $\bar{t}_{Kanten}$ (beziehungsweise Knoten $\bar{t}_{Knoten}$) multipliziert mit der Anzahl der Kanten ($t_{ins} = K * \bar{t}_{Kanten}$ bzw. $t_{ins} = (K+1) * \bar{t}_{Knoten}$). Wenn der Mittelwert der Berechnungsaufwände für die Kanten also maximal so stark wächst wie die Anzahl der Kanten, ändert er nichts an der Wachstumsart des Gesamtberechnungsaufwands.

Alle Überlegungen dieses Abschnitts sind auch auf die Suchbäume von beliebigen Suchalgorithmen über binäre Variablen (z.B. Nomore++) anzuwenden. Es gibt für die binäre Variablen dann jeweils die Knoten im Suchbaum (bei Nomore++ kommen z.B. zu den Knoten für die Atome noch die Knoten für die Körper, die wie Atome mit Wahrheitswerten belegt werden können) .

4.1.1 Berechnung aller Antwortmengen

Nachfolgend sind einige Modelle für die Suche nach Antwortmengen aufgestellt, die der Reihenfolge entsprechend immer komplexer und leistungsfähiger werden. Hier soll zuerst die Suche von allen Antwortmengen betrachtet werden, da dies einen leichter zu untersuchenden Fall darstellt als die Suche nach einer Antwortmenge.

4.1.1.1 Ausprobieren aller Belegungen

Die einfachste Methode zur Suche nach allen Antwortmengen ist die, bei der alle möglichen Belegungen von Atomen ausprobiert werden. Dabei werden die Belegungen zurückgegeben, welche Antwortmengen darstellen.

```
1  function TryAll(P,<T,F>)
2     if <T,F> covers Atoms(P) then
3        if conflict(P,<T,F>) then
4           return empty set
5        else
6           return {T}
7        end if
8     else
9        x=choose(Atoms(P)\(T U F))
10       return (TryAll(P,<T U {x},F>)
11          U TryAll(P,<T,F U {x}>))
```

```
12    end if.
```

Listing 4.1: Algorithmus zum Ausprobieren aller Belegungen

Listing 4.1 zeigt den Algorithmus dieser Art der Suche, dabei ist `P` das logische Programm für das die Antwortmengen gesucht werden sollen, $I =< T, F >$ ist eine dreiwertige Interpretation, wobei `T` die Menge der wahren Atome und `F` die Menge der falschen Atome ist. Die Funktion `I covers Atoms(P)` ist nur wahr, wenn in der dreiwertigen Interpretation I alle Atome von P belegt sind. Mit `conflict(P,I)` wird geprüft, ob I im Programm P einen Widerspruch erzeugt. Die Funktion `choose(Atoms(P)\(T U F))` wählt aus den Atomen von P ein noch nicht mit einem Wahrheitswert belegtes Atom aus. Die Operation `U` vereinigt zwei Mengen und `empty set` ist die leere Menge $\emptyset$.

Gestartet wird die Suche mit `TryAll(P,I=<` $\emptyset, \emptyset$ `>)`. Mit dem zweiten Teil der äußersten if-Bedingung `return (TryAll(P,<T U {x},F>) U TryAll(P, <T,F U {x}>))`, werden alle möglichen Belegungen generiert, die beiden `TryAll`-Aufrufe erzeugen dabei jeweils eine Kante. Mit dem ersten Teil der if-Bedingung werden die erzeugten Belegungen überprüft.

Es ist $K(n)$ die Anzahl der Kanten in Ebene n und K die Anzahl der Kanten im gesamten Graph, dann ist:

$$K(1) = 2 \tag{4.1}$$

$$K(n) = K(n-1) * 2 = 2^n \tag{4.2}$$

$$K = \sum_{n=1}^{A} 2^n = 2^{A+1} - 2 \tag{4.3}$$

In der Kantenebene A stellen die Kanten das Überprüfen der Belegungen dar, damit sind dort alle Kanten vorhanden.

Das Wachstum der Kanten und damit des Berechnungsaufwands ist in diesem Modell eindeutig exponentiell.

Mit dem folgendem Beispiel sollen die Ausmaße dieses Wachstums verdeutlicht werden. Bei 64 Atomen gibt es $K = 2^{65} - 2$ Kanten, fast doppelt soviel wie 2^{64}. Eine Zahl, die im Hinblick auf 64 bit Pointer und Rechnerarchitektur auftaucht, von der es heißt, dass sie größer ist, als die Zahl der Atome auf der Erde. Der Versuch auf diese Weise alle Antwortmengen eines Programms mit nur 64 Atomen zu suchen, wäre also vergleichbar damit, alle Atome der Erde zu untersuchen.

Moderne Antwortmengensolver sind in vielen Fällen in der Lage alle Antwortmengen von Programmen mit einigen hundert Atomen zu berechnen. Für Programme mit 64 Atomen benötigen sie oft nur Millisekunden, dies demonstriert die Leistungsfähigkeit der verwendeten Algorithmen. Wenn bei einem logisches Programm mit 64 Atomen allerdings alle möglichen Interpretationen Antwortmengen sind, ist die Erzeugung jeder dieser 2^{64} Antwortmengen zu aufwändig, egal welche vollständige Suchmethode verwendet wird.

4.1.1.2 Ausschließen einiger Kanten

Die Einsparungen beim Berechnungsaufwand basieren auf dem Ausschließen von Wahrheitswertbelegungen, wenn diese mit Sicherheit keiner Antwortmenge entsprechen.

```
1  function DelSome(P,<T,F>)
2     if <T,F> covers Atoms(P) then
3        if conflict(P,<T,F>) then
4           return empty set
5        else
6           return {T}
7        end if
8     else
9        x=choose(Atoms(P)\(T U F))
10       if not(conflict(P,<T U {x},F>)) then
11          AW= DelSome(P,<T U {x},F>
12       end if
13       if not(conflict(P,<T,F U {x}>)) then
14          AW= AW U DelSome(P,<T,F U {x}>)
15       end if
16       return AW
17    end if.
```

Listing 4.2: Algorithmus für das Ausschließen einiger Kanten

Listing 4.2 zeigt den Algorithmus dieser Suche. Die Bedeutung der verwendeten Funktionen ist die Gleiche wie im letzten Abschnitt für Listing 4.1. Der Unterschied zu Listing 4.1 ist, dass durch `conflict(P,<T,F>)` schon früher bei der Suche Belegungen ausgeschlossen werden, dadurch werden Kanten gelöscht. Die Menge `AW` ist die Menge der Antwortmengen. Auch hier symbolisieren die Selbstaufrufe durch `DelSome` die Kanten.

$K_d(n)$ ist die Anzahl der Kanten oder der Aussagen über Atome, die in Ebene n ausgeschlossen (deleted) werden können oder wie oft die `conflict` Funktion in dieser Ebene ($|T \cup F| + 1 = n$) wahr ist. Dabei werden nur Kanten gezählt, die in dieser Ebene noch vorkommen. Es werden also keine Kanten gezählt, die in der Ebene nicht existieren, weil auf einer niedrigeren Ebene eine ihrer Vorgängerkanten ausgeschlossen wurde.
Es gilt:

$$0 \leq K_d(n) \leq 2K(n-1) \leq 2^n \quad n = 1 \ldots A \tag{4.4}$$

$$K(1) = 2 - K_d(1) \tag{4.5}$$

$$K(n) = K(n-1) * 2 - K_d(n) \tag{4.6}$$

$$K = \sum_{n=1}^{A} K(n) \tag{4.7}$$

$$
\begin{aligned}
K &= K(1) + K(2) + \ldots + K(A) \\
&= (2 - K_d(1)) + ((2 - K_d(1)) * 2 - K_d(2)) + (((2 - K_d(1)) * 2 - \\
&\quad K_d(2)) * 2 - K_d(3)) + \ldots + (\ldots(((2 - K_d(1)) * 2 - K_d(2)) * 2 \\
&\quad -K_d(3)) * 2 - K_d(4))\ldots) * 2 - K_d(A)) \\
&= (2 - K_d(1)) + (2 - K_d(1)) * 2 - K_d(2) + (2 - K_d(1)) * 4 - \\
&\quad K_d(2) * 2 - K_d(3) + \ldots + (2 - K_d(1)) * 2^{A-1} - \\
&\quad K_d(2) * 2^{A-2} - K_d(3) * 2^{A-3} - K_d(4) * 2^{A-4} - \ldots - K_d(A) \\
&= (2 - K_d(1)) * (1 + 2 + 4 + \ldots + 2^{A-1}) - \\
&\quad K_d(2) * (1 + 2 + 4 + \ldots + 2^{A-2}) - \\
&\quad K_d(3) * (1 + 2 + 4 + \ldots + 2^{A-3}) - \ldots - K_d(A) \\
&= (2 - K_d(1)) * (2^A - 1) - K_d(2) * (2^{A-1} - 1) - \\
&\quad K_d(3) * (2^{A-2} - 1) - \ldots - K_d(A) \\
&= (2^{A+1} - 2) - K_d(1) * (2^A - 1) - K_d(2) * (2^{A-1} - 1) - \\
&\quad K_d(3) * (2^{A-2} - 1) - \ldots - K_d(A) \qquad (4.8)
\end{aligned}
$$

Der erste Term $(2^{A+1}-2)$ aus Formel 4.8 stellt die Anzahl der möglichen Kanten im Suchbaum dar. Von diesem Term werden dann, jeweils für die einzelnen Ebenen, die Anzahl der Kanten der Untersuchbäume abgezogen, die durch das Löschen von Kanten in dieser Ebene eingespart werden, beziehungsweise im Suchbaum entfallen. Dafür sind die Terme $K_d(n) * (2^{A-(n-1)} - 1)$ verantwortlich, wobei $(2^{A-(n-1)} - 1)$ die Anzahl der Kanten eines Untersuchbaums ist, welcher durch Ausschließen einer Kante in Ebene n eingespart wird. Damit ist $(2^{A-(n-1)} - 1)$ sozusagen das Gewicht, das angibt, wie viel das Ausschließen von Kanten in Ebene n bringt. Das Ausschließen von Kanten bringt demnach um so mehr, je früher es geschieht.

Dies soll an folgender vereinfachten Betrachtung verdeutlicht werden. Um sie überschaubar zu halten, wird nur das Ausschließen einer Wahrheitswertbelegung für ein beliebiges Atom a betrachtet. Dieses Atom wird nur im Suchbaum von Ebene n_1 zu n_2 $(n_1 < n_2)$ bewegt und es wird auch davon ausgegangen, dass pro Ebene nur jeweils ein Atom den Knoten in dieser Ebene zugeordnet ist.

Entweder wird eine Kante für ein Atom a in Ebene n_1 ausgeschlossen oder es können $2^{n_2-n_1}$ Kanten in der Ebene n_2 ausgeschlossen werden. Die Belegungen für andere Atome gelten sowohl für eine Kante in Ebene n_1, als auch für $2^{n_2-n_1}$ Kanten in Ebene n_2, denn mit jeder Ebene, die a nach unten rutscht, verdoppeln sich die Kanten für eine Wahrheitswertbelegung des Atoms a mit den gleichen Grundbedingungen. Weil in dieser Betrachtung davon ausgegangen wird, dass Belegungen von anderen Atomen nicht ausgeschlossen werden, können Atome zwischen Ebene n_1 und n_2 keinen Einfluss auf a haben. Wenn sie zum Ausschließen von Kanten von Atom a in Ebene n_2 führen würden, würde a in Ebene n_1 zum Ausschließen von Kanten bei diesen Atomen führen.

$dK_d(n_1, n_2)$ ist die Anzahl von zusätzlichen Kanten im Suchbaum, wenn das Atom a von Ebene n_1 zu Ebene n_2 verschoben wird. Dann ist:

$$\begin{aligned} dK_d(n_1, n_2) &= (2^{A-(n_1-1)} - 1) - 2^{n_2-n_1} * (2^{A-(n_2-1)} - 1) \\ &= (2^{A-(n_1-1)} - 1) - 2^{n_2-n_1} * 2^{A-(n_2-1)} + 2^{n_2-n_1} \\ &= (2^{A-(n_1-1)} - 1) - 2^{A-n_2+1+n_2-n_1} + 2^{n_2-n_1} \\ &= 2^{A-(n_1-1)} - 1 - 2^{A-(n_1-1)} + 2^{n_2-n_1} \\ &= 2^{n_2-n_1} - 1 \end{aligned} \tag{4.9}$$

Daraus folgt, dass die Anzahl der zusätzlichen Kanten exponentiell mit der Anzahl der Ebenen wächst, die das Atom nach unten verschoben wird. Damit wächst die Einsparung von Kanten exponentiell mit der Zahl der Ebenen, mit der eine Aussage früher verworfen wird. Die Reihenfolge, in der Atome von einem Algorithmus behandelt werden, hat also einen starken Einfluss auf den Berechnungsaufwand.

Im Allgemeinen allerdings werden bei einer Suche viele verschiedene Belegungen für viele verschiedene Atome verworfen. Wenn eine Belegung für ein Atom verworfen wird, geschieht dies aufgrund einer Menge (auch $\emptyset$) von Belegungen, die früher getroffen wurden und mit der die Belegung für das Atom bezüglich des Programms unvereinbar ist. Wenn solche Mengen von Belegungen von Atomen (diese stellen dreiwertige Interpretationen dar), die zusammen unvereinbar sind, betrachtet werden, ist es vorteilhaft, die Belegungen von Atomen bei der Suche in einer Weise zu machen, dass möglichst schnell, möglichst viele dieser Mengen in der Menge der gemachten Wahrheitswertbelegungen enthalten sind, damit Kanten möglichst früh und reichlich gelöscht werden können.

4.1.1.3 Reale Suche

In diesem Modell gibt es drei Operatoren, die bei der Suche ausgeführt werden: choice (nichtdeterministische Auswahl), Schließen von Atombelegungen und Finden von Widersprüchen. Die choice-Operation überprüft beide Möglichkeiten der Belegung eines Atoms. Knoten, bei denen ein choice ausgeführt wird, haben zwei ausgehende Kanten. Beim Schließen von Atombelegungen wird aus den bisher getroffenen Aussagen über Atome eine neue Aussage über ein Atom geschlossen, über das es noch keine Aussagen gibt. Im Suchbaum gibt es an dem entsprechenden Knoten nur eine abgehenden Kante. Das Schließen von Atombelegungen wird auch Inferenz genannt. Durch die Operation zum Finden von Widersprüchen kann eine dreiwertige Interpretation verworfen werden. Ein entsprechender Ast im Suchbaum wird nicht weiter expandiert. Diese Operation wird vor der choice-Operation ausgeführt.

Dieses Modell ist nur eine andere Betrachtungsweise der Vorgänge des Modells aus dem vorhergehenden Abschnitt 4.1.1.2, in der anstatt der ausgeschlossenen Kanten die drei Operationen, die unterschiedlich viele Kanten ausschließen, betrachtet werden.

```
1  function RealS(P,<T,F>)
2     I := expand(P,<T,F>)
3     if conflict(P,<T,F>) then
4        return empty set
5     else
6        if <T,F> covers Atoms(P) then
7           return {T}
8        else
9           x := choose(Atoms(P)\(T U F))
10          return (RealS(P,<T U {x},F>))
11             U RealS(P,<T,F U {x}>))
12       end if
13    end if.
```

Listing 4.3: Algorithmus der realen Suche

Listing 4.3 zeigt den Algorithmus dieser Suche. Die Bedeutung der verwendeten Funktionen ist die Gleiche wie im letzten Abschnitt für Listing 4.2. Mit der Funktion `expand(P,<T,F>)` werden Atombelegungen geschlossen. Da keine bis mehrere Atombelegungen auf einmal geschlossen werden können, werden durch die Funktion Pfade im Suchbaum erzeugt. Mit der `conflict(P,<T,F>)` Funktion werden Wiedersprüche gefunden. Die Funktion `choose(Atoms(P)\(T U F))` wählt, wie gehabt, ein noch nicht belegtes Atom aus, sie ähnelt z.B. der `heuristic(P,I)` Funktion von Smodels in Listing 2.1 Seite 12 . Die zwei Selbstaufrufe durch `RealS` symbolisieren die zwei ausgehende Kanten einer choice-Operation.

Mit den drei Operationen dieses Modells, enthält es alle grundlegenden Elemente von Antwortmengensolvern wie Smodels und Nomore++. Die Unterschiede der Systeme liegen unter anderem in der verwendeten Logik, die Art wie die Operationen arbeiten, der Herangehensweise und den verwendeten Heuristiken.

Im Nachfolgendem sei c_n der Anteil der choice und w_n der Anteil der Widersprüche in Ebene n.

$$\begin{aligned}
0 \leq \quad w_n \quad &\leq 1 \quad n = 1 \ldots A \quad (w_n * K(n-1)) \in \mathbb{N} \\
0 \leq \quad c_n \quad &\leq 1 \quad n = 1 \ldots A \quad (c_n * (1 - w_n) * K(n-1)) \in \mathbb{N} \\
K(n) \quad &= \quad (K(n-1) * (1 - w_n)) * (1 + c_n) \\
K(n) \quad &= \quad K(n-1) * (1 - w_n + c_n - w_n c_n) \\
K \quad &= \quad \sum_{n=1}^{A} \prod_{i=1}^{n} (1 - w_n + c_n - w_n c_n)
\end{aligned} \tag{4.10}$$

Nach Abschnitt 4.1.1.2 ist es vorteilhaft, um die Kantenanzahl möglichst gering zu halten, möglichst früh, möglichst viele Widersprüche zu finden und, da wo dies nicht möglich ist, mögliche Kanten zu schließen. So selten und so spät wie möglich sollte die choice-Operation eingesetzt werden.

Dies wird durch die Formel 4.10 bestätigt. Da $(1 - w_n + c_n - w_n c_n)$ Terme mit kleineren Indizes n häufiger auftauchen, ist es, um die Kantenanzahl gering zu halten, vorteilhaft, vor allem $(1 - w_n + c_n - w_n c_n)$ Terme mit kleineren Indizes klein zu halten. Daraus folgt, dass vor allem w_n Terme mit kleineren Indizes größer und c_n Terme mit kleineren Indizes kleiner sein sollten, also möglichst früh Widersprüche gefunden werden sollten und möglichst spät choices gemacht werden sollten.

Diesem Vorgehen stehen die nötigen Informationen für die Operatoren gegenüber. Der choice-Operator kann, bis auf die letzte Ebene, immer eingesetzt werden, ohne dass die Vollständigkeit eingebüßt wird. Zum Ausschließen einer Kante werden meist Informationen bzw. schon gemachte Belegungen benötigt. Damit werden zum Schließen von nur einer Kante im allgemeinen weniger Informationen benötigt, als zum Ausschließen beider Kanten, also zum Schließen eines Widerspruchs.

Dem Suchalgorithmus können mehr Informationen zur Verfügung gestellt werden, dies kann beispielsweise durch Heuristiken oder lookahead (Vorausschauen) geschehen. Die Bereitstellung dieser Informationen kostet allerdings zusätzliche Zeit. Die Zeit, die der Suchalgorithmus insgesamt benötigt, sollte aber möglichst klein sein.

Mit der Formel 4.10 können auch Aussagen über das Verhältnis von choices zur Kantenanzahl gemacht werden. Dafür wird zur Vereinfachung der Anteil der Widersprüche nicht beachtet, also $w_n = 0$ für alle $n = 1 \ldots A$ gesetzt. Ist die Anzahl der Ebenen, in denen es choices gibt, gleich oder kleiner einer Konstanten C, wächst die Zahl der Kanten maximal linear mit der Anzahl der Atome des Programms P.
Da:

$$\begin{aligned} \prod_{i=1}^{n}(1 + c_i) &\leq \prod_{i=1}^{C} 2 = 2^C \\ \Rightarrow K &\leq \sum_{n=1}^{A} 2^C = A * 2^C \end{aligned} \tag{4.11}$$

Jeder choice erzeugt genau einen zusätzlichen Ast. Ohne einen choice ist nur ein Ast vorhanden. Es sind also insgesamt die Anzahl der choices plus 1 Äste vorhanden. Jeder Ast hat maximal die Tiefe A. Daraus folgt, dass die Zahl der Kanten auch maximal linear mit der Anzahl der Atome des Programms P wächst, wenn die Anzahl der choices insgesamt gleich einer Konstanten C_2 ist.
Da:

$$K \leq A * (C_2 + 1) \tag{4.12}$$

Weiterhin ist das Minimum der Kanten, da zu jedem choice mindestens zwei Kanten gehören, begrenzt durch:

$$min(K) = 2C_2 \tag{4.13}$$

Aus 4.12 und 4.13 folgt:

$$2C_2 \leq K \leq A * (C_2 + 1) \tag{4.14}$$

4.1.2 Berechnung einer Antwortmenge

Meistens werden nicht alle Antwortmengen eines Programms gesucht, sondern nur eine (Entscheidungsproblem). Bei der Suche nach allen Antwortmengen geht es darum, möglichst viel von Suchbaum auszuschließen oder wegzuschneiden. Im Gegensatz dazu, geht es bei der Suche nach einer Antwortmenge darum, möglichst wenig vom Baum bei der Suche zu benutzen.

Wenn ein Programm sehr viele Antwortmengen hat, ist das bei der Suche nach allen Antwortmengen für einen niedrigen Berechnungsaufwand schlecht, da alle Antwortmengen gefunden werden sollen und der Suchraum bzw. Suchbaum damit wahrscheinlich größer wird, als wenn es nur wenige Antwortmengen gibt. Bei der Suche nach einer Antwortmenge sind mehr Antwortmengen aber vorteilhaft, da mehr Wege zum Ziel führen und damit die Wahrscheinlichkeit auf Abwegen zu geraten geringer wird.

Die beiden Zielsetzungen unterscheiden sich also signifikant. Bei der Suche nach einer Antwortmenge ist es entscheidend, möglichst wenig Umweg zu machen. Die Größe des Umwegs ergibt sich aus der Anzahl der falschen Entscheidungen, multipliziert mit der durchschnittlichen Anzahl von Kanten der Unterbäume, die nach falschen Entscheidungen exploriert werden. Bei der Suche können maximal A mal wirklich falsche Entscheidungen getroffen werden, in dem Sinne, dass von einem Pfad abgewichen wird, der zu einer Antwortmenge führt. Wenn keine Antwortmenge existiert, führt die Entscheidung eine zu suchen zu keiner Antwortmenge und ist in dem Sinne eine falsche Entscheidung. Diese falschen Entscheidungen sind von den wrong choices (falsche Entscheidungen) die Smodels ausgibt, zu unterscheiden. Da Smodels auch choices zählt, die im Suchbaum einer schon vorher gemachten falschen Entscheidung liegen und an denen man sich daher eigentlich nicht mehr falsch entscheiden kann.

Der Untersuchbaum einer falschen Entscheidung, ist der Untersuchbaum der durchsucht werden muss, bis sicher ist, dass er zu keiner Antwortmenge führt. Er ist also eine Art erfolgloser Suchbaum, bei dem gezeigt werden muss, dass die Menge aller seiner Antwortmengen leer ist. In diesem Sinne sind hier doch noch die Überlegungen nützlich, die bei der Suche nach allen Antwortmengen gemacht wurden. Ein Unterschied ist allerdings, dass bei diesen Untersuchbäumen in der Regel schon Vorwissen vorhanden ist.

```
1  function OneAW(P,<T,F>)
2     I := expand(P,<T,F>)
3     if conflict(P,<T,F>) then
4        return empty set
5     else
6        if <T,F> covers Atoms(P) then
7           return {T}
8        else
9           x := choose(Atoms(P)\(T U F))
10          if chooseValue(P,x) then
```

```
11              I1=<T U {x},F>
12              I2=<T,F U {x}>
13          else
14              I1=<T,F U {x}>
15              I2=<T U {x},F>
16          end if
17          AW=OneAW(P,I1)
18          if (AW==empty set) then
19              return OneAW(P,I2)
20          else
21              return AW
22          end if
23      end if
24  end if.
```

Listing 4.4: Algorithmus für die Suche nach einer Antwortmenge

Das Listing 4.4 zeigt den Algorithmus dieser Suche. Die Bedeutung der verwendeten Funktionen ist wieder die gleiche wie im letzten Abschnitt für Listing 4.3. Mit der Funktion `chooseValue(P,x)` wird für das Atom `x` eine Wahrheitswertbelegung ausgesucht, die als erstes ausprobiert werden soll, `I1` ist dabei die partielle Interpretation, für die zuerst eine Antwortmenge gesucht werden soll, und `I2` die partielle Interpretation, für die danach eine Antwortmenge gesucht werden soll. Mit `(AW==empty set)` wird geprüft, ob `AW` die leere Menge $\emptyset$ ist. Der Algorithmus gibt `empty set` also die leere Menge $\emptyset$ zurück, wenn keine Antwortmenge existiert.

Mit den Funktionen `choose(Atoms(P)\(T U F))` und `chooseValue(P,x)` wird die Entscheidung getroffen, welcher Ast als nächstes durchsucht werden soll, hier können falsche Entscheidungen getroffen werden.

Es ist allerdings bei anderen Verfahren zur Suche einer Antwortmenge auch möglich, dass bei ihren Suchalgorithmus nicht immer die ganzen erfolglosen Untersuchbäume durchsucht werden. Es ist auch eine Suche möglich, bei der mehrere Pfade parallel durchsucht werden und die dann die zuerst gefundene Antwortmenge zurückgibt. Oder auch eine Suche, die aus den aktuell, noch nicht beendeten Pfaden, sich , mithilfe einer Heuristik, den Aussichtsreichsten für die nächste Operation aussucht. Beide Suchen haben den Vorteil, dass erfolglose Untersuchbäume zum Teil nur teilweise durchsucht werden, bevor eine Antwortmenge gefunden wird. Der Nachteil der Suchen ist, dass in der Regel der benötigte Speicherplatz exponentiell mit der Problemgröße wächst und dass dabei wahrscheinlich auch mehrere weitere Untersuchbäume teilweise durchsucht werden, die weitere Antwortmengen enthalten. Da nur bei Untersuchbäumen, die vollständig durchsucht wurden, sicher ist, welche von ihnen Antwortmengen enthalten und welche nicht. Deshalb werden derartige Verfahren hier nicht näher betrachtet.

Dass nur bei vollständig durchsuchten Untersuchbäumen sicher ist, ob sie eine Antwortmenge enthalten, führt auch dazu, dass bei Programmen die keine Ant-

wortmengen haben, unabhängig von der Art der Suche, der ganze Suchbaum durchsucht werden muss. Dann kann das Modell des letzten Abschnittes 4.1.1.3 angewendet werden.

Wenn erfolglose Untersuchbäume vollständig durchsucht werden, ist $K_{wc}(n)$ die Anzahl der Kanten des erfolglosen Untersuchbaumes, der, bei der Suche nach einer Antwortmenge, durch die falsche Entscheidung in Ebene n durchsucht werden muss. Da nur eine Antwortmenge gesucht wird, kann in jeder Ebene maximal eine falsche Entscheidung auftreten. Wenn es keine falsche Entscheidung in Ebene n gibt, ist $K_{wc}(n) = 0$.

Somit ist die Anzahl der Kanten K^1 im Suchbaum bei der Suche nur einer Antwortmenge, so fern es eine gibt:

$$K^1 = \sum_{n=1}^{A}(1 + K_{wc}(n)) \tag{4.15}$$

$$0 \leq K_{wc}(n) \leq 2^{A-n+1} - 1 \quad n = 1 \ldots A \tag{4.16}$$

Wahrscheinlich nimmt $K_{wc}(n)$ mit wachsenden n exponentiell ab. Dafür gibt es zwei Argumente. Das Erste ist die auferlegte Begrenzung aus Formel 4.16 . Der erfolglose Untersuchbaum kann nur noch die Kanten eines Baumes der Tiefe von $a-n$ haben, plus die eine Kante für die falsche Entscheidung. Das zweite Argument ist, dass in Ebene n schon $n - 1$ Wahrheitswertbelegungen der Ebenen unter ihr als Vorwissen vorhanden sind. Wie schon in im vorhergehenden Abschnitt 4.1.1.3 beschrieben, begünstigt Vorwissen das Beschneiden des Untersuchbaums.

Weiterhin nimmt mit der Zunahme von Vorwissen, beziehungsweise n, auch die Wahrscheinlichkeit falsche Entscheidungen zu machen ab, da Vorwissen genutzt werden kann, sie zu vermeiden.

4.1.2.1 Berechnung mehrerer Antwortmengen

Nachdem eine Antwortmenge gefunden wurde, können auch weitere Antwortmengen gesucht werden. Dazu können teilweise die Informationen, die beim Finden der schon gefundenen Antwortmengen gewonnen wurden, weiter verwendet werden. Das heißt unter anderem, dass schon durchsuchte Teilbäume nicht noch einmal durchsucht werden müssen. Es muss auch nicht mehr bei der Wurzel des Suchbaums begonnen werden, sondern es können, vom Blatt der letzten gefundenen Antwortmenge beginnend, die nicht durchsuchten Teilbäume zu den auf dem Pfad zur Wurzel liegenden Entscheidungen benutzt werden. Dadurch ist die Anzahl der besuchten Kanten des Suchbaums, die zum Finden einer weiteren Antwortmenge besucht wurden, wahrscheinlich geringer, als die zum Finden der ersten Antwortmenge.

Nachdem alle Antwortmengen gefunden wurden, werden bei der Suche nach einer weiteren Antwortmenge alle noch verbleibenden, nicht durchsuchten Teilbäume, durchsucht und ausgegeben, dass keine weitere Antwortmenge existiert.

Unter der Annahme, dass nur schlecht voraus gesehen werden kann, ob eine Entscheidung falsch oder richtig ist, dürften die noch verbleibenden, nicht durchsuchten Teilbäume in ihrer Kantenanzahl in der Größenordnung des Suchbaums für die erste Antwortmenge liegen.

4.2 Kurvenverhalten

Das Nachfolgende wurde schon größtenteils in Kapitel 3 angesprochen, es soll aber hier noch einmal zusammengefasst und weiter vertieft werden. Es geht dabei um einen Erklärungsversuch der Erfüllbarkeits- und Berechnungsaufwandskurvenverläufe bei SAT und ASP für zufällig generierte Formeln bzw. Programme mit fester oder gemischte Klausel bzw. Körperlänge.

4.2.1 Kurvenverhalten SAT

Die Klauseln bei SAT entsprechen den Regeln bei ASP und die Variablen bei SAT den Atomen bei ASP.

Bei kleinen m/n (entspricht dem hier verwendeten $R/A = Regeln/Atome$) Faktor gibt es viele Modelle, da wenige Klauseln pro Variable vorhanden sind und damit auch weniger unvereinbare Wertekombinationen. Wenn es viele Modelle gibt, führen viele Pfade im Suchbaum zum Ziel (einem Modell), daher ist die Anzahl der falschen Entscheidungen, die getroffen werden können, geringer. Durch die wenigen Klauseln pro Variable können allerdings auch schlechter weitere Wertebelegungen geschlossen (inferiert) werden, da es unwahrscheinlicher wird anwendbare Klauseln zu finden. Dadurch müssen mehr Entscheidungen gefällt werden, bei den meisten Entscheidungen ist es aber nicht wichtig wie sich entschieden wird, da beide Wahlmöglichkeiten zu Modellen führen.

Bei großen m/n Faktor gibt es wenige Modelle, da viele Klauseln pro Variable vorhanden sind und damit auch viele unvereinbare Wertekombinationen. Die möglichen Wertebelegungen der aussagenlogischen Formel werden durch die vielen Klauseln stark begrenzt. Viele oder alle Pfade im Suchbaum führen dann zu keinem Modell, wenn aber falsche Entscheidungen gemacht wurden, kann dies viel schneller erkannt werden, da es viel mehr Klauseln gibt, die bestimmte Teilbelegungen von Variablen ausschließen. Die vielen Klauseln pro Variable vereinfachen auch die Inferenz von Wertebelegungen, da es viel wahrscheinlicher ist anwendbare Klauseln zu finden. Dadurch wird die Anzahl der nötigen Entscheidungen und damit der Berechnungsaufwand weiter reduziert.

Zwischen den beiden Bereichen, einem bestimmten „mittleren“ m/n Faktor, beim Phasenübergang der Erfüllbarkeit, liegt die Anzahl der Modelle, der Entscheidungen, die Inferenzmöglichkeiten und die Tiefe der erfolglosen Untersuchbäume zwischen denen von kleinen und großen m/n Faktoren. Dadurch wird der durchsuchte Untersuchbaum und damit auch der Berechnungsaufwand viel größer.

So entsteht das beobachtete leicht-schwer-leicht Muster.

4.2.2 Kurvenverhalten ASP

Die Unterschied zwischen den hier untersuchten logische Programmen und aussagenlogische Formeln der SAT-Untersuchungen, die in Abschnitt 3.2 Seite 17 beschrieben sind, ist einerseits, die andere Bedeutung des „←“ Symbols, und andererseits, dass es bei normalen logischen Programmen mindestens ein positives Literal (das Kopfliteral) in jeder Regel gibt.

Die andere Bedeutung des „←“ Symbols bewirkt, wie in Abschnitt 2.3.9 auf Seite 9 beschrieben, dass ein Teil der Modelle keine Antwortmengen sind. Eine Folge davon ist, dass nur Atome, die auch im Kopf einer Regel vorkommen, auch in einer Antwortmenge vorkommen können.

Dadurch das positives Kopfliteral können keine negativen Literale als Fakten vorkommen und ein Atom, das als Fakt vorkommt, kann nicht widerlegt werden. Wenn ein Atom als Fakt vorkommt, kommt es auch in allen Antwortmengen vor und es gibt keine Möglichkeit, dass es wegen eines Zirkelschlusses nicht wahr ist.

Diese Unterschiede werden durch die Operationen zum Schließen von Atombelegungen berücksichtigt, die bei den untersuchten ASP-Solvern polynomiell realisierbar sind. Unter der Annahme, dass sich dadurch die Anzahl der choices und damit des Berechnungsaufwands nur im geringeren Maße verändert, dürfte der Berechnungsaufwand bei ASP sich ähnlich wie bei äquivalenten Problemen bei SAT verhalten, der Anteil der erfüllbaren Probleme aber anders sein. Wenn also der Berechnungsaufwand bei 3-SAT ein bestimmtes Verhalten zeigt, dürfte dieses Verhalten auch beim 2-LP Generierungsmodell (konstante Körperlänge mit 2 Literalen pro Körper) auftauchen, das im Abschnitt 5.1.1 beschrieben wird. Das Verhalten des Anteils der erfüllbaren Probleme kann aber anders sein.

Die andere Bedeutung des „←“ bewirkt auch, dass, anders als bei SAT, die möglichen Wertebelegungen auch bei kleinen R/A (=*Regeln*/*Atome*) Werten stärker begrenzt sind. Denn um so weniger Regeln pro Atom es gibt, um so unwahrscheinlicher wird es, dass ein Atom als Kopf einer (anwendbaren) Regel auftaucht. Dadurch sind die Wahrheitswertbelegungen von Atome, die nicht im Kopf einer Regel auftauchen, auf die Belegung Falsch beschränkt, dies vereinfacht natürlich auch die Regeln in denen solche Atome im Körper vorkommen. Allerdings können nur durch Regeln Widersprüche für Atombelegungen erzeugt werden, also nur durch Regeln können Programme unerfüllbar werden. Da es bei kleinen R/A Werten nur wenige Regeln pro Atom gibt, ist der Anteil der erfüllbaren Programme hoch, aber die erfüllbaren Programme haben weniger Antwortmengen, als im Bereich zwischen kleinen und großen R/A Werten.

Im Bereich von großen R/A Werten beschränken, wie bei SAT, die vielen Regeln pro Atom die möglichen Wahrheitswertbelegungen. Es gibt bei großen R/A Werten im Durchschnitt weniger Antwortmengen, weniger erfüllbare Programme und falsche Entscheidungen werden schneller bemerkt, als im Bereich zwischen kleinen und großen R/A Werten. Bei gemischter Körperlänge und großen R/A Werten, beschränkt eine höhere Faktenanzahl pro Atom die Anzahl der Antwortmengen und vereinfacht die Antwortmengenberechnung, sie erhöht aber den Anteil der

erfüllbaren Probleme.

In den Bereichen von großen und kleinen R/A Werten wird, durch die stärkere Beschränkung der möglichen Wahrheitswertbelegungen, die Inferenz erleichtert (es können wahrscheinlich mehr Atombelegungen geschlossen werden, da anwendbare Regeln wahrscheinlicher sind) und die Zahl der nötigen (oder falschen) Entscheidungen reduziert. Dadurch ist der Berechnungsaufwand dort geringer, als im Bereich zwischen großen und kleinen R/A Werten.

Danach sollten Programme im Bereich von kleinen und großen R/A Werten leichter lösbar sein und weniger Antwortmengen besitzen, als im Bereich dazwischen.

Kapitel 5

Generierungsmodelle der logischen Programme

In diesem Kapitel werden die Modelle zur Generierung der logischen Programme für die Testreihen vorgestellt. Es dient der Zusammenfassung aller Generierungsmodelle, so dass sie schnell nachgeschlagen werden können.

Zu beachten ist, dass bei den nachfolgenden Modellen die Parameter, welche die Art der generierten Programme bestimmen, zur Generierung der Programme dienen. Die entstehenden Programme können durchaus andere Parameter haben, z.B. können nicht alle Atome verwendet sein oder es können Regeln doppelt generiert werden. Modelle bei denen die Parameter zur Generierung immer die gleichen sind, wie die Parameter des entstehenden Programms, sind aufwändiger zu realisieren.

5.1 Zufällige logische Programme

Die zufälligen logischen Programme sind an die Modelle aus dem SAT-Bereich angelehnt und enthalten keine allquantifizierten Variablen, sie sind also schon normale logische Programme.

Bei den nachfolgenden Modellen wurden nicht, wie in [29], das doppelte Vorkommen von Regeln oder Atomen im Körper vermieden.

5.1.1 Feste Körperlänge k-LP (fixed bodylength)

In diesem k-LP Modell werden die generierten logische Programme durch drei Parameter bestimmt, die Anzahl A der Atome, R der Regeln und k der Literale pro Regelkörper. Jedes logisches Programm hat dann die vorgegebene Anzahl von Regeln R und Literalen pro Körper k. Der Kopf einer Regel wird aus der Menge der Atome zufällig ausgewählt. Dabei hat jedes Atom die gleiche Wahrscheinlichkeit gewählt zu werden. Für die einzelnen Literale des Körpers einer Regel wird aus der gleichen Menge von Atomen zufällig ein Atom ausgewählt, auch dabei haben

alle Atome die gleiche Wahrscheinlichkeit gewählt zu werden. Das gewählte Atom wird dann mit der Wahrscheinlichkeit von 0.5 negiert.

Ich bin dabei von der Notation in [29] und k-SAT abgewichen, da ich von der Definition von normalen logischen Programmen aus Abschnitt 2.1.1 auf Seite 3 ausgehe. Das k-LP Modell in dieser Arbeit entspricht also dem $(k + 1)$-LP Modell in [29] und dem $(k + 1)$-SAT Modell.

5.1.2 Gemischte Körperlänge k-pLP (mixed bodylength)

Das hier verwendete k-pLP Modell für gemischte Körperlänge (mixed bodylength) ist an das constant-probability (konstante Wahrscheinlichkeit) Modell von SAT angelehnt. In ihm werden die generierten logische Programme durch drei Parameter bestimmt, die Anzahl A der Atome, R der Regeln und der durchschnittlichen Anzahl k von Literalen pro Regelkörper im Programm. Der Kopf einer Regel wird aus der Menge der Atome zufällig ausgewählt. Dabei hat jedes Atom die gleiche Wahrscheinlichkeit gewählt zu werden. Für die Literale des Körpers einer Regel wird aus der gleichen Menge von Atomen jedes mit der gleichen Wahrscheinlichkeit ausgewählt und dieses dann mit der Wahrscheinlichkeit von 0.5 negiert. Die Wahrscheinlichkeit, dass entweder ein bestimmtes Literal im Körper einer Regel vorkommt, beträgt also $\frac{k}{2A}$.

Der Vorteil dieses Modells besteht darin, dass mit ihm alle syntaktisch möglichen normalen Programme generiert werden können.

Dieses Modell kann auch so betrachtet werden, dass die k-pLP Programme aus k-LP Programmen mit $0 \leq k \leq 2A$ gemischt werden.

5.1.3 Gemischte Körperlänge mit zusammenhängenden Abhängigkeitsgraphen

Um Programme mit gemischter Körperlänge und zusammenhängenden Abhängigkeitsgraphen zu erzeugen wurden zwei Generierungsmodelle aufgestellt. Die Programme in beiden Modellen werden durch drei Parameter bestimmt, der Anzahl A der Atome, R der Regeln und der durchschnittlichen Anzahl k der Literale pro Regelkörper. Das Programm hat einen zusammenhängenden Abhängigkeitsgraphen, wenn beim Abhängigkeitsgraphen des Programms alle gerichteten Kanten durch ungerichtete ersetzt werden.

Für das erste Modell sind die Atome mit $(1 \ldots A)$ und die Regeln mit $(1 \ldots R)$ durchnummeriert. Bei der Generierung der Programme werden die ersten $A - 1$ Regeln des logischen Programms so generiert, dass im Körper der i'ten Regel das Atom $i + 1$ vorkommt (negiert oder unnegiert ist dabei gleich wahrscheinlich) und im Kopf der Regel ein Atom kleiner als $i + 1$. Dadurch sind alle Atome miteinander verbunden und Atome niedrigeren Indexes tauchen etwas häufiger auf. Dann werden die restlichen Regeln mit zufällig gewählten Atomen als Kopf erzeugt und in die Körper aller Regeln solange zufällig gewählte und zufällig negierte Atome

eingefügt, bis die gewünschte durchschnittliche Körperlänge k erreicht ist, wobei die Regel, zum Einfügen des nächsten Körperliterals, zufällig gewählt wird.

In einem zweiten Modell wird dafür gesorgt, dass jedes Atom (außer das Erste) mindestens in einem Regelkopf vorkommt. Die Generierung funktioniert wie im ersten Modell, nur dass im Kopf der i'ten Regel für $i < A$ das Atom $i + 1$ vorkommt und im Körper der Regel mindestens ein Atom kleiner als $i+1$, negiert oder unnegiert.

5.2 Hamiltonsche Zyklen auf Zufallsgraphen

Ein weiteres Generierungsmodell wird für das Finden von Hamiltonsche Zyklen auf Zufallsgraphen angewand. Meine Wahl viel auf dieses Generierungsmodell, da es noch relativ einfach ist und schon früher untersucht wurde. Ein Unterschied zu den zufällig erzeugten Programmen ist, dass den Programmen dieses Generierungsmodells ein komplexeres Problem zugrunde liegt.

Die Programme zur Berechnung von Hamiltonschen Zyklen auf einem gerichteten Graphen bestehen aus zwei Teilen. Einen Teil zur Berechnung der Zyklen (Encoding) und einen Teil der den Graphen kodiert (Datenteil oder Instanz).

Der Teil zur Berechnung der Zyklen wurde aus [29] Seite 113 übernommen. Es handelt sich dabei um ein modifiziertes Programm der Universität Kentucky ASP Benchmark Webseite `http://www.cs.engr.uky.edu/ai/benchmark-suite/hamcyc.sm` . Dabei wurden ein paar Atome umbenannt, um mit Niemelä's Kodierung konsistent zu sein. Das Programm ist in Listing 5.1 zu sehen.

```
1 {hc(X,Y)}:-arc(X,Y).
2 :- 2{hc(X,Y):arc(X,Y)},vertex(Y).
3 :- 2{hc(X,Y):arc(X,Y)},vertex(X).
4 :-vertex(X), not r(X).
5 r(Y):- hc(X,Y),arc(X,Y), initialvtx(X).
6 r(Y):- hc(X,Y),arc(X,Y),r(X), not initialvtx(X).
7 initialvtx(0).
```

Listing 5.1: Hamiltonsche Zyklen auf Zufallsgraphen

Der Teil, welcher den Graphen darstellt, besteht aus Fakten der Form „`vertex(X).`“ für die Knoten und „`arc(X,Y).`“ für die Kanten, wobei `X` und `Y` Namen für Knoten des Graphen sind, die aus dem Bereich der natürlichen Zahlen kommen. Der Knoten 0 muss immer vorhanden sein. Die Kante „`arc(X,Y).`“ geht von Knoten `X` zu Knoten `Y`.

Die Parameter zur Generierung von Graphen sind die Anzahl der Knoten N und die Anzahl der Kanten E. Bei der Generierung werden zuerst N Knoten erzeugt und den Zahlen 0, ..., $(N-1)$ zugeordnet. Dann werden für die Kanten E Paare von diesen Zahlen (den Knoten) gebildet, wobei sowohl die erste Zahl, als auch die zweite Zahl des Paares jeweils gleich verteilt, zufällig aus dem Bereich $0,\ldots,(N-1)$ gewählt wird. Doppelte Paare bzw. Kanten werden ausgeschlossen.

Kapitel 6

Testergebnisse

Leider kann von Ergebnissen, die logische Programme bei einem Antwortmengensolver hervorbringen, nicht auf die Ergebnisse, welche die logischen Programme bei anderen Antwortmengensolver hervorbringen, geschlossen werden. Wenn ein logisches Programm beziehungsweise Problem bei einem Antwortmengensolver doppelt so viele choice points hat, wie ein anderes Programm, muss diese Relation nicht bei einem anderen Antwortmengensolvern erhalten bleiben. Der Grund dafür können kleine Implementationsunterschiede sein. Wenn beispielsweise ein Antwortmengensolver die Atome als choice points nimmt die am häufigsten vorkommen und ein anderer Antwortmengensolver nicht, kann es Unterschiede geben.

All die kleinen Implementationsunterschiede zu berücksichtigen, ist allerdings unmöglich. Deshalb sind die Ergebnisse der Antwortmengensolver für einzelne Programmbeispiele, nicht unbedingt zu allgemeinen Aussagen über die Antwortmengensolver heran ziehbar. Für verlässlichere Aussagen ist es notwendig größere Anzahlen von Programmen für die Solver statistisch auszuwerten. Dies ist ein Gegenstand der nachfolgenden statistischen Untersuchungen.

6.1 Testaufbau

Nachfolgend sind einige allgemeine Bemerkungen über den Aufbau, die Daten und die Auswertung der gemachten Testreihen zu finden.

Bei den einzelnen Testläufen wurde für einen Testpunkt, soweit nicht anders angegeben, 1000 logische Programme generiert und ausgewertet.

Die Testpunkte sind meist großflächig erhoben worden, so dass möglichst alle relevanten Bereiche vorhanden sind und von mir auch schnell eine neue Untertestreihe ausgewertet werden konnte, z.B. Testreihen bei konstanten R/A Faktor. Durch das aufgetretene exponentielle Wachstum, sind allerdings einige Bereiche nur schwer zugänglich, so dass einige Testreihen abgebrochen werden mussten.

Auf der beigelegten DVD sind viel mehr Daten vorhanden, als hier erwähnt werden. Allerdings werde ich die wesentlichen Aspekte abhandeln.

Die Testreihen werden im Nachfolgendem mit Namen gekennzeichnet, die dem

Verzeichnisnamen entsprechen, in dem ihre Daten abgelegt sind. Dieser Name wird als Eigenname verwendet.

Die Testreihen wurden zum Einen auf meinen privaten Rechner, AMD Atlon 1800+ 512 MB Arbeitsspeicher unter Suse Linux 9.0, durchgeführt. Bei diesem Rechner hatte ich den alleinige Zugriff, so dass die Zeitwerte in den Testläufen nicht durch (andere) Benutzer verfälscht wurden. Andere Testreihen wurden unter Benutzung von Rechnern der Universität Potsdam erstellt. Diese Rechner konnten parallel von anderen Benutzern verwendet werden. Auch wurde eine Testreihe teilweise auf verschiedenen Rechnern zusammengestellt. Darum sollte man, bei der Auswertung der Zeit, besondere Vorsicht walten lassen. Andere Parameter, wie choice points oder der Wahrheitswertzuweisungen, sind davon nicht betroffen. Auch auf den Universitätsrechnern wurden die Testreihen unter neueren Linux Version erstellt, allerdings mit verschiedenen Linux Versionen.

Die Werte der Ausgabeparameter wurden bei zwei Testreihen auf meinen privaten Rechner mit BMTool von Wolfgang Faber (seine Hompage: `http://www.kr.tuwien.ac.at/staff/faber/`) erfasst und sind in dessen Ausgabeformat abgespeichert. Bei den restlichen Testläufen wurden die Ausgabewerte durch eigene Parser erfasst und in der BMTool Ausgabeform ausgegeben.

Alle generierten logische Programme wurden zu Beginn von LParse übersetzt, dabei wurden teilweise auch die Programme vereinfacht, so dass die Parameterwerte (Anzahl von Atom und Regeln) für die Generierung der Programme nicht unbedingt die gleichen sind, wie die Parameterwerte die von den Antwortmengensolver ausgegebenen werden.

Smodels und Nomore++ wurden mit ihren Standardparametern ausgeführt, beziehungsweise es wurden ihnen beim Aufruf keine anderen Eingabeparameter übergeben, als eventuell der Dateiname des logischen Programms und 0 falls alle Antwortmengen gesucht wurden. Auch BMTool wurde für den Aufruf von Smodels und Nomore++ keine Parameter übergeben.

Die Generierungsparameter sind implizit im Dateinamen für die Testpunkte gespeichert. Für mehr Informationen sei hier auf die „liesmich.txt“ Datei im Stammverzeichniss der DVD verwiesen.

Ausgabeparameter die bei den Smodels Testreihen erfasst wurden, sind von der Ausgabe von Smodels übernommen.
Zu diesen Parametern gehören:

- die Zeit (time)
- die Anzahl der choice points (Entscheidungspunkte)
- die Anzahl der wrong choices (falsche Entscheidungen)
- die Anzahl der Antwortmengen
- die Anzahl der Wahrheitswertzuweisungen (truth assignments)
- die Anzahl der Atome des von Smodels gelesenen Programms

- die Anzahl der Regeln des von Smodels gelesenen Programms

Die Anzahl der choice points gibt an, wie oft Schritt 2 aus Abschnitt 2.4.1 Seite 11 bei der Suche ausgeführt wird, oder wie oft der Smodels-Algorithmus (Listing 2.1 Seite 12) den Teil mit den Selbstaufrufen anläuft. Die wrong choices sind die choice points bei denen beide Wahrheitswertbelegungen für ein Atom ausprobiert werden. Die Anzahl der wrong choices gibt also an, wie oft beim Smodels-Algorithmus beide rekursive Selbstaufrufe ausgeführt werden.
Bei der Nomore++ Ausgabe wurden die folgenden Parameter gespeichert:

- die Zeit (time)
- die Anzahl der choice points (Entscheidungspunkte)
- die Anzahl der choices
- die Anzahl der Antwortmengen

Die choice points von Nomore++ sind analog zu den choice points bei Smodels. Die Anzahl choices ist die Anzahl der Wahrheitswertbelegungen die Nomore++ an choice points ausprobiert hat oder die Anzahl der rekursiven Aufrufe von Nomore++.

Bei der Anzahl der Antwortmengen wurde von BMTool, bei den Testreihen cases und casesN, nur erfasst, ob es Antwortmengen gibt, aber nicht wie viele. Leider wurden die Werte der Atom- und Regelanzahl nicht von Anfang an erhoben, so dass sie teilweise bei den Testreihen cases und casesN fehlen.

Die nachfolgenden Diagramme stellen nur eine kleine Auswahl aller generierten Diagramme dar. Sie sollen charakteristische Sachverhalte verdeutlichen, auf der beigelegten DVD sind weitere Diagramme zu den Testreihen zu finden. Für weitere Erläuterungen der Daten auf der DVD sei auf die „liesmich.txt“ und „info.txt“ Dateien verwiesen.

Alle getroffenen Aussagen betreffen natürlich, wenn nicht anders gekennzeichnet, immer nur die untersuchten Bereiche, da Aussagen außerhalb dieser nicht ohne weiteres möglich sind. Ein Grund dafür ist exponentielles Wachstum. Dieses kann im untersuchtem Bereich noch so schwach sein, dass es nicht bemerkt wird, so dass nur eine polynomielle Komponente des Wachstum bemerkt wird. Wenn über das Wachstum von Werten einer Testreihe gesprochen wird, wird das plausibelste Wachstumsverhalten zugrunde gelegt, da mit endlich vielen Testpunkten das zugrunde liegende Wachstumsverhalten nie mit Sicherheit bestimmt werden kann. Das Wachstumsverhalten einer Testreihe mit n Testpunkten kann beispielsweise immer mit einem Polynom n-ten Grades approximiert werden, trotzdem kann eine exponentielle Funktion zu bevorzugen sein.

Im nachfolgenden steht L für die durchschnittlichen Anzahl der Körperliteralen, A für die Anzahl von Atomen und R für die Anzahl der Regeln in den Programmen. Wenn nicht anders gekennzeichnet sind mit diesen Parametern die Generierungsparameter der Programme gemeint. Mithilfe einer, auf eine Problemklasse

angepassten, Skalierungsformel (z.B. R/A bei 2-LP) kann ein Parameter ermittelt werden, bezüglich dem das Wachstumsverhalten der Ausgabewerte größtenteils unabhängig von untersuchtem Bereich wird. Wenn über das Verhalten einer Kurve gesprochen wird, ist ihr Wachstumsverhalten gemeint.

Höhenlinien begrenzen Bereiche in denen die dargestellten Werte unterschiedliche Höhen haben. Sie sind bei vielen dreidimensionalen Diagrammen unten am Boden des Diagramms zu finden und helfen bei Aussagen über den Höhenverlauf der dargestellten Kurvenfläche (zweidimensionale Kurve).

6.2 Feste Körperlänge k-LP

6.2.1 Berechnung einer Antwortmenge bei 2-LP

Bei der Testreihe cases1 wurde, für die Generierung der logischen Programme, das k-LP Modell für feste Körperlänge (fixed bodylength) aus 5.1.1 auf Seite 40 verwendet. Für die so generierten logische Programme wurde eine Antwortmengen mit Smodels Version 2.27 berechnet. Die Testreihe cases1 wurde auf meinen Heimrechner erhoben.

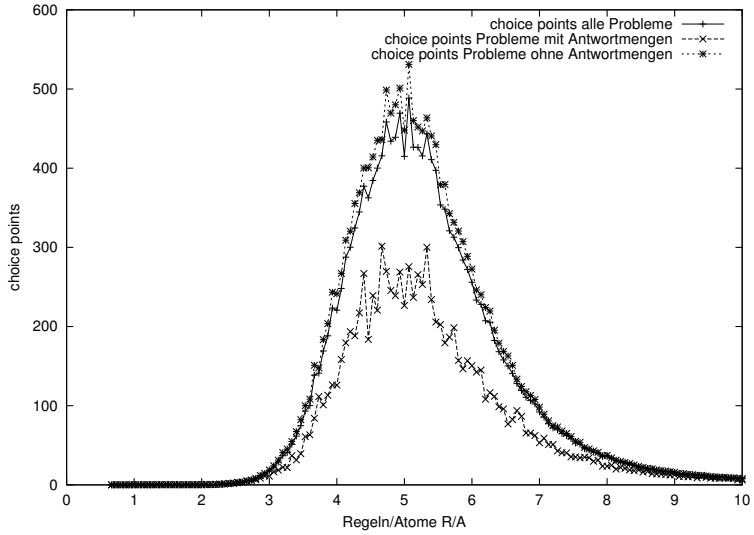

Abbildung 6.1: Durchschnittliche choice points der erfüllbaren, unerfüllbaren und aller Probleme bei 2-LP für 150 Atome

Die Abbildung 6.1 zeigt die durchschnittliche Anzahl der choice points für erfüllbare, unerfüllbare und alle Probleme bei 2-LP mit 150 Atomen. Die Ähnlichkeit zur Abbildung 3.6 auf Seite 22 aus [29] Seite 82 ist auffallend. Ich kann also Yuting Zhao's Ergebnisse für k-LP trotz leicht unterschiedlicher Modelle bestätigen. Anscheinend sind die Effekte von doppelten Atomen im Körper und doppelten Regeln

vernachlässigbar, zumindest im untersuchten Bereich. Die Gründe für den Kurvenverlauf wurden schon in Abschnitt 4.2.2 auf Seite 38 erläutert und werden hiermit bestätigt.

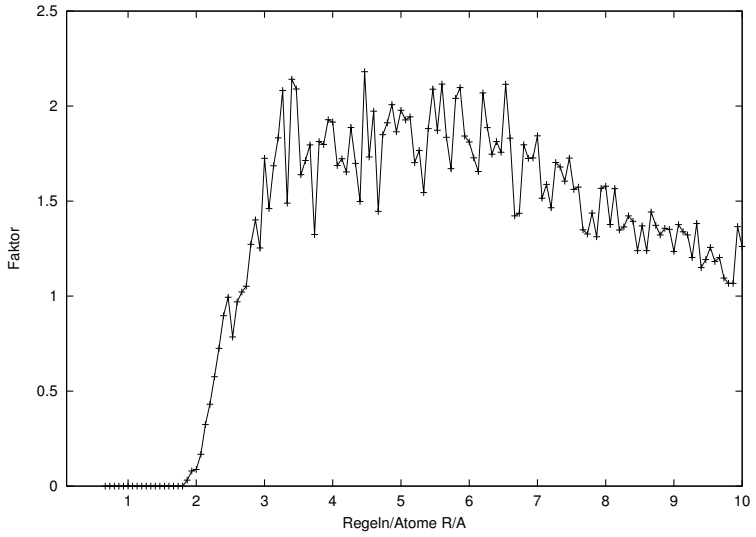

Abbildung 6.2: Quotient von choice points der unerfüllbaren und erfüllbaren Probleme bei 2-LP und 150 Atomen

Wie in Abbildung 6.2 zu sehen, liegt der Quotient von choice point für unerfüllbare Probleme, zu choice point für erfüllbare Probleme ((unerfüllbaren Probleme) /(erfüllbaren Probleme)) rund bei 1.5 . In der Anfangsphase allerdings, bis $R/A = 2$, liegt die durchschnittliche Anzahl der choice points für unerfüllbare Probleme bei 0 . Dadurch liegt dort auch der Quotient bei 0 und nähert sich dann dem Wert 1.5 an, ab rund $R/A = 7$ nimmt er wieder leicht ab. Selbst bei wenigen Regeln sind unerfüllbare Probleme vorhanden (bei 100 Regeln 331 unerfüllbare Probleme) und ihre Anzahl steigt schnell an. Bei 270 Regeln ($R/A = 1.8$) sind 729 unerfüllbare Probleme (= 72.9% aller Probleme) vorhanden, die durchschnittliche Anzahl der choice points der unerfüllbare Probleme liegt aber immernoch bei 0 . Die Anzahl der unerfüllbaren Probleme ist nicht für den niedrigen Wert der durchschnittlichen choice points verantwortlich, sondern unerfüllbare Probleme sind bei wenigen Regeln pro Atom trivial zu lösen.

In dem Bereich, in dem bei unerfüllbaren Problemen keine choice points benötigt werden, gibt es bei erfüllbaren Problemen durchaus choice points (zwischen durchschnittlich 0.08 bei 100 Regeln und 0.41 bei 270 Regeln), allerdings keine wrong choices. Dabei ist zu beachten, dass bei Programmen mit mehr als einer Antwortmenge auch choice points benötigt werden, denn die zwei möglichen Wertebelegungen an den choice point führen dann zu unterschiedlichen Antwortmengen. Die choice points ermöglichen also mehrere Antwortmenge, indem sie als Abzweigung zu ihnen dienen. Ab dem Bereich (ab 270 Regeln) in dem unerfüll-

bare Probleme choice points haben, haben erfüllbare Probleme wrong choices. Im Bereich mit weiniger als 270 Regeln sind also alle Probleme trivial lösbar.

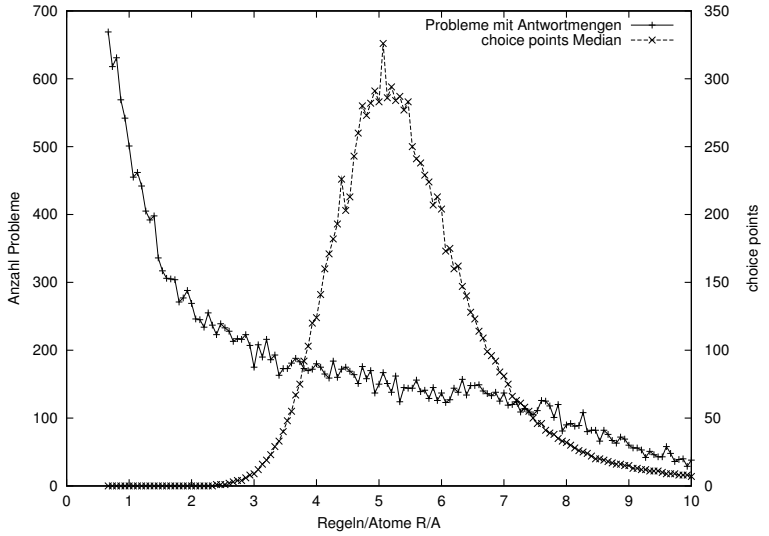

Abbildung 6.3: Median der Anzahl der choice points und Anzahl der erfüllbaren Probleme bei 2-LP für 150 Atome

Abbildung 6.3 zeigt den Median (den Wert in der Mitte, wenn die choice points geordnet sind) der Anzahl der choices points (Skala rechts) und die Anzahl der erfüllbaren Probleme (Skala links) bei 2-LP für 150 Atome. Das Aussehen der Median Kurve von Abbildung 6.3 ähnelt stark dem von Abbildung 6.1, nur dass die konkreten Werte anscheinend kleiner sind, als die des Durchschnitts für alle Probleme. Das lässt darauf schließen, dass sich Durchschnitt und Median ähnlich verhalten.

Bei der Erfüllbarkeitskurve ist zu erkennen, dass es keine Phase gibt, in der die meisten logischen Programme erfüllbar sind. Dass die Anzahl der erfüllbaren Probleme mit der Zahl der Regeln von Anfang an sinkt, kann dadurch erklärt werden, dass mit der Zunahme der Regeln, auch die Zahl der ungeraden Zyklen zunimmt. Die Anzahl der geraden Zyklen nimmt natürlich auch zu, nur steigt die Erfüllbarkeitsquote nicht dadurch, dass mehr Antwortmengen möglich sind. Mit Zunahme der Regelanzahl können nur mehr Bedingungen (und damit auch potentielle Wiedersprüche) hinzukommen.

Abbildung 6.4 zeigt das Maximum der Anzahl der choice points für die Testpunkte bei 2-LP für 150 Atome. Auch sie ähnelt den beiden vorangegangenen Abbildungen 6.1 und 6.3. Allerdings ist sie wesentlich stärker zerklüftet, was daher rührt, dass hier nur ein logisches Programm für die Punkte herangezogen wurde. Weiterhin liegen die Werte deutlich über denen des Durchschnitts und des Medians. Für die maximale Anzahl von choice points gibt es eine obere Grenze bei Smodels, die bei $(2^A - 1)$ liegt (Anzahl der Knoten im maximal aufgespannten

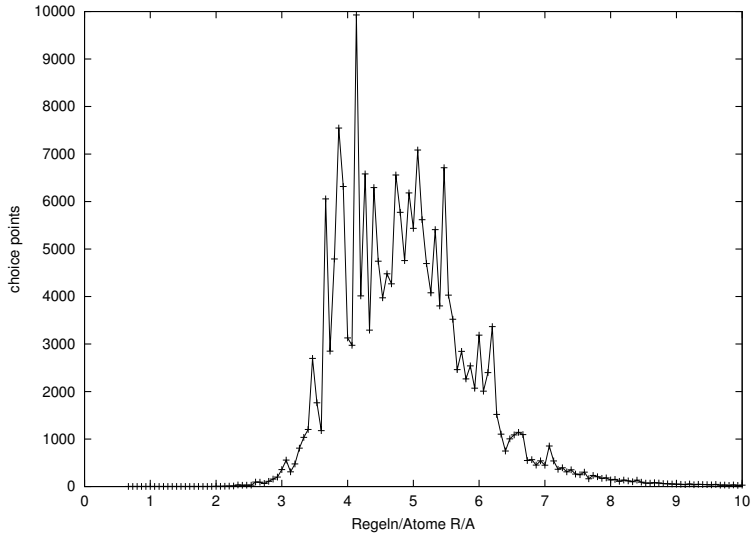

Abbildung 6.4: Maximale Anzahl der choice points bei 2-LP für 150 Atome

Suchbaum), im Durchschnitt oder Median braucht man wesentlich weniger, das Maximum kann sie im schlimmsten Fall erreichen. Die dargestellten Maximalwerte der choice points liegen allerdings noch weit unter der oberen Grenze (hier $(2^{150} - 1)$).

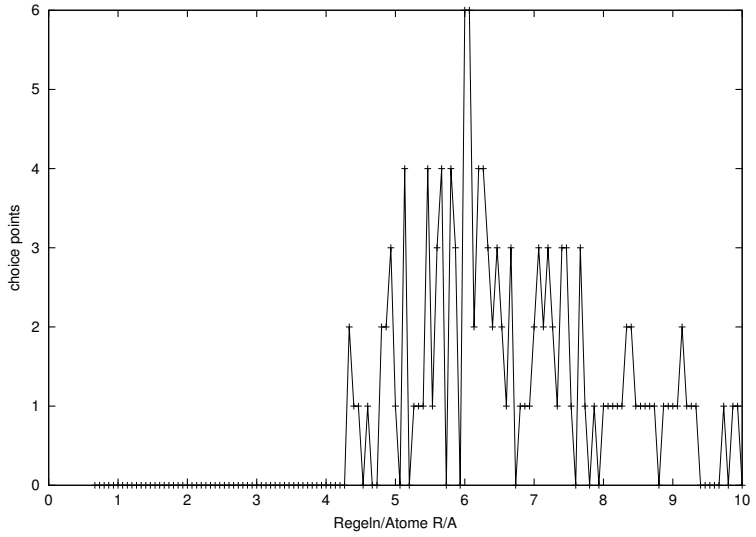

Abbildung 6.5: Minimale Anzahl der choice points bei 2-LP für 150 Atome

Abbildung 6.5 zeigt das Minimum der choice points in den jeweiligen Testpunkten bei 2-LP für 150 Atome. Die Werte, die erreicht werden, sind im Verhält-

nis zu den anderen drei Abbildungen 6.1, 6.3 und 6.4 sehr gering. Es kann davon ausgegangen werden, dass, wenn die Anzahl der generierten Testbeispiele pro Testpunkt erhöht wird, sich die Anzahl der minimalen choice points irgendwann überall bei 0 befindet. Das heißt, die leichten Probleme, die mit wenigen choice points zu lösen sind, nehmen vielleicht mit steigendem Durchschnitt der choice points ab, aber wenn genug Probleme generiert werden, werden immer welche zu finden sein. Der Grund ist, dass mit dem *k*-LP Modell mit allen Generierungsparametern immer trivial lösbare Probleme generierbar sind, z.B. Probleme bei denen alle Atome, die als Kopfatom in einer Regel vorkommen, in keinem Regelkörper vorkommen.

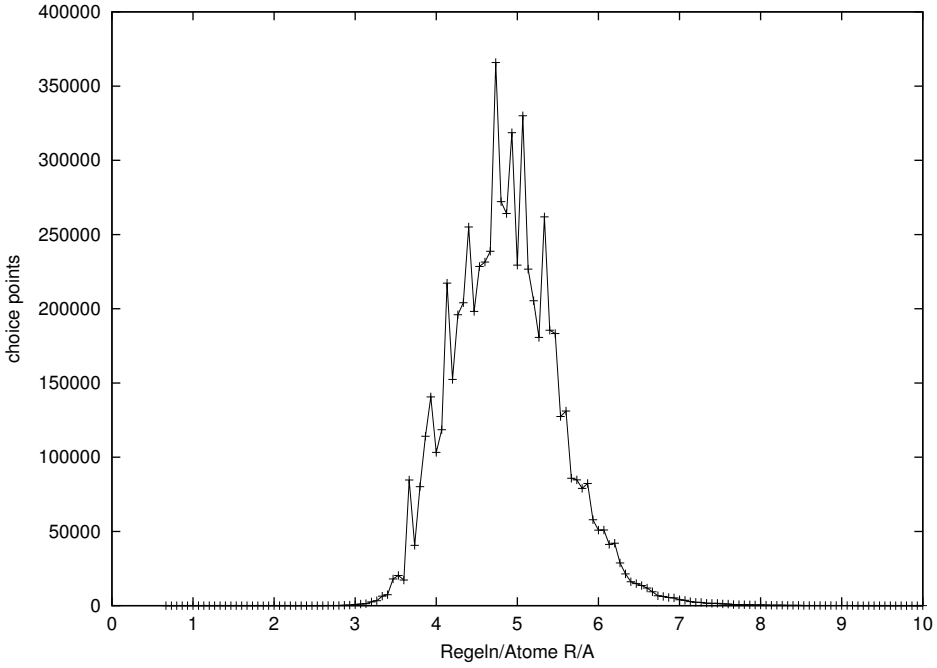

Abbildung 6.6: Varianz in der Anzahl der choice points bei 2-LP für 150 Atome

Abbildung 6.6 zeigt die Varianz der choice points bei 2-LP für 150 Atome. Auch hier ist das gleiche Muster wie bei den drei Abbildungen 6.1, 6.3 und 6.4 zu finden.

Mit diesen Abbildungen für die choice points, kann auf die innere Verteilung der choice point Anzahlen eines Testpunktes (wie viele Programme eines Testpunktes wie viele choice points haben) im schweren Bereich geschlossen werden. Diese Verteilung beginnt bei 0 (wegen Minimum). Sie ist asymmetrisch (wegen dem Unterschied von Median und Durchschnitt), wobei mehr Probleme im Bereichen mit wenigen choice points liegen und die wenigen Probleme mit mehr choice points den Durchschnitt über den Median heben. Im Bereich von sehr vielen choice

points liegen demnach nur wenige Probleme.

Wenn das Muster der Kurven 6.1 für den Durchschnitt als Muster für die Schwere von Problemklassen angenommen wird, nimmt mit der Schwere der Problemklassen das Maximum der Verteilung im Testpunkt stark zu.

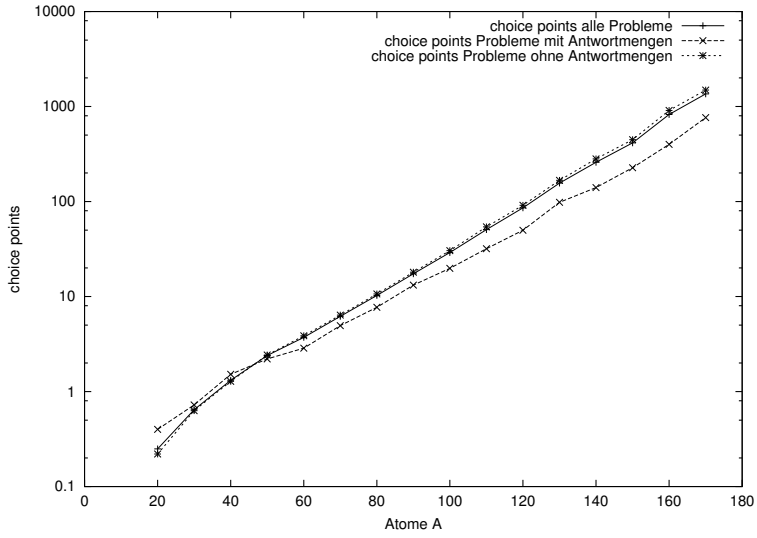

Abbildung 6.7: Anzahl der choice points der erfüllbaren, unerfüllbaren und aller Probleme bei 2-LP für $R/A = 5$

Die Abbildung 6.7 zeigt die durchschnittliche Anzahl der choice points von erfüllbaren, unerfüllbaren und allen Problemen für 2-LP bei dem Faktor $R/A = 5$. Die drei Kurven zeigen, abgesehen von den konkreten Anstiegswerten, ein ähnliches Verhalten.

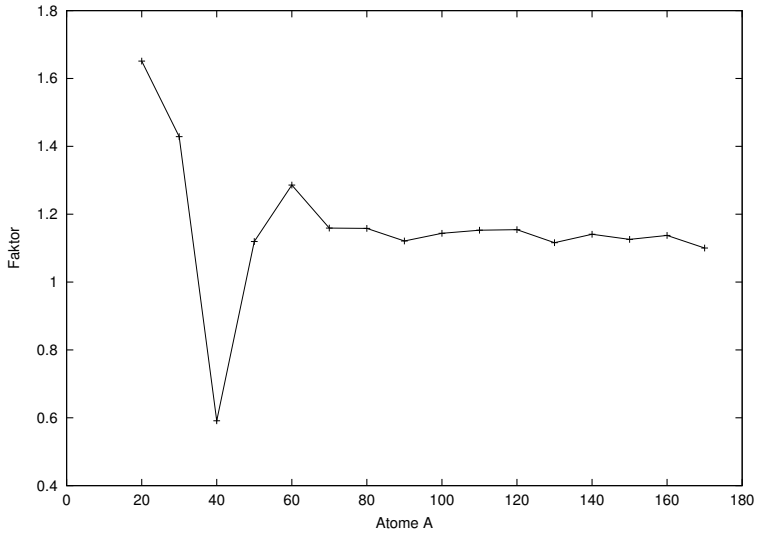

Abbildung 6.8: Quotient von log choice points der unerfüllbaren und erfüllbaren Probleme bei 2-LP und $R/A = 5$

Ein ähnliches Verhalten der Werte wird durch die Abbildung 6.8 bestätigt. Hier

wurde für die einzelnen Testpunkte der Logarithmus der Anzahl der durchschnittlichen choice points bei unerfüllbaren Problemen durch den Logarithmus bei erfüllbaren Problemen geteilt (log(unerfüllbare Probleme)/log(erfüllbare Probleme)). Diese Kurve ist sozusagen der Quotient der Exponentenfunktionen der choice point Wachstumsfunktionen für unerfüllbare und erfüllbare Probleme.

Nach anfänglichen stärkeren Schwankungen bei unter 60 Atomen, pendelt sich die Kurve ziemlich stabil bei einem Wert von rund 1.1 ein. Das heißt, das Wachstum der Kurven unterscheidet sich nur durch einen konstanten Faktor im Exponenten (a in 2^{R*a+b}).

Das Wachstumsverhalten ist also weitestgehend unabhängig davon, ob die Probleme erfüllbar sind oder nicht.

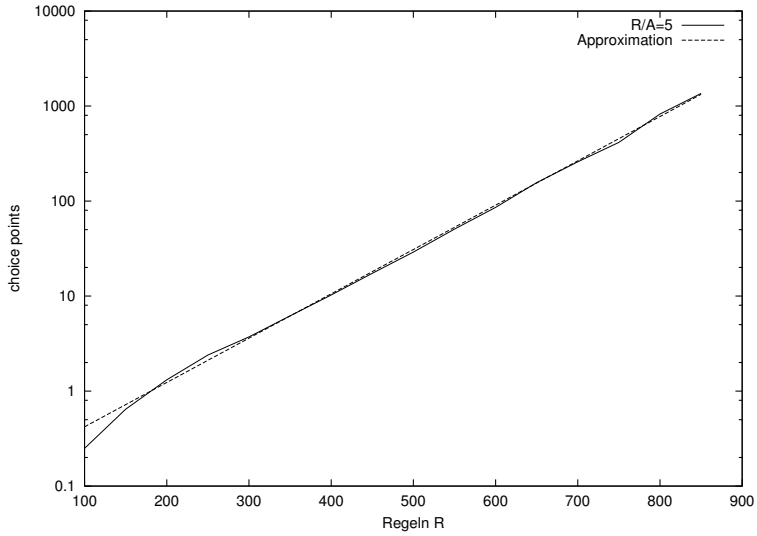

Abbildung 6.9: Durchschnittliche choice points und Approximation bei 2-LP und $R/A = 5$

Für die choice point Kurve für alle Probleme aus Abbildung 6.7 wurde eine exponentielle Funktion, die sich ihr annähern, approximiert. Bereiche unterhalb von einem choice point wurden dabei ignoriert. Die Funktion hat die Form: $f_k^1 = 2^{R*a+b}$, wobei a und b vom R/A Faktor und der Anzahl von Körperliteralen abhängige Konstanten sind und R die Anzahl der Regeln ist. Demnach ist $1/a$ die Anzahl der Regeln, um die die Programme vergrößert werden müssen, bei konstanten R/A Faktor, um die durchschnittliche Anzahl choice points zu verdoppeln.

Wegen $f_k^1(F,R) = 2^{R*a+b}$ und $R = F * A$ ist $g_k^1(F,A) = g_k^1(F,R/F) = 2^{A*F*a+b}$ für die gleichen a und b, wobei F für die entsprechenden R/A Faktoren steht. Wenn also ein Programm bei konstanten R/A Faktor um $\frac{1}{R/A*a}$ Atome vergrößert wird, verdoppelt sich im Durchschnitt der Berechnungsaufwand.

Abbildung 6.9 zeigt die durchschnittlichen choice points und die Approximation. Die Funktion für die Approximation ist: $f_k^1(5,R) = 2^{R*0.0155-2.8}$. Daraus folgt,

dass bei $R/A = 5$ durch zusätzlich 65 Regeln bzw. 13 Atome die durchschnittliche choice point Anzahl rund verdoppelt wird.

Die Originalkurve wächst leicht stärker als exponentiell und beschreibt einen nach oben offenen Bogen, denn zwischen 200 und 300 und ab 800 Regeln ist die Originalkurve über der Approximation zwischen 300 und 800 Regeln aber darunter. Diese Tendenz ist zwar schwach, aber auch bei Kurven für andere R/A Faktoren zu finden, dass dies auf statistischen Ungenauigkeiten beruht ist sehr unwahrscheinlich. Es ist natürlich nicht ausgeschlossen, dass das Wachstum für noch größere Anzahlen von Regeln in ein rein exponentielles Wachstum übergeht. Der Grund für den nach oben offenen Bogen könnte allein an einer polynomiellen Komponente des Wachstums liegen, deren anfänglich starker Anstieg sich mit der Zeit immer mehr abflacht.

Auch bei andere R/A Werte zeigt sich ein ähnliches Wachstumsverhalten.

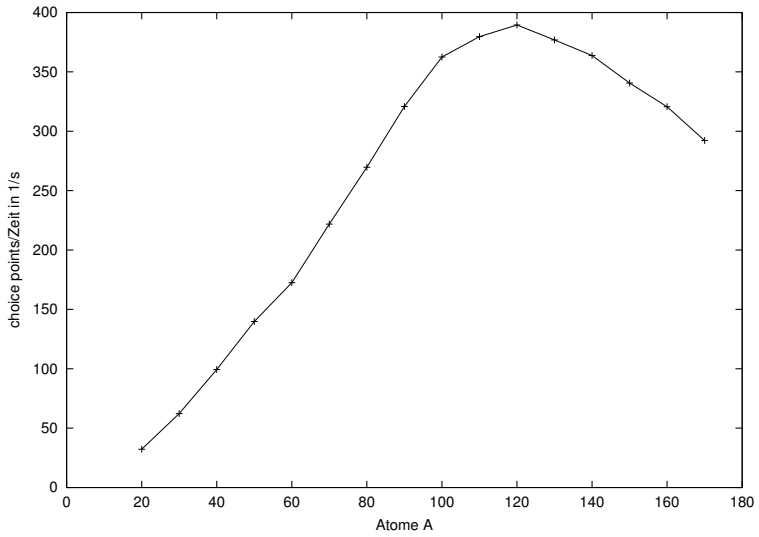

Abbildung 6.10: 2-LP choice points Zeit Quotient für den Faktor $R/A = 5$

Interessant ist auch Abbildung 6.10, sie zeigt den durchschnittlichen choice point (/) Zeit Quotienten für den Faktor $R/A = 5$. Für dieses Diagramm wurden für die einzelnen Probleme die ermittelte Anzahl der choice points durch die ermittelte Zeit dividiert ((choice points)/Zeit). Dargestellt ist jeweils der Durchschnitt für die einzelnen Testpunkte. Bei der Kurve deutlich zu sehen ist ein wenig-viel-wenig Muster. Während in der Anfangsphase durchschnittlich am wenigsten choice points pro Zeiteinheit besucht werden, steigt mit der Atomzahl die Zahl der choice points pro Zeiteinheit auf ein Maximum und sinkt danach wieder ab. Anzunehmen ist, dass sich die Kurve mit der Zeit weiter abflachen wird.

Dieses wenig-viel-wenig Verhalten könnte auf die choice point Vermeidungsstrategien (z.B. Heuristiken und lookahead) und der Zeit zum Einlesen des Programms zurückzuführen sein. Während am Anfang, durch sie die Anzahl der choi-

ce points pro Zeiteinheit stark reduziert wird, versagen diese mit der Zeit. Sie können allerdings nur solange schlechter werden, bis sie so gut wie nutzlos sind. Dann kommt ein zweiter Effekt zu tragen. Bei größeren Programmen gibt es auch mehr Möglichkeiten weiter zu schließen, da mehr Regeln berücksichtigt werden müssen, so dass bei größer werdenden Programmen mehr Zeit zum Schließen zwischen den choice points verbraucht wird.

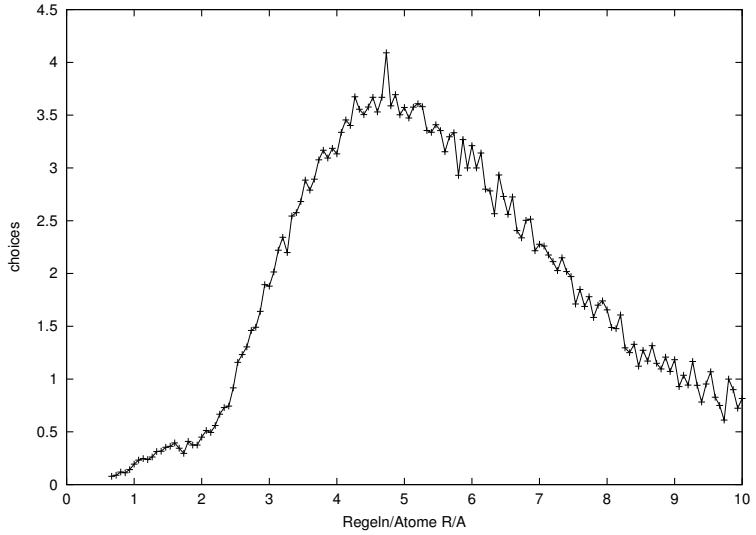

Abbildung 6.11: Differenz choice points und wrong coices für erfüllbare Probleme bei 2-LP und 150 Atomen

In Abbildung 6.11 sind die durchschnittlichen Unterschiede bei erfüllbaren Problemen zwischen choice points und wrong choices für 2-LP bei 150 Atomen aufgeführt. Dafür wurden für die einzelnen erfüllbaren Probleme die ermittelten wrong choices von den choice points abgezogen ((choice points) − (wrong choices)) und für den Testpunkt der Durchschnitt über alle erfüllbare Probleme gebildet. Es gibt zwar ein wenig-viel-wenig Muster, aber im Vergleich zu den tatsächlich gemachten choice points, sind die Werte gering und die Kurve weniger ausgeprägt. Die Anzahl der wrong choices liegt nur etwas unter der Anzahl der choice points, fast alle Entscheidungen führen also zu keiner Lösung bzw. Antwortmenge. Wobei in dem leicht Bereichen der Unterschied noch größer ist, dort werden also proportional weniger wrong choices gemacht. Bedingt durch die kleineren Werte in den untersuchten leicht Bereichen, kann dies aber nicht verallgemeinert werden.

Bei den unerfüllbaren Problemen ist natürlich die Anzahl wrong choices gleich der Anzahl der choice points.

In Abbildung 6.12 ist die Kurve des gleichen Parameters für einen konstanten Faktor $R/A = 5$ dargestellt. Demnach wächst der Unterschied choice points und wrong choices ungefähr linear, wenn von den Anfangswerten bei wenigen Atomen abgesehen wird. Aber auch hier ist der Unterschied nur gering, so dass sich die

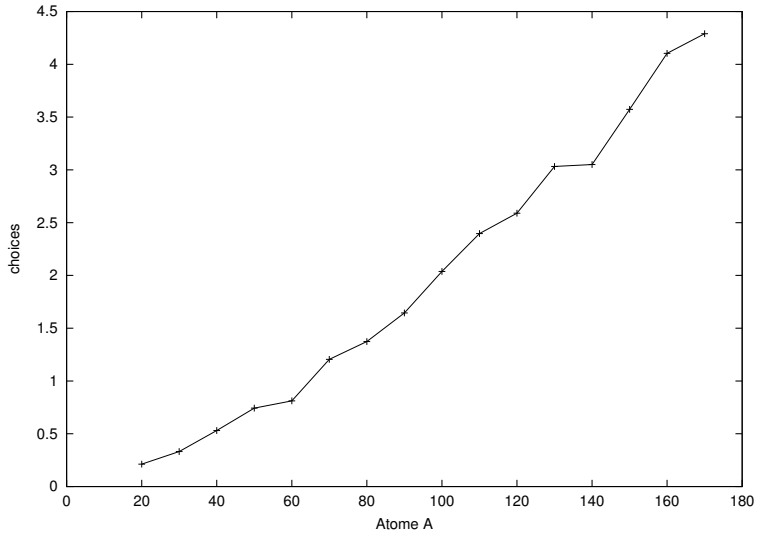

Abbildung 6.12: Differenz choice points und wrong coices für erfüllbare Probleme bei 2-LP und $R/A = 5$

Kurven für choice points und wrong choices sehr ähnlich verhalten.

Das ähnliche Verhalten der Anzahl von choice point und wrong choice wird aus dem Modell für die Suche nach einer Antwortmenge des Abschnittes 4.1.2 Seite 34 ersichtlich. Danach ist max(choice points) = (wrong choices) + A, da maximal A mal richtige Entscheidungen getroffen werden können, also maximal A mal nicht beide Subaufrufe von Smodels gemacht werden. Wenn also entweder die choice point oder wrong choice Anzahl mehr als linear mit der Anzahl der Atome A wächst, muss der jeweils andere irgendwann, wenn A groß genug ist, auch ähnlich stark wachsen.

Da auf dem Pfad von der Wurzel des Suchbaums bis zur Antwortmenge auch Atombelegungen durch Inferenz geschlossen werden, kann nicht gesagt werden, wieviele choice points es auf dem Pfad insgesamt gibt. Damit kann auch nicht gesagt werden, wie hoch der Anteil der richtigen Entscheidungen auf diesem Pfad ist. Wenn aber angenommen wird, dass bei den zufälligen Programmen von k-LP die Wahrscheinlichgkeit sich richtig zu entscheiden bei rund 50% liegt, zeigen Abbildung 6.12 und 6.11 auch ungefähr die durchschnittliche Anzahl der erfolglosen Untersuchbäumen bzw. falschen Entscheidungen bei erfüllbaren Problemen. Bei $R/A = 5$ und 150 Atomen bedeuted dies, dass von den durchschnittlich rund 300 choice points fast alle in den durchschnittlich rund 3.5 falschen Untersuchbäumen auftreten.

6.2.1.1 Untersuchung eines Testreihenpunktes bei cases1

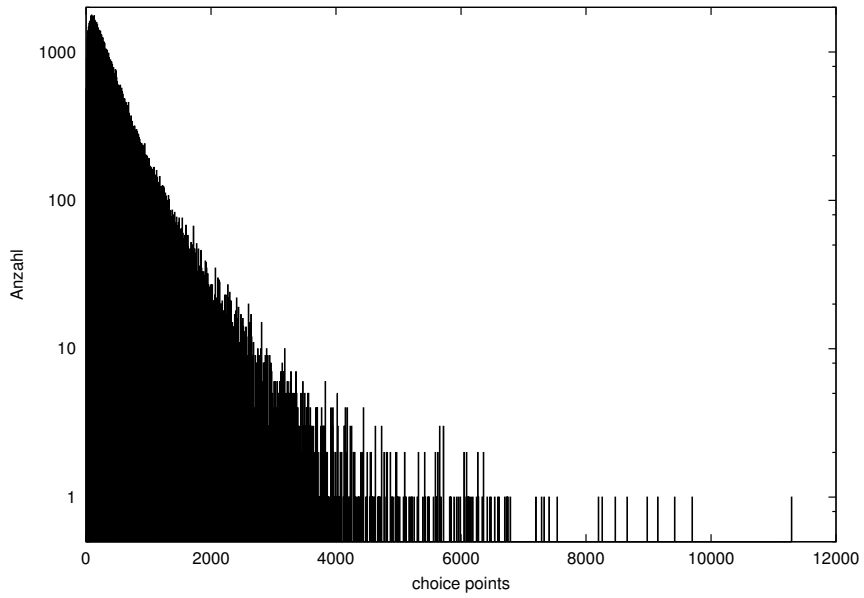

Abbildung 6.13: Anzahl der Programme mit choice points in den Intervallen bei 2-LP für 150 Atome und 750 Regeln

Um die gemachten Behauptungen über die innere Verteilung eines Testpunktes zu bestätigen, wurden für verschiedene Testpunkte mehrere Probleme generiert und die Ergebniswerte mehreren Intervallen zugeordnet. In Abbildung 6.13 ist eine charakteristische Kurve bei 2-LP für 150 Atome, 750 Regeln ($R/A = 5$) und 88828 generierte Probleme zu sehen. Gesucht wurde dabei nur eine Antwortmenge, was der Testreihe cases1 entspricht. Angegeben ist die Anzahl der Probleme, die choice points in den entsprechenden Intervallen hatten. Ein Intervall umfasst dabei choice points in einem Abstand von 10, z.B. die Anzahl der Probleme mit 100 bis 110 choice points. Die y-Achse der Anzahl ist logarithmisch eingeteilt.

Diese Verteilung bestätigt die Behauptung über die Verteilung innerhalb der Testpunkte. Nach einem sehr starken anfänglichen Anstieg der Anzahl, hat die Verteilung ein Maximum bei relativ wenigen choice points erreicht. Danach sinkt die Anzahl von Problemen in den Intervallen und gleitet in hohen choice point Werten aus.

Bei anderen Testpunkten und Berechnungsaufwandsparametern (Zeit, wrong choices usw.) sieht die Verteilung ähnlich aus, ist aber unterschiedlich stark gedehnt, beziehungsweise gestaucht, und mehr oder weniger stark ausgeprägt. Dabei ist die Dehnung der Verteilung im Bereich der größeren Anzahlen von choice points stärker. Mit den Werten für Durchschnitt, Median, Maximum und Minimum können die Veränderungen abgeschätzt werden.

6.2.2 Berechnung aller Antwortmengen bei *k*-LP

Bei der Testreihe cases wurde, für die Generierung der Programme, das *k*-LP Modell für feste Körperlänge (fixed bodylength) aus 5.1.1 auf Seite 40 verwendet. Für die so generierten logische Programme wurden alle Antwortmengen mit Smodels Version 2.27 berechnet. Die Testreihe wurde auf meinem Heimrechner erhoben.

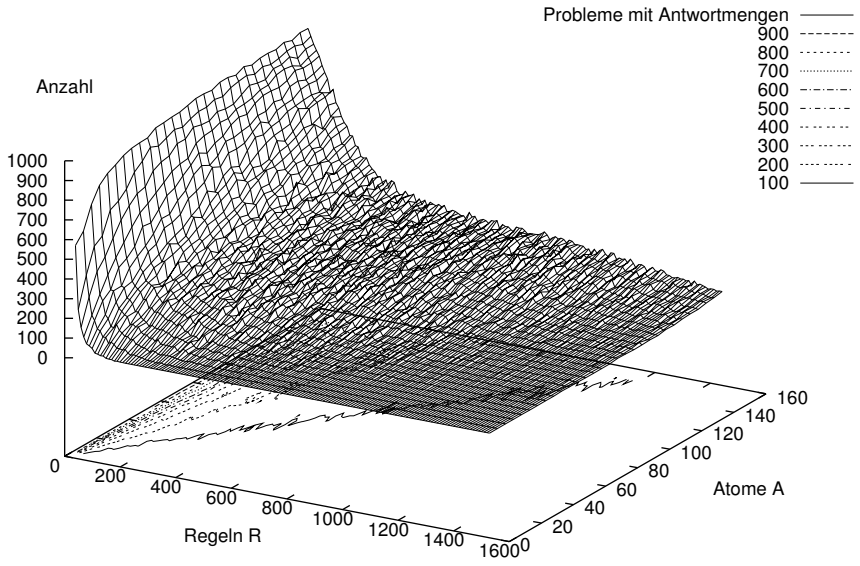

Abbildung 6.14: Anzahl der erfüllbaren Programme bei 2-LP

Abbildung 6.14 zeigt die Anzahl der logischen Programme des 2-LP Modells die erfüllbar sind, zwischen 10 und 1500 Regeln und 10 und 150 Atomen. Die Höhenlinien, auf dem Boden in der Regel-Atom-Ebene des Diagramms, zeigen ein annähernd lineares Verhalten, wobei die Verlängerung des gradlinigen Teils in die Nähe des Koordinatenursprungs kommt. Der R/A (Regeln/Atome) Faktor ist demnach auf diesen Höhenlinien ungefähr konstant. Dieser Sachverhalt unterstreicht die Aussagekraft der Skalierungsformel R/A.

Zu beachten ist, dass der Anteil der erfüllbaren Probleme nicht von der Anzahl der gesuchten Antwortmengen abhängt. Daher zeigt die Abbildung 6.14, von statistischen Ungenauigkeiten abgesehen, auch die Erfüllbarkeitskurve für den Fall, dass nur eine Antwortmenge gesucht wird. Auch hier ist zu erkennen, dass es anscheinend keine Phase gibt, in der die meisten Programme erfüllbar sind. Das Absinken der Anzahl der erfüllbaren Probleme mit der Zahl der Regeln, ist wie bei der Suche nach einer Antwortmenge bei cases1 zu erklären.

Der größtenteils lineare Verlauf der Höhenlinien lässt darauf schließen, dass sich an diesem Bild auch bei mehr Regeln oder Atomen nichts ändert.

Da bei dieser Testreihe alle Antwortmengen eines Programms berechnet wurden, ist die Anzahl der wrong choices gleich der Anzahl der choice points. Aussagen, die für die choice points gemacht werden, gelten also auch für die wrong choices.

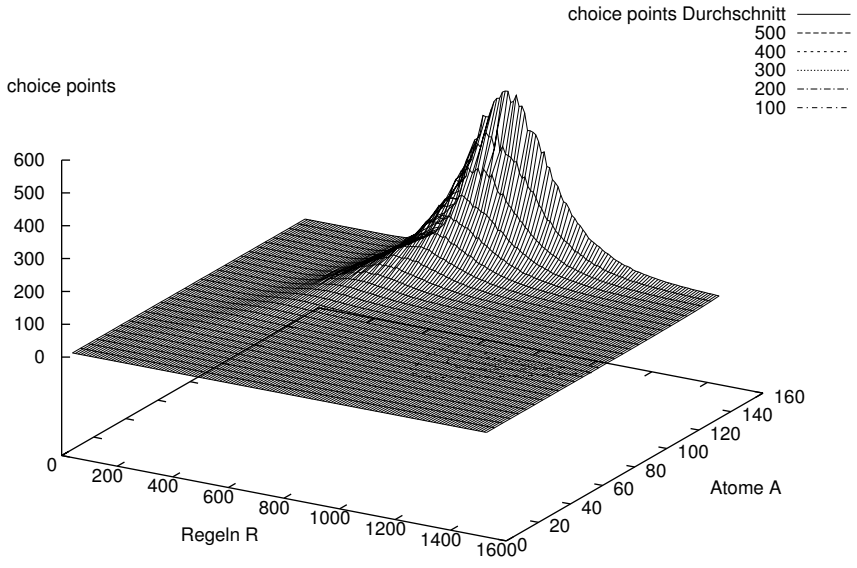

Abbildung 6.15: Durchschnittliche Anzahl der choice points bei 2-LP

Abbildung 6.15 zeigt die durchschnittliche Anzahl der choice points die auf der Suche nach den Antwortmengen durchlaufen wurden. Deutlich ist hier der schwere Bereich zu erkennen. Die Maxima der Kurven für feste Atomzahlen oder Regelzahlen liegen dabei ungefähr auf einer Geraden, bei rund $R/A = 5$, wobei sie mit steigender Regel- oder Atomzahl stark zunehmen.

Abbildung 6.16 zeigt den Median der Anzahl der choices points. Das Aussehen von Abbildung 6.16 ähnelt stark dem von Abbildung 6.15, nur dass die konkreten Werte kleiner sind. Der Durchschnitt und Median der choice points verhalten sich also ähnlich.

Abbildung 6.17 zeigt das Maximum der Anzahl der choice points für die Testpunkte. Auch sie ähnelt den beiden vorangegangenen Abbildungen 6.15 und 6.16. Allerdings ist sie wesentlich stärker zerklüftet. Weiterhin liegt die Höhe der Werte deutlich über denen der Durchschnitts- und Medianswerte zu liegen.

Abbildung 6.18 zeigt das Minimum der choice points in den jeweiligen Test-

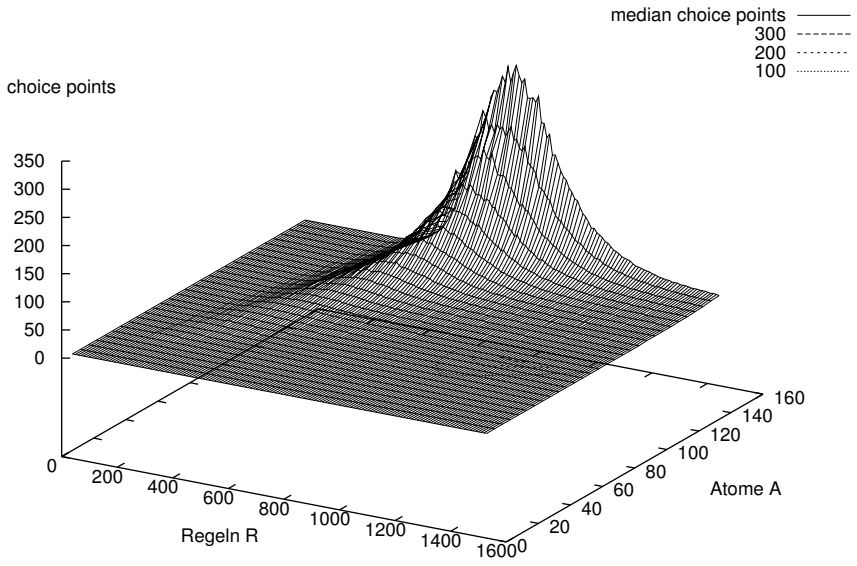

Abbildung 6.16: Median der Anzahl der choice points bei 2-LP

punkten. Die Werte die erreicht werden, sind im Verhältnis zu den anderen drei Abbildungen 6.15, 6.16 und 6.17 sehr gering. Wenn die Anzahl der generierten Testbeispiele pro Testpunkt erhöht wird, wird sich die Anzahl der minimalen choice points irgendwann überall bei 0 befinden. Die leichten Probleme, die mit wenigen choice points zu lösen sind, nehmen mit steigendem Durchschnitt der choice points ab, aber wenn genug Probleme generiert werden, werden immer welche zu finden sein.

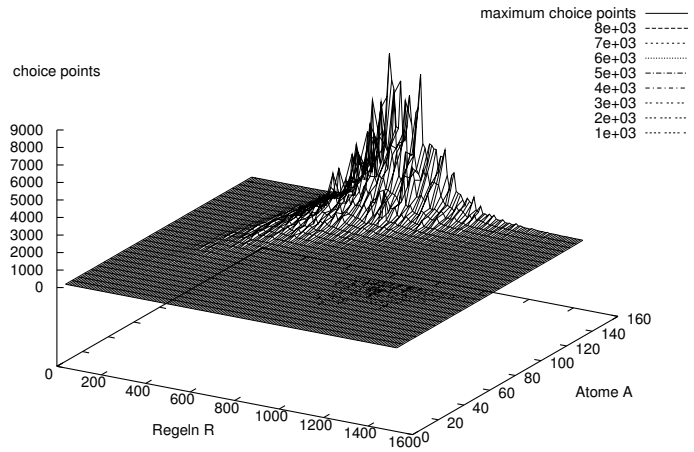

Abbildung 6.17: Maximale Anzahl der choice points bei 2-LP

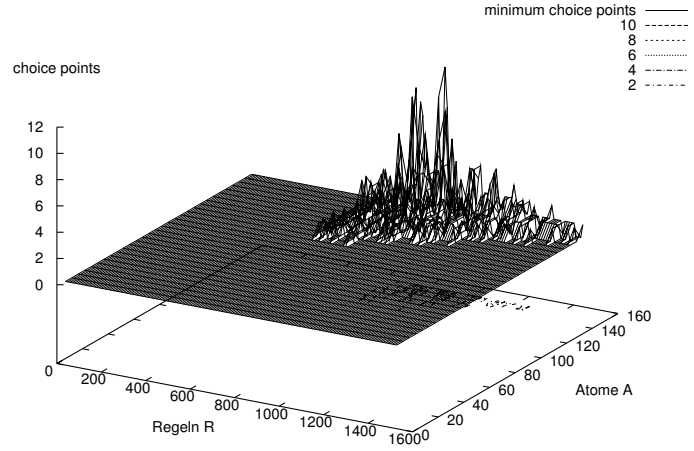

Abbildung 6.18: Minimale Anzahl der choice points bei 2-LP

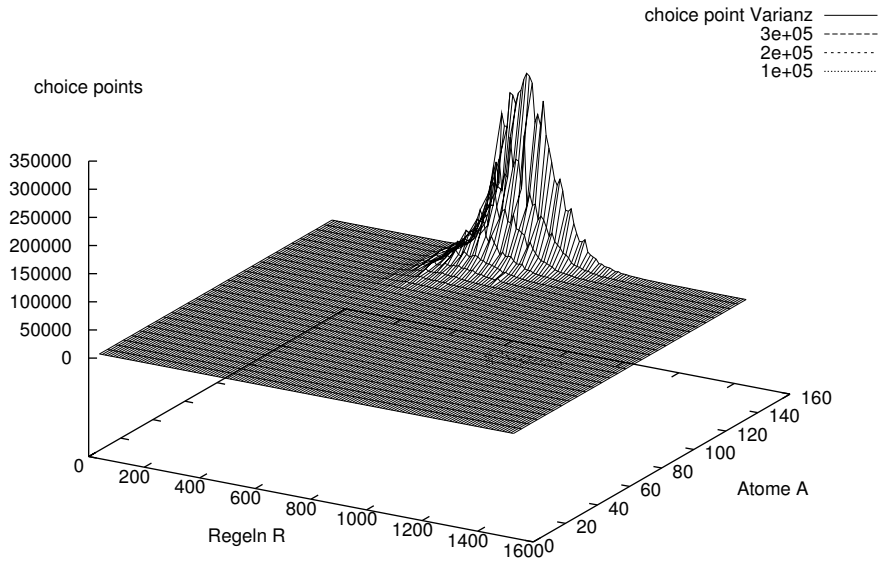

Abbildung 6.19: Varianz in der Anzahl der choice points bei 2-LP

Abbildung 6.19 zeigt die Varianz der choice points. Auch hier ist das gleiche Muster wie bei den drei Abbildungen 6.15, 6.16 und 6.17 zu finden.

Alle Abbildungen für die choice points (6.15, 6.16, 6.17, 6.18 und 6.19) zeigen die gleichen Muster, wie die Kurven für die choice points wenn eine Antwortmenge gesucht wird (6.1, 6.3, 6.4, 6.5 und 6.6), ausgedehnt auf eine dritte Dimension (Regelachse). Wenn von den konkreten Werten abgesehen wird, ist das Verhalten, das Kurven für die Berechnung einer und aller Antwortmengen zeigen, gleich. Die Erklärungen, die für die Kurven von Abbildung 6.4 (maximale choice points) und 6.5 (minimale choice points) der Testreihe cases1 gelten, gelten auch für die entsprechenden Kurven von Abbildung 6.17 (maximale choice points) und 6.18 (minimale choice points) der Testreihe cases.

Auch die Verteilung der Anzahlen von choice point innerhalb eines Testpunktes, bei der Berechnung aller Antwortmengen, ähnelt dem bei der Berechnung von einer Antwortmenge.

Wie schon in Abschnitt 4.1.2 Seite 34 erwähnt, sind die Ausgabewerte der Suche bei unerfüllbaren Problemen (Probleme ohne Antwortmenge) unabhängig davon, wie viele Antwortmengen gesucht werden. Daher sind die Ergebniswerte bei unerfüllbaren Problemen bei den Testreihen cases und cases1, von statistischen Ungenauigkeiten abgesehen, gleich.

Abbildung 6.20 zeigt die durchschnittliche Zeit, die zum Berechnen aller Antwortmengen benötigt wurde. Auch hier fällt die große Ähnlichkeit zu der Abbil-

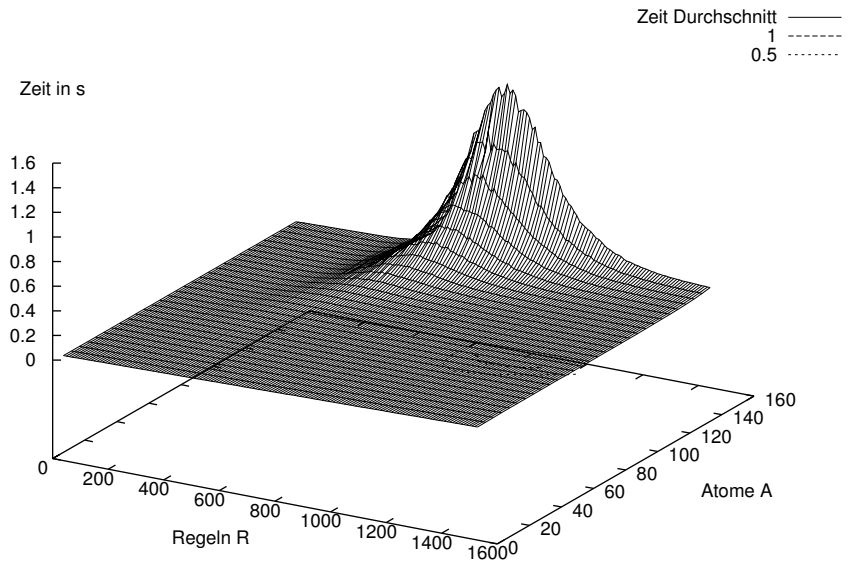

Abbildung 6.20: Durchschnittliche Zeit bei 2-LP

dung 6.15 auf. In der Tat ist dem auch so, dass bei allen Testreihen, das Verhalten der Zeiten, aber auch anderer Berechnungsaufwandsparameter, wie der Anzahl der wrong choices und Wahrheitswertzuweisungen, dem der choice points stark ähnelt.

Die Ähnlichkeit der Zeit und der choice points ist auch aus Abbildung 6.21 ersichtlich, in der der choice point (/) Zeit Quotient dargestellt ist. Für dieses Diagramm wurden für die einzelnen Probleme die ermittelte Anzahl der choice points durch die ermittelte Zeit dividiert ((choice points)/Zeit). Dargestellt ist jeweils der Durchschnitt für die einzelnen Testpunkte. Zu sehen ist, dass im schweren Bereich dieser Quotient zwar stark zunimmt, dass aber der Verlauf der Maxima für verschiedene Atomanzahlen bei mehr Atomen stark abflacht. Da die Anzahl der choice points auf einer R/A Linie mindestens exponentiell wächst (siehe 6.23), der Unterschied zwischen choice point und Zeit aber nicht, verhält sich die Zeit fast immer genauso wie die choice points.

Abbildung 6.22 zeigt die minimal benötigte Zeit. Deutlich zu sehen ist, die nahezu lineare Zunahme mit der Anzahl der Regeln. Der Unterschied zu den minimalen choice points aus Abbildung 6.18 ist also ein Faktor, der linear mit der Anzahl der Regeln wächst. Zu beachten ist, dass das lineare Wachstum der Zeit mit der Anzahl der Regeln, schon allein durch die Zeit entsteht, die zum Einlesen der Regeln benötigt wird. Bei allen Generierungsmodellen muss die Zeit also mindestens linear mit der Anzahl der Regeln wachsen.

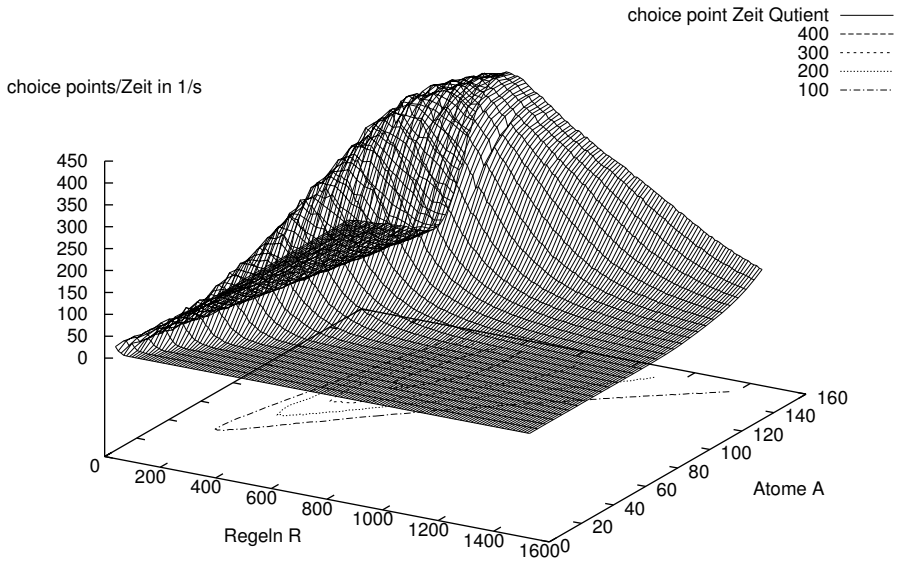

Abbildung 6.21: Durchschnittlicher choice point Zeit Quotient bei 2-LP

Die angegebene Zeit ist allerdings die von Smodels benötigte Zeit und berücksichtigt nicht LParse. Da LParse einige Programme verkürzt, in dem es z.B. doppelte Regeln löscht, kann es durchaus sein, dass Smodels Programme liest, die nicht mehr die Länge haben, die sie bei ihrer Generierung hatten. Allerdings ist der Anteil der Programme, die von LParse um einen signifikanten Anteil (bei tausend Regeln eine Regel zu löschen ändert nicht viel) verkürzt werden können, im untersuchtem Bereich gering. Um so weniger Atome es gibt, um so größer ist der Anteil der stärker verkürzbaren Programme, da weniger unterschiedliche Regeln generiert werden können. Bei 2-LP mit 10 Atomen können immerhin 4000 ($= 10 * 2 * 10 * 2 * 10$ mögliche Kopfatome mal mögliche Körperliterale für jedes Literal im Körper) verschiedene Regeln generiert werden.

Die Ähnlichkeit des Verhaltens der choice points und der Zeit ist also sehr groß, deshalb können durchaus die choice points zur weiteren Betrachtung des Berechnungsaufwands bei dieser Smodels Version herangezogen werden. Allerdings ist das Verhalten, wie Abbildung 6.21 zeigt, nicht identisch, so dass allein die Untersuchung der choice points zur Untersuchung des Berechnungsaufwands (im Sinne der benötigten Zeit) eines ASP-Solvers wohl nicht ausreicht.

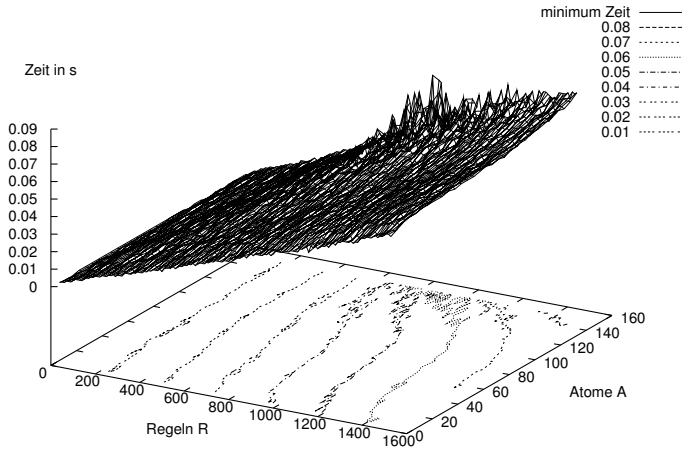

Abbildung 6.22: Minimale Zeit bei 2-LP

6.2.2.1 Verhalten bei konstantem R/A Faktor

Um das Wachstumsverhalten bei der Berechnung aller Antwortmengen genauer zu untersuchen, werden hier, für verschiedene konstante Werte der Skalierungsfunktion R/A, die Kurven untersucht.

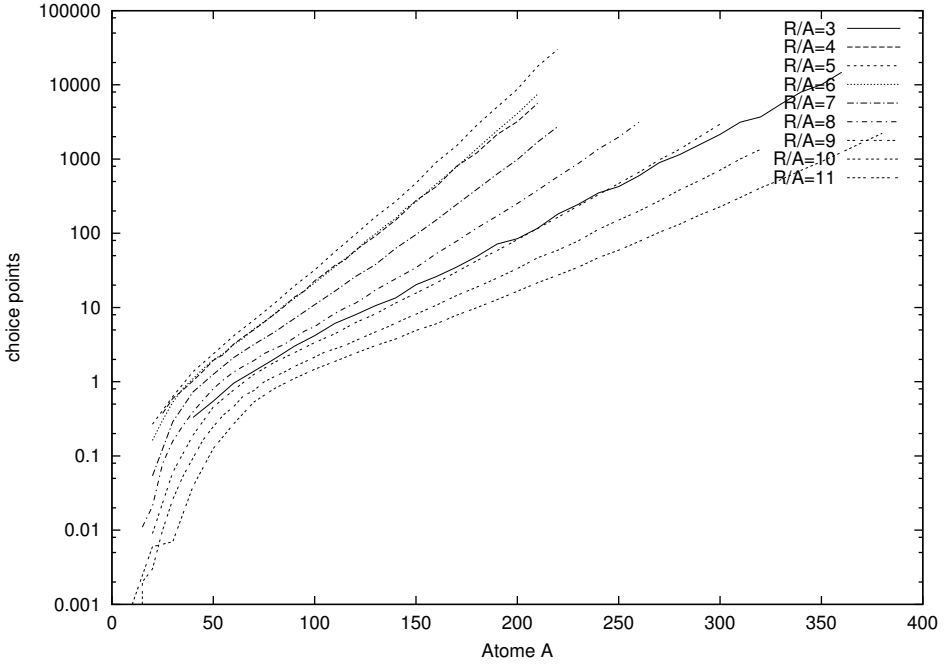

Abbildung 6.23: Durchschnittliche choice points bei 2-LP für verschiedene R/A Faktoren

Das mindestens exponentielle Wachstum der Anzahl der choice points ist in Abbildung 6.23 zu sehen. Hier wurden für die R/A Faktoren 3 bis 11 die Kurven dargestellt. Nach einem sehr starken Anstieg für Problemklassen mit durchschnittlich weniger als einem choice point, gehen die Kurven in eine stabile Phase über. Das Anstiegsverhalten aller Kurven ist sehr ähnlich. Wenn die Kurven in unterschiedlichen Diagrammen dargestellt werden, sind sie ohne eine Beschriftung der x-Achse nicht zu unterscheiden. Alle Kurven enden außerdem ungefähr in der gleichen Größenordnung von choice points.

Der Bereich unterhalb von einem choice point, kann als Anlaufphase gesehen werden. Der starke Anstieg dort, kann von einer linearen oder polynomiellen Komponente des Wachstums herrühren, welche dort noch einen stärkeren Einfluss haben, als der exponentielle Anteil.

Außer bei 1-LP und 0-LP findet sich dieses Muster auch bei anderen k-LP Faktoren. Sehr kleine und große R/A Faktoren sind allerdings schwierig zu untersuchen, da sehr große Programme generiert werden müssen. So ist der choice

point Durchschnitt für $R/A = 1$ im untersuchten Bereich bei 2-LP bis 1100 Atome annähernd konstant unter 0.2 choice points.

Demnach ist das Wachstum überall für k größer 1 mindestens exponentiell, nur der konkrete Anstieg unterscheidet sich. Das führt dazu, dass für eine Verdopplung der durchschnittlichen benötigten choice points (oder Zeit) die Programme nur um einen konstanten Faktor (an Regeln und Atomen, so dass R/A konstant ist) vergrößert werden müssen. Damit können auch in den leichten Bereichen (bei niedrigen oder hohen R/A Faktoren) die Probleme schnell schwer bis unlösbar werden. Exponentielles Wachstum ist sozusagen ein zweischneidiges Schwert, da es nicht nur bedeutet, dass der Aufwand exponentiell mit der linearen Zunahme der Programmgröße zunimmt, sondern auch exponentiell mit der linearen Abnahme der Programmgröße abnimmt. Es kann schnell der Eindruck entstehen, dass in bestimmten Bereichen (bei bestimmten R/A Werten) das Wachstum geringer als exponentiell ist, nur weil die Größe der untersuchten Programme noch zu klein ist und der exponentielle Anteil nicht signifikant ist.

Für die Kurven aus Abbildung 6.23 und weiteren R/A Faktor Kurven wurden exponentielle Funktionen, die sich ihnen annähern, approximiert. Dabei wurde wie bei der Approximation für cases1, zur Berechnung einer Antwortmenge, für Abbildung 6.9 Seite 53 vorgegangen.

Im Nachfolgenden seien einige Approximationen aufgeführt, wobei $f_k(R/A, R)$ die Funktion für k-LP mit dem Faktor R/A ist.

$$
\begin{aligned}
f_2(2,R) &= 2^{R*0.0065-4.0} \\
f_2(3,R) &= 2^{R*0.016-3.0} \\
f_2(4,R) &= 2^{R*0.0184-3.0} \\
f_2(5,R) &= 2^{R*0.016-3.0} \\
f_2(6,R) &= 2^{R*0.0125-3.0} \\
f_2(7,R) &= 2^{R*0.0094-3.0} \\
f_2(8,R) &= 2^{R*0.007-3.0} \\
f_2(9,R) &= 2^{R*0.0053-3.0} \\
f_2(10,R) &= 2^{R*0.0041-3.0} \\
f_2(11,R) &= 2^{R*0.0033-3.0} \\
f_2(12,R) &= 2^{R*0.0027-3.0}
\end{aligned}
$$

Abbildung 6.24 zeigt exemplarisch die Kurve für den R/A Faktor 5 mit der dazugehörigen Approximation $f_2(5, R)$. Bei anderen R/A Faktoren sind die Verhältnisse von Originalkurve zu Approximation ähnlich. Die Originalkurven für $R/A > 2$ wachsen alle leicht stärker als die jeweiligen exponentiellen Approximationen und beschreiben einen nach oben offenen Bogen. Es ist natürlich auch hier nicht ausgeschlossen, dass das Wachstum für noch größere Anzahlen von Regeln

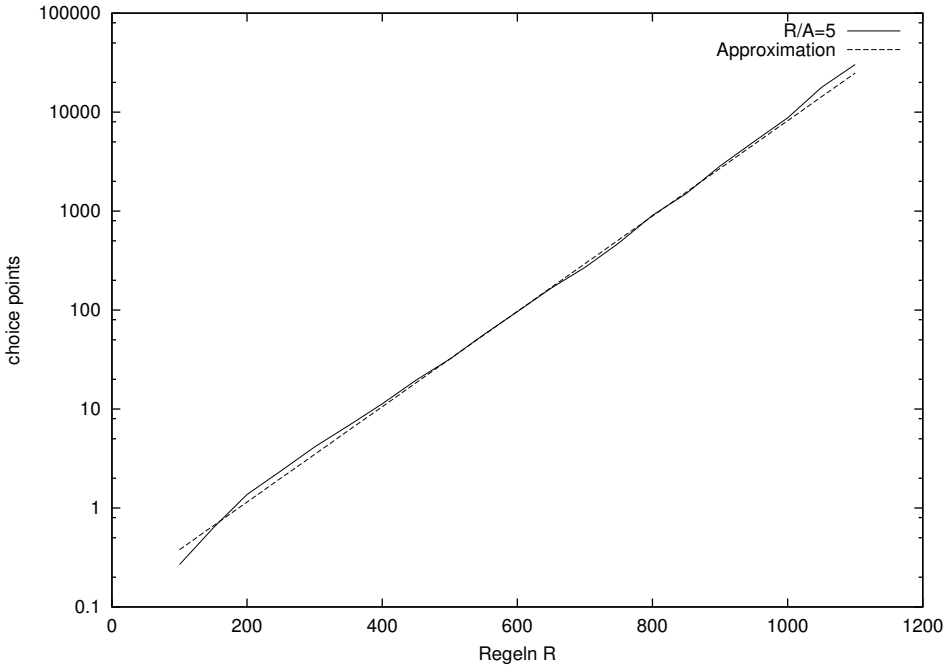

Abbildung 6.24: Durchschnittliche choice points für 2-LP bei den Faktor $R/A = 5$ mit Approximation

in ein rein exponentielles Wachstum übergeht.

Wenn die Approximation $f_2(5, R) = 2^{R*0.016-3.0}$ der Kurve mit der Approximation $f_k^1(5, R) = 2^{R*0.0155-2.8}$ der Kurve, bei der Suche nach einer Antwortmenge auch für 2-LP und dem gleichen Faktor $R/A = 5$, verglichen wird, ist der Anstieg der Kurven nur leicht unterschiedlich.

Interessant ist auch Abbildung 6.25, sie zeigt die choice point Zeit Quotienten für die R/A Faktoren 3 bis 11. Bei allen Kurven deutlich zu sehen ist ein wenig-viel-wenig Muster. Während in der Anfangsphase durchschnittlich am wenigsten choice points pro Zeiteinheit besucht werden, steigt mit der Atomzahl die Zahl der choice points pro Zeiteinheit auf ein Maximum und sinkt danach wieder ab. In der 3-dimensionalen Abbildung 6.21 für den choice point Zeit Quotienten war dieses Verhalten noch nicht zu sehen, da das Maximum aller Kurven in Abbildung 6.25 bei rund 150 Atomen liegt, dem Ende der Skala in Abbildung 6.21 . Anzunehmen ist, dass sich die Kurven mit steigender Anzahl von Atomen weiter abflachen.

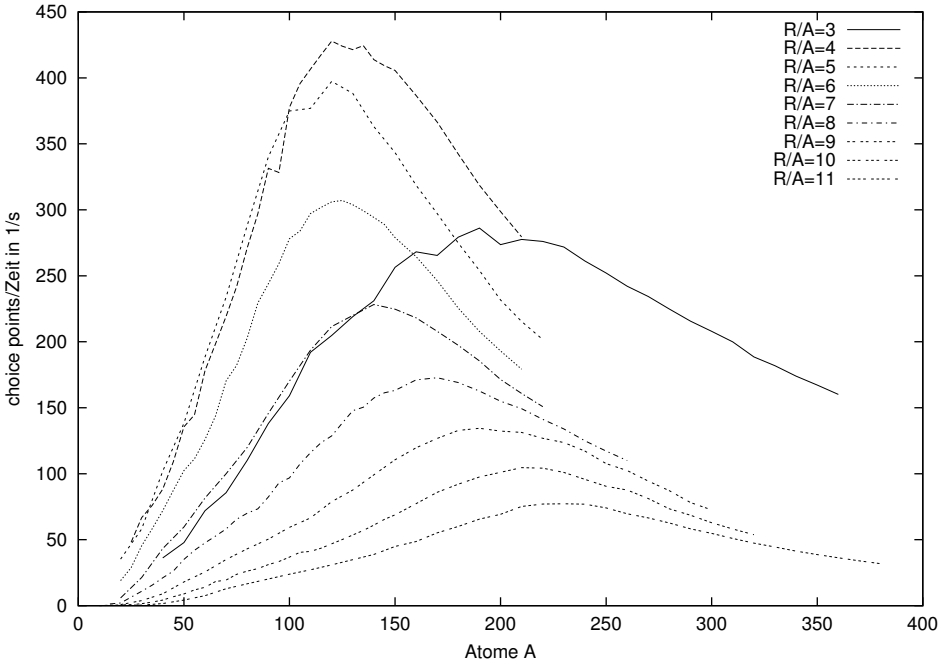

Abbildung 6.25: 2-LP choice points Zeit Quotient für verschiedene R/A Faktoren

6.2.2.2 Vergleich von erfüllbaren und unerfüllbaren Problemen

Bei der Suche nach allen Antwortmengen kann die Anzahl der Antwortmengen exponentiell mit der Anzahl der Atome wachsen. Hier wird untersucht welche Auswirkung die Anzahl der Antwortmengen auf den Berechnungsaufwand hat, indem Probleme ohne Antwortmengen (unerfüllbare Probleme) mit Problemen mit Antwortmengen (erfüllbare Probleme) verglichen werden.

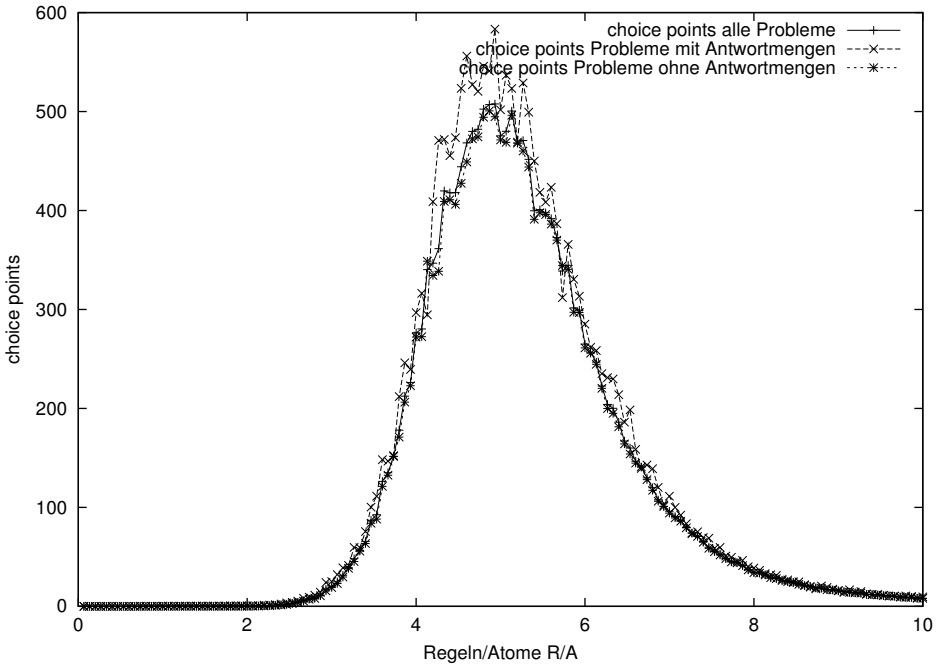

Abbildung 6.26: Durchschnittliche choice points für erfüllbare, unerfüllbare und alle Probleme für 2-LP bei 150 Atomen

Wie in Abbildung 6.26 zu sehen, unterscheidet sich die durchschnittliche Anzahl der choice points, welche für erfüllbare, unerfüllbare und alle Probleme benötigt werden, nicht sehr. Für erfüllbare Probleme werden im Durchschnitt dabei etwas mehr choice points (Zeit) benötigt als für unerfüllbare. Das kann nach dem realen Suchmodell aus Abschnitt 4.1.1.3 Seite 31 damit erklärt werden, dass, wenn alle Antwortmengen gesucht werden, erfüllbare Probleme die sind, bei denen auch die letzte Ebene des Suchbaums bearbeitet wird. Bei unerfüllbaren Problemen geschieht dies nicht. Somit ist der Suchbaum bei erfüllbaren Problemen etwas tiefer und es gibt im Durchschnitt etwas mehr choice points. Zu beachten ist hier noch, dass im schwerem Bereich nur noch wenige erfüllbare Probleme vorhanden sind, wodurch die Kurve für erfüllbare Probleme auf der Basis von wenigen Problemen berechnet wurde und damit ungenauer ist.

Wie in Abbildung 6.27 zu sehen, liegt der Quotient vom choice point Durch-

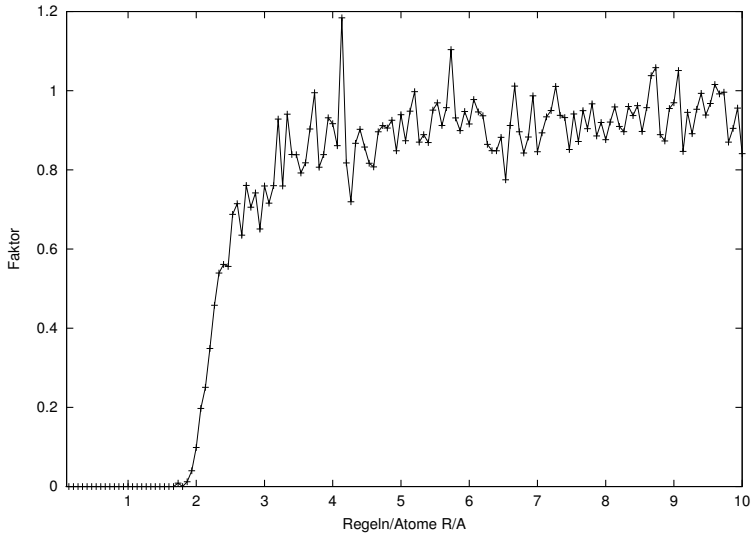

Abbildung 6.27: Quotient der choice points für unerfüllbare und erfüllbare Probleme für 2-LP bei 150 Atomen

schnitt für unerfüllbare Probleme, zu choice point für erfüllbare Probleme ((unerfüllbare Probleme)/(erfüllbare Probleme)), knapp unter eins. In der Anfangsphase allerdings, bis $R/A = 2$, liegt die durchschnittliche Anzahl der choice points für unerfüllbare Probleme bei 0 und nähert sich dann den Werten für erfüllbare Probleme an. Bei wenigen Regeln sind zwar nur wenige unerfüllbare Probleme vorhanden (bei 10 Regeln 29 unerfüllbare Probleme), aber ihre Anzahl steigt schnell an. Bei 270 ($R/A = 1.8$) Regeln sind 711 unerfüllbare Probleme (= 71.1% aller Probleme) vorhanden, die durchschnittliche Anzahl der choice points der unerfüllbare Probleme liegt aber immernoch bei 0. Bei erfüllbare Problemen in diesem Bereich gibt es aber durchaus choice points (=wrong choices). Auch hier ist also der Sachverhalt wie bei der Testreihe cases1.

In Abbildung 6.28 sind die Kurven für erfüllbare, unerfüllbare und aller Probleme bei 2-LP für den Faktor $R/A = 5$ dargestellt. Danach gleichen sich die Werte für erfüllbare und unerfüllbare Probleme mit steigender Problemgröße immer mehr an. Der Quotient vom choice point Durchschnitt für unerfüllbare Probleme, zu choice point für erfüllbare Probleme ((unerfüllbare Probleme)/(erfüllbare Probleme)) nähert sich mit steigender Atom- oder Regelzahl der 1 an. Bei anderen R/A Faktoren größer 1 gleicht das Verhältnis von unerfüllbaren zu erfüllbaren Problemen dem hier angesprochenen Verhältnis.

Da das Verhalten des Berechnungsaufwands von erfüllbaren und unerfüllbaren Problemen bei der Suche nach einer oder aller Antwortmengen ähnlich ist und das Verhalten bei unerfüllbare Problemen unabhängig von der Anzahl der gesuchten Antwortmengen ist, ist das Verhalten unabhängig davon, ob eine oder alle Antwortmengen gesucht werden und ob die Probleme erfüllbar oder unerfüllbar sind,

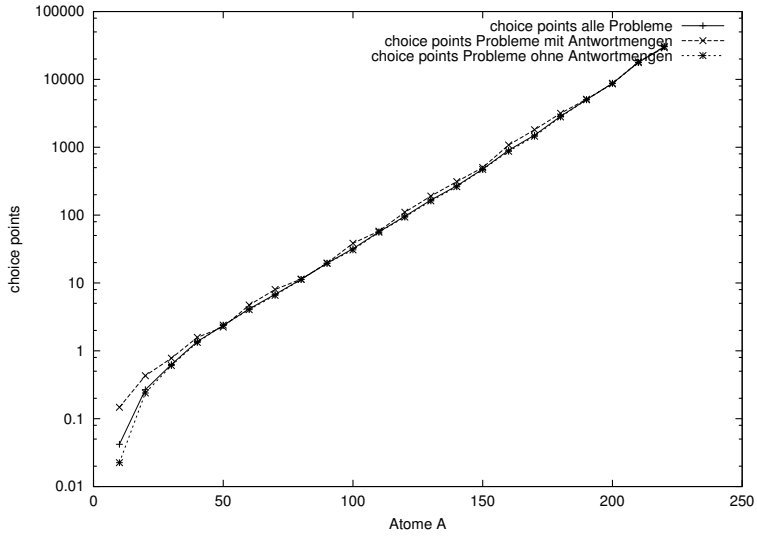

Abbildung 6.28: Durchschnittliche choice points für erfüllbare, unerfüllbare und aller Probleme für 2-LP bei dem Faktor $R/A = 5$

immer ähnlich. Danach gelten alle Aussagen, die über das Berechnungsaufwandsverhalten von k-LP getroffen wurden, sowohl für die Suche nach einer, wie nach allen Antwortmengen.

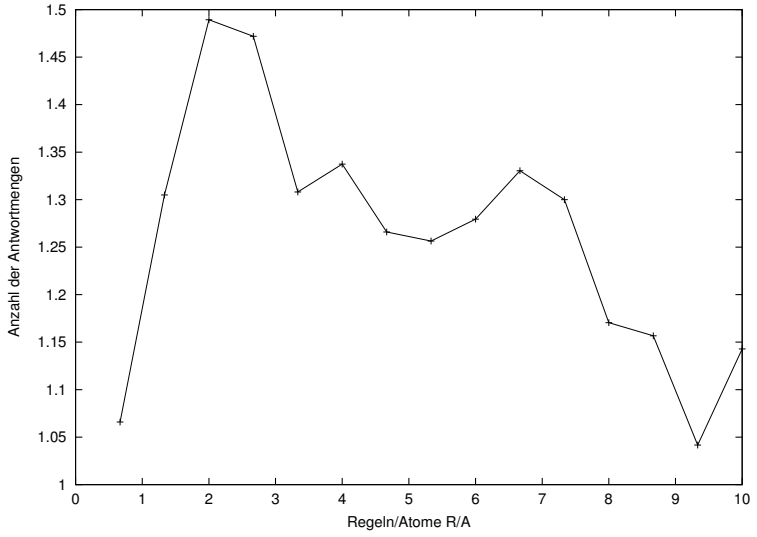

Abbildung 6.29: Anzahl der Antwortmengen bei erfüllbaren Problemen für 2-LP bei 150 Atomen

Die Abbildung 6.29 zeigt die durchschnittliche Anzahl der Antwortmengen für

erfüllbare Probleme bei 150 Atomen und wurde aus der Testreihe casesAll entnommen. Die Testreihe casesAll entspricht der Testreihe cases, mit den Unterschieden, dass bei ihr die Anzahl der Antwortmengen mit erhoben wurde, sie auf den Universitätsrechnern lief und weniger Testpunkte berechnet wurden. Danach haben erfüllbare Programme durchschnittlich rund 1 bis 1.5 Antwortmengen, also sehr wenige im Verhältnis zu der Anzahl von choice points. Eine Antwortmenge stellt natürlich bei erfüllbaren Problemen das Minimum an Antwortmengen dar.

Zu beachten ist, dass die Anzahl der choice points mindestens gleich der Anzahl der Antwortmengen minus 1 sein muss (min(Anzahl choice points) = (Anzahl Antwortmengen)−1), da jeder choice point genau einen Ast zu dem Suchbaum hinzufügt, der Suchbaum ohne choice points einen Ast hat und jede Antwortmenge ein Blatt im Suchbaum ist.

6.2.2.3 Verhalten bei verschiedenen k Werten für k-LP

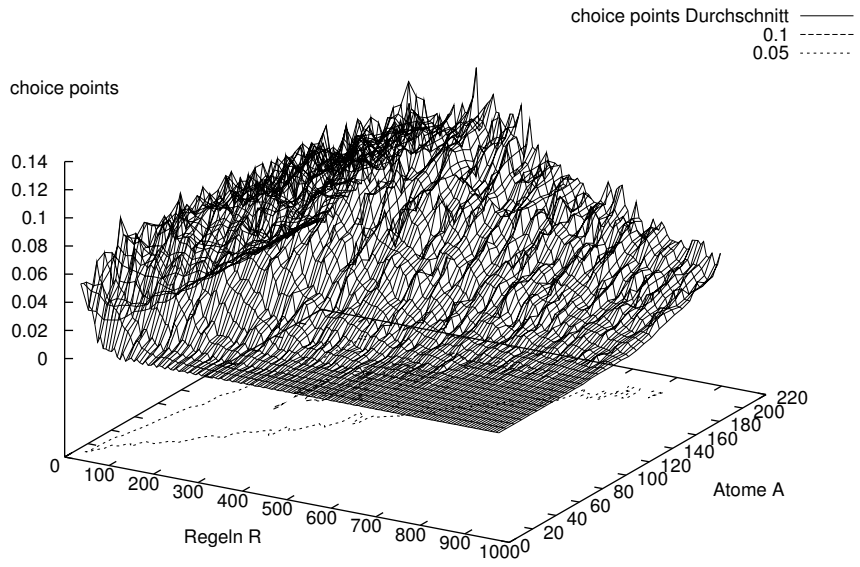

Abbildung 6.30: Durchschnittliche choice points für 1-LP

Abbildung 6.30 zeigt die durchschnittlichen choice points bei 1-LP. Auch hier gibt es, wie bei 2-LP, ein leicht-schwer-leicht Muster, wobei das Maximum ungefähr bei dem $R/A = 2$ Faktor liegt. Allerdings ist kein exponentielles Wachstum zu sehen. Die annähernd linearen Höhenlinien auf dem Boden des Diagramms deuten auf ein geringes Wachstum des Berechnungsaufwandes hin. Die Werte liegen alle weit unter einem choice point, so dass Aussagen, wie für Abbildung 6.23, schwierig sind. Im dargestellten Bereich sind die Probleme trivial. Beim Faktor $R/A = 1$ wurde eine Testreihe bis zu 8000 Regeln bzw. Atomen durchgeführt, die Anzahl der durchschnittlichen choice points blieb dabei ungefähr konstant und die durchschnittliche Zeit nahm linear zu. Bei anderen R/A Faktoren war das Verhalten ähnlich, auch wenn sie nicht so weit untersucht wurden, so dass auf ein zumindest polynomielles Wachstum geschlossen werden kann. Dies ist auch das zu erwartende Ergebnis, da 1-LP (wie 2-SAT) polynomiell lösbar ist. Mit Gewissheit zu sagen das 1-LP polynomiell mit der verwendeten Smodels-Version lösbar ist, ist aus den oben aufgeführten Gründen (vielleicht ist das exponentielles Wachstum in dem untersuchtem Bereich noch zu gering) allerdings unmöglich.

Abbildung 6.31 zeigt die durchschnittlichen choice points bei 3-LP. Wieder ist ein diesmal stark ausgeprägtes leicht-schwer-leicht Muster zu erkennen, dessen Maximum bei ungefähr $R/A = 8$ liegt. Das Diagramm zeigt nur die Messwerte

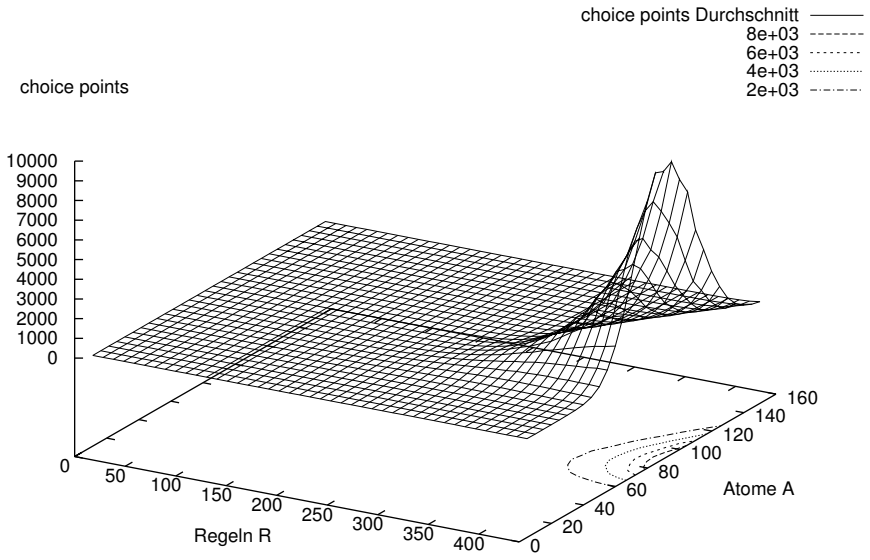

Abbildung 6.31: Durchschnittliche choice points für 3-LP

bis zu 450 Regeln, da danach der Berechnungsaufwand für weitere Berechnungen zu groß wird. Der Anstieg der durchschnittlichen choice points ist also um einiges stärker als der Anstieg bei 2-LP.

Abbildung 6.32 beinhaltet die Kurven für die choice points bei 50 Atomen für 1-LP, 2-LP und 3-LP. Zu sehen ist, dass sich das Maximum nach rechts, in Bereiche von höheren R/A Faktoren, verschiebt. Weiterhin gibt es bei linearen Anstieg von k, einen exponentiellen Anstieg der Werte für choice points, in den unterschiedlichen Bereichen der Kurve, z.B. von Maximum zu Maximum.

Von den oben aufgeführten Unterschieden abgesehen, sind die Werte und Kurven für die Testreihe cases sehr ähnlich zu denen von cases1.

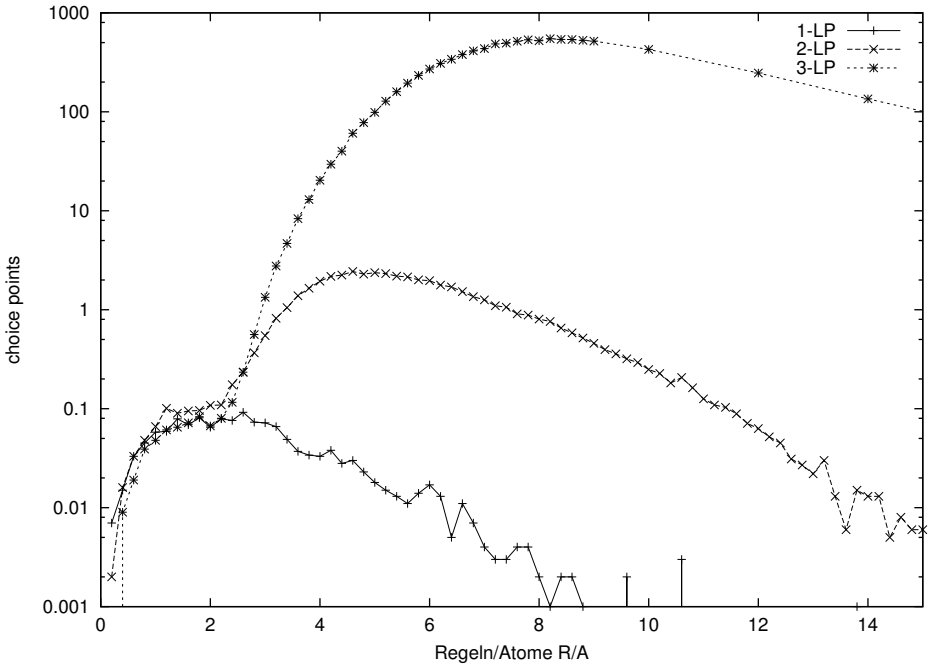

Abbildung 6.32: Durchschnittliche choice points für 1-LP, 2-LP und 3-LP bei 50 Atomen

6.3 Gemischte Körperlänge

6.3.1 Berechnung einer Antwortmenge bei *k*-pLP

Die Testreihe casesMix wurde mithilfe von Smodels Version 2.28 und dem *k*-pLP Programmgenerierungsmodell für gemischte Körperlänge aus Abschnitt 5.1.2 von Seite 41 durchgeführt. Gesucht wurde jeweils eine Antwortmenge. Die Testreihe lief auf den Universitätsrechnern.

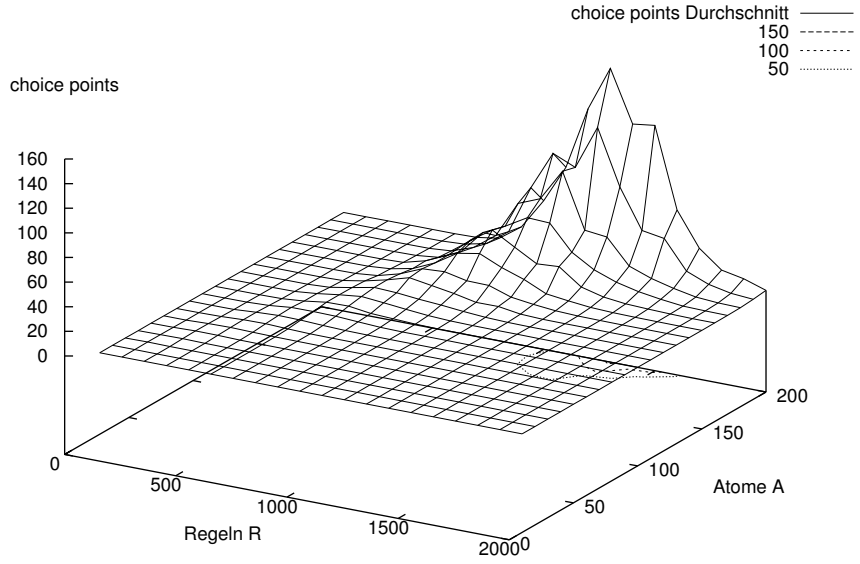

Abbildung 6.33: Durchschnittliche choice points bei durchschnittlich 3.8 Literalen

Abbildung 6.33 zeigt die durchschnittlichen choice points bei durchschnittlich 3.8 Literalen pro Regelkörper. Auch hier wiederholt sich das Muster der Testreihe cases aus Abbildung 6.15 (Seite 59) und 6.31 (Seite 75). Die choice point Werte sind geringer als beim 3-LP Modell, in dem durchschnittlich 3 Literale pro Regelkörper vorkommen. Probleme mit gemischter Körperlänge sind leichter zu lösen, als Probleme des *k*-LP Modells mit vergleichbaren Werten für die Anzahl von Atomen, Regeln und durchschnittlichen Literalen. In der Tat wiederholen sich die Muster für choice points, wrong choices, Zeit und Wahrheitswertzuweisungen, von cases1 mit niedrigeren Werten, so dass die Ergebnisse für diese Parameter von cases, mit den Einschränkungen von cases1, auch für casesMix gelten.

Das Berechnungsaufwandswachstum ist bei casesMix geringer als bei cases1. Der Grund dafür ist mit dem Modell aus Abschnitt 4.1.1.3 auf Seite 31 gut zu

erklären. Die Programme von casesMix sind eine Mischung aus Regeln mit unterschiedlicher Körperlänge. Darin sind auch 0-LP Regeln, also Fakten, und 1-LP Regeln enthalten. Fakten benötigen keine Voraussetzungen (Informationen) um Informationen zu liefern, sie sind wahr. Regeln mit Körperlänge 1 (1-LP) können mit wenig Aufwand weitere Informationen liefern. Im Gegensatz zu dem *k*-LP Modell für feste Körperlänge, mit *k* größer 1, können schon am Anfang mit wenig Aufwand Informationen gewonnen werden, welche die weitere Suche vereinfachen. Das dennoch exponentielle Wachstum kommt wohl von dem Anteil der Regeln mit Körperlänge größer 1, bei denen die zusätzlichen Informationen, die durch die Fakten gewonnen wurden, nur zum Teil oder gar nicht weiter helfen.

Im Nachfolgenden werden nicht weiter die Ähnlichkeiten beleuchtet, sondern neue Muster betrachtet. Für die Verifizierung der Ähnlichkeiten, sei auf die beiliegende DVD verwiesen.

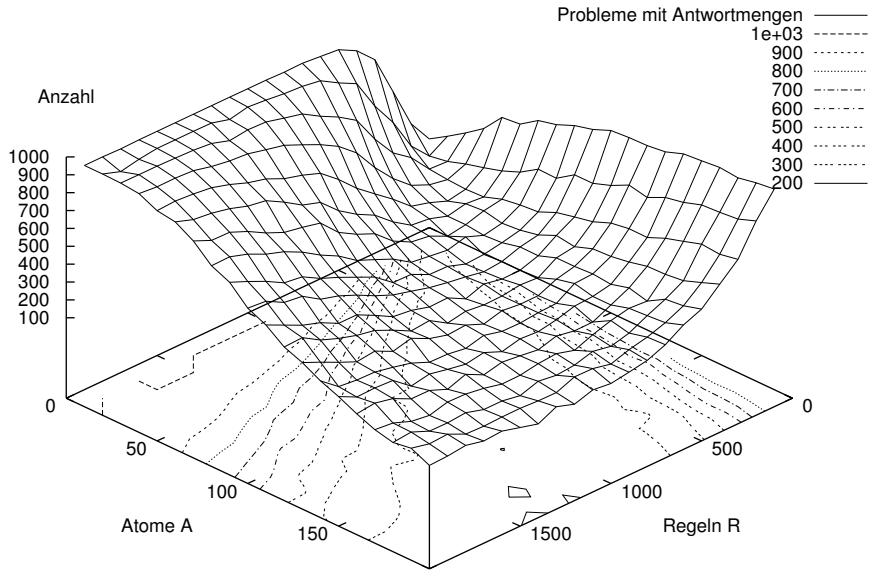

Abbildung 6.34: Anzahl der erfüllbaren Probleme mit durchschnittlich 3.8 Literalen

Abbildung 6.34 zeigt die Anzahl der erfüllbaren Probleme bei durchschnittlich 3.8 Körperliteralen. Hier tritt ein deutlicher Unterschied zu den Kurven (z.B. 6.14) für k-LP (konstante Körperlänge) zu Tage. Es gibt nicht nur viele erfüllbare Probleme bei wenigen Regeln pro Atom, sondern auch bei vielen Regeln pro Atom. Auch dies liegt am Anteil der Fakten. Fakten sind nicht widerlegbar. Wenn also ein Programm entsprechende Fakten beinhaltet, um die potentiellen Widersprüche zu

vermeiden, ist es erfüllbar. Mit Zunahme der Regeln nimmt auch die Zahl der Fakten zu. Damit ist ein größerer Anteil der Probleme erfüllbar. Bei wenigen Regeln pro Atom bewirkt die Zunahme der Regeln allerdings, wie bei cases, eine Abnahme der erfüllbaren Programme, da der Anteil der Fakten noch nicht ausreicht um wesentlich mehr Programme erfüllbar zu machen.

Die Zunahme der Fakten bei vielen Regeln pro Atom ist auch ein Grund für den in Abbildung 6.33 zu sehenden niedrigen Berechnungsaufwand in diesem Bereich. Alles, was aus Fakten zu schließen ist, benötigt nicht viel Berechnungsaufwand.

Auch hier sind, wie in cases, die Höhenlinien wieder annähernd Geraden, die in Richtung Ursprung verlaufen. Was für die Ähnlichkeit zu cases und der Skalierungsformel R/A auch bei casesMix spricht.

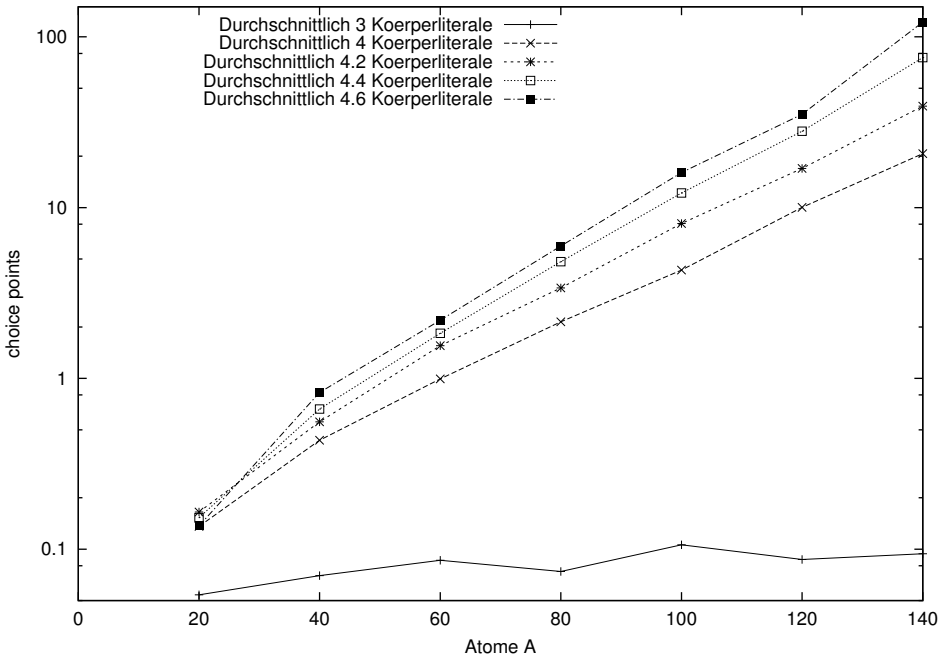

Abbildung 6.35: Durchschnittliche choice points bei unterschiedlichen durchschnittlichen Anzahlen von Körperliteralen und $R/A = 5$

Abbildung 6.35 zeigt bei dem Faktor $R/A = 5$ für die durchschnittlichen Anzahlen von Körperliteralen von 3, 4, 4.2, 4.4 und 4.6 die durchschnittlichen choice points. Bei allen Literalanzahlen gibt ein exponentielles Wachstum, auch wenn es bei 3 relativ schwach ausfällt. Mit steigender Zahl der durchschnittlichen Literale, wird der Anstieg der Wachstumskurven größer.

Leider ist ein dreidimensionales Diagramm, das als zwei seiner Achsen Literale und choice points besitzt, wegen des starken Anstiegs der choice points mit der durchschnittlichen Literalzahl, wenig hilfreich. Deshalb nähere ich mich dem

Problem von einer anderen Seite.

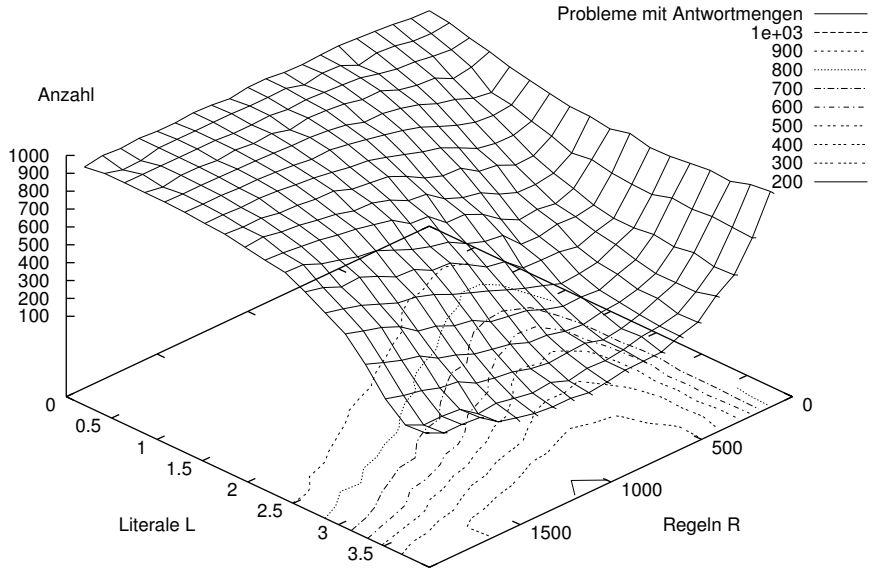

Abbildung 6.36: Anzahl der erfüllbaren Probleme bei unterschiedlichen durchschnittlichen Anzahl von Körperliteralen und $A = 150$

Die Abbildung 6.36 zeigt die Anzahl von erfüllbaren Problemen, zu Regeln- und Körperliteralanzahlen bei 150 Atomen. Das Diagramm ähnelt dem aus Abbildung 6.34 . Für das viel-wenig-viel Muster, wenn die Literalanzahl konstant ist, ist die gleiche Erklärung wie bei Abbildung 6.34 anzuwenden. Mit sinkender Literalzahl nimmt der Anteil der erfüllbaren Probleme zu, da der Anteil der kurzen Regeln und damit auch der Fakten zunimmt. Um so höher die durchschnittliche Literalzahl wird, um so mehr lange Regeln gibt es und um so seltener ist ein Programm erfüllbar.

Im Zusammenhang mit der Abbildung 6.37 kann diese Kurve zum Einschätzen des Verhaltens des Berechnungsaufwands bei unterschiedlichen durchschnittlichen Anzahlen von Körperliteralen genutzt werden.

Das in Abbildung 6.37 zu sehende Verhalten ist charakteristisch für casesMix. Es zeigt die durchschnittlichen choice points und erfüllbaren Probleme bei durchschnittlich 3 Körperliteralen und 150 Atomen. Das Maximum der choice points liegt in der Nähe des Minimums der erfüllbaren Probleme. Da dieses Verhalten in allen bei casesMix untersuchten Bereichen auftritt, ist das Maximum des Berechnungsaufwands immer in der Nähe des Minimums der Anzahl der erfüllbaren Probleme zu finden.

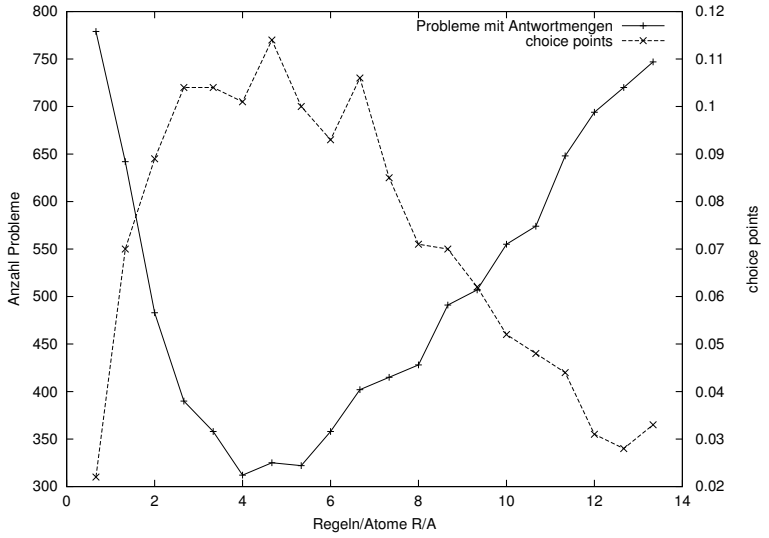

Abbildung 6.37: Anzahl der erfüllbaren Probleme und durchschnittliche choice points bei durchschnittlich 3 Literalen und $A = 150$

Zusammen mit der Kurve aus Abbildung 6.36 kann daher die Position des Maximums des Berechnungsaufwands bei konstanter Atom- oder Regelanzahl bei unterschiedlichen durchschnittlichen Anzahlen von Literalen abgeschätzt werden. Die Skalierungsformel R/A auch auf Literale L bei casesMix zu erweitern, ist leider nicht mehr so einfach und kann mit den mir zur Verfügung stehenden Mitteln nur ansatzweise bewerkstelligt werden. Ihre Form müsste wohl $R/(A * f(L))$ entsprechen, wobei $f(L)$ etwas stärker wächst als $L * \log(L)$.

Abbildung 6.38 zeigt das Wachstum der durchschnittlichen choice points bei 150 Atomen und unterschiedlichen Anzahlen von Regeln. Die x-Achse steht hierbei für die durchschnittlichen Anzahlen von Körperliteralen. Wenn in Abbildung 6.36 statt der Anzahl der erfüllbaren Probleme die durchschnittlichen choice points dargestellt wären, wären die Kurven aus Abbildung 6.38 durch Schnitte parallel zur Literalachse entstanden. Zu sehen ist ein starkes Wachstum oberhalb von durchschnittlich einem choice point. Ob das Wachstum exponentiell ist, ist schwer zu sagen, da die Kurven alle mit der Zeit in den leichten Bereich kommen. Um dies festzustellen wäre eine Skalierungsfunktion hilfreich. Allerdings kann aufgrund des starken Wachstums ein exponentielles Wachstum mit Zunahme der Literale angenommen werden.

Das Wachstum des Berechnungsaufwands ist mit Zunahme der durchschnittlichen Anzahl von Literalen wesentlich stärker, als bei der Zunahme der Regeln und Atome bei konstanten R/A. Wenn also schwere Probleme bei gemischter Körperlänge generiert werden sollen, ist eine Steigerung der durchschnittlichen Anzahl von Literalen wohl der beste Weg dahin.

Da Programme des k-pLP Modells alle möglichen Körperlängen beinhalten

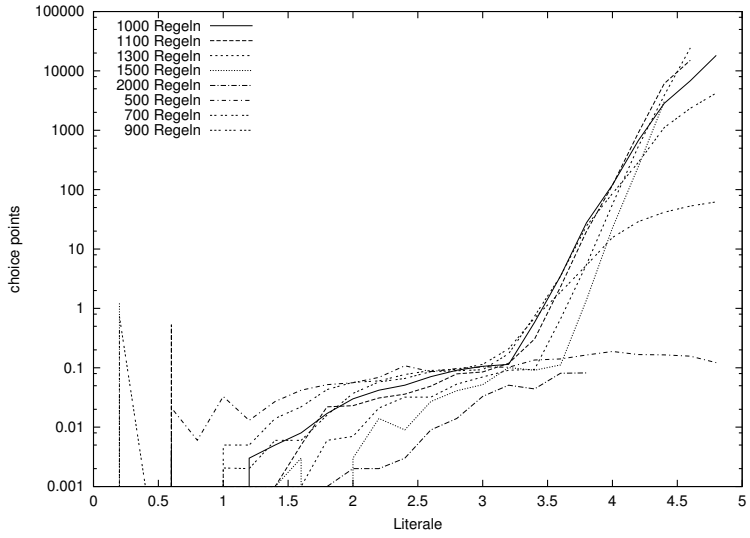

Abbildung 6.38: Durchschnittliche choice points bei unterschiedlichen vielen Regeln und $A = 150$

können, ist anzunehmen, dass die gefundenen Muster für alle Generierungsparameterbereiche bei k-pLP, nur in unterschiedlicher Ausprägung, zu finden sind. Dies heißt auch, dass das Wachstum bei konstanten R/A Faktor und unterschiedlichen Regel-, Atom- oder Literalanzahlen, unabhängig von den anderen Generierungsparametern, immer exponentiell ist. Bei geringer Anzahl von Atomen, Regeln oder durchschnittlicher Anzahl von Literalen ist es nur so gering, dass es nicht auffällt.

6.3.2 Zusammenhängende Programme

Die Testreihe casesCon wurde mithilfe des ersten Modells (jedes Atom kommt einmal im Körper vor) und casesConR mithilfe des zweiten Modells (jedes Atom kommt einmal im Kopf vor) des Abschnitts 5.1.3 auf Seite 41 generiert. Der verwendete Solver ist Smodels Version 2.28. Gesucht wurde jeweils eine Antwortmenge. Gerechnet wurde auf den Universitätsrechnern.

Die Generierungsmodelle der Testreihen sind dem Generierungsmodell der Testreihe casesMix sehr ähnlich. Der Hauptunterschied ist, dass alle Atome über Regeln miteinander verbunden sind. Alle Programme, die mithilfe der in casesCon und casesConR benutzten Generierungsmodelle generiert werden können, können auch mit dem *k*-pLP Generierungsmodell, das in casesMix benutzt wird, generiert werden, nicht aber umgekehrt. Bei casesCon und casesConR ist garantiert, dass jedes Atom, eventuell über weitere Atome, mit jedem anderem Atom zusammenhängt. Zwei Atome hängen zusammen, wenn eines vom anderen abhängt.

Leider ist mir bei der Erstellung der Generatoren ein Programmierfehler unterlaufen, so dass ein Atom mehr als Kopf möglich war, als es überhaupt Atome im Programm geben sollte. Dies sollte sich aber, bei der großen Anzahl von Atomen mit denen Programme generiert wurden, nicht wesentlich auswirken.

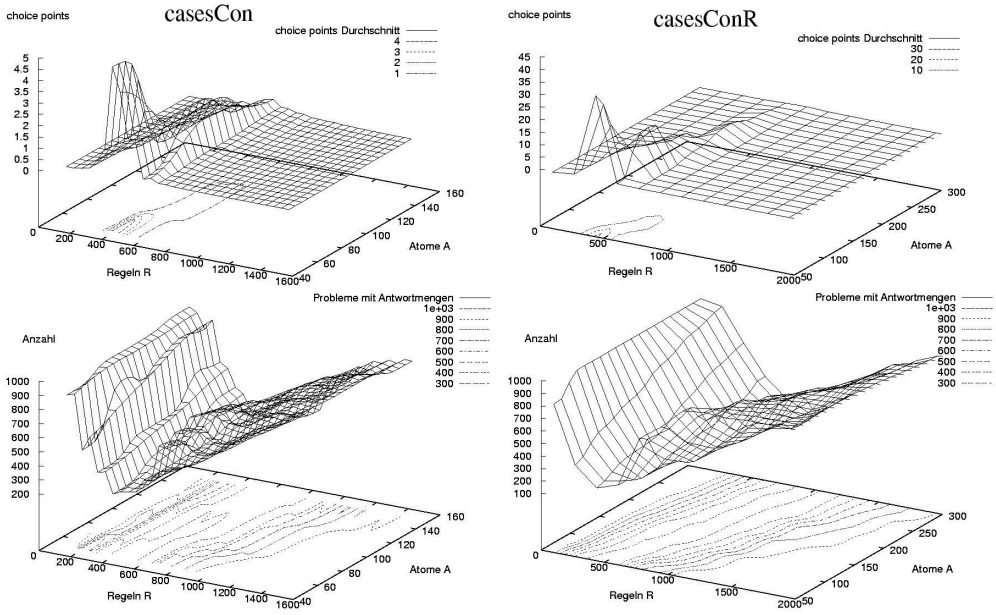

Abbildung 6.39: Durchschnittliche choice points und Erfüllbarkeit bei durchschnittlich 10 Körperliteralen für casesCon und casesConR

Auf Abbildung 6.39 zeigen die oberen Diagramme die durchschnittliche Anzahl von choice points und die unteren die Anzahl der erfüllbaren Probleme. Die Diagramme auf der linken Seite gehören zu casesCon und die auf der rechten Seite

zu casesConR. Alle Diagramme beziehen sich auf Probleme mit durchschnittlich 10 Literalen pro Körper. Das Erste, was auffällt ist, dass sowohl casesCon als auch casesConR wesentlich einfacher sind als casesMix. Während bei casesMix schon ab durchschnittlich 5 Literalen pro Körper bei 150 Atomen eine Testreihe für verschiedene Regelanzahlen sehr schwer auszuführen ist, ist es bei casesCon und casesConR selbst bei durchschnittlich 10 Literalen kein großes Problem. Denn bei allen drei Testreihen wächst das Berechnungsaufwandsmaximum, bei verschiedenen durchschnittlichen Anzahlen von Literalen in ähnlichen untersuchten Bereichen (gleiche maximale Regel- und Atomanzahl), stark mit der durchschnittlichen Anzahl von Literalen. An sich hatten die gemachten Testreihen casesCon und casesConR wohl zu kleine Werte als Obergrenze für die Atom-, die Regelanzahl und die durchschnittliche Anzahl von Literalen. Gute Grenzen sind allerdings nur schwer auszuloten, da durch das exponentielle Wachstum die Grenze zwischen den zu leichten und den zu schweren Problemen schmal ist.

Weiterhin fällt auf, dass die Höhenlinie, bei den unteren Erfüllbarkeitsdiagrammen zwar relativ gradenlinig sind, aber die Verlängerung der gradlinigen Abschnitte nicht mehr in der Nähe des Koordinatenursprungs verläuft. Die R/A Skalierungsformel hat bei casesCon und casesConR seine Aussagekraft verloren, da bei einem konstanten R/A Faktor der Anteil der erfüllbaren Probleme nicht einmal mehr annähernd konstant ist. Die Erfüllbarkeitskurven zeigen eine grabenähnliche Struktur, die sich, im Gegensatz zum Verhalten der Erfüllbarkeit bei casesMix (siehe Abbildung 6.34 Seite 78), nur allmählich verbreitert.

Die oberen Diagramme für die durchschnittlichen choice points fallen aus dem Bild, wie es sich bisher ergeben hat. Statt eines Höhenzuges der mit steigender Atom- und Regelanzahl exponentiell wächst, sind hier Höhenzüge zu sehen, die mit steigender Anzahl von Regeln und Atomen kleiner werden, wobei bei casesConR noch höhere Werte als bei casesCon erreicht werden, casesConR also schwieriger ist als casesCon. Der schwere Bereich liegt jeweils ungefähr in dem Bereich, in dem die Erfüllbarkeitskurve ihr Minimum hat.

Der Hauptunterschied zwischen casesMix (k-pLP) und casesCon sowie casesConR ist, dass alle Atome in den Programmen immer zusammenhängen. Dadurch ist es wahrscheinlicher, dass aus vorhandenen Informationen auf den Wert eines weiteren Atoms geschlossen werden kann. Der Anteil der erfüllbaren Probleme müsste sich also erhöhen. Weiterhin bewirkt ein höherer Anteil von Fakten stärker als bei casesMix, dass die Programme erfüllbar sind, da eher Regeln vorhanden sind um sie zu nutzen, beziehungsweise in denen die dazugehörigen Atome im Körper vorkommen. Dadurch dürften, im Gegensatz zu casesMix, Probleme häufiger erfüllbar sein und die Erfüllbarkeitskurve steigt mit Zunahme der Regel früher an.

Die Anzahl der Zusammenhänge und deren Komplexität ist allerdings bei casesMix und casesCon sowie casesConR ungefähr gleich. Bei casesCon sowie casesConR wird nur dafür gesorgt, dass ein Teil der Regeln einen baumartigen Zusammenhang der Atome realisiert. Um bei den Abhängigkeitsgraphen von casesMix k-pLP Programmen einen Zusammenhang wie bei casesCon sowie casesConR

zu realisieren, werden nur ein paar zusätzliche Verbindungen benötigt (maximal eine Verbindungen weniger als Atome vorhanden sind).

Die choice point Diagramme können so erklärt werden, dass der rechte Teil, bei mehr Regeln, der choice point Kurve bei casesMix sozusagen abgeschnitten wird. So dass die coice point Kurven bei casesCon und casesConR nur eine Art linken Teil der casesMix Kurve zeigen. Dieser Schnitt wird dadurch bewirkt, dass sich die Informationen, durch den Zusammenhang der Atome, stärker auswirken und sich die Zahl der choice points, gerade bei mehr Atomen und Regeln, stärker verringert.

So wie der Zusammenhang der Atome in den Programmen bei casesCon und casesConR realisiert wird, werden die Programme, durch die zusätzliche Information die er liefert, also eher leichter lösbar als bei casesMix. Auch bei casesMix dürften allerdings die Atome in den meisten Programmen zusammenhängen, dabei hängen alle Atome um so eher zusammen, um so mehr Regeln im Verhältnis zu den Atomen ein Programm hat. Ein Generierungsmodell, das den Zusammenhang der Atome auf andere Weise bewerkstelligt, könnte die Programme auch im Verhältnis zu casesMix schwieriger machen.

Das Verhalten der Berechnungsaufwandskurven bei casesCon und casesConR erschwert natürlich das Finden von schweren Bereichen.

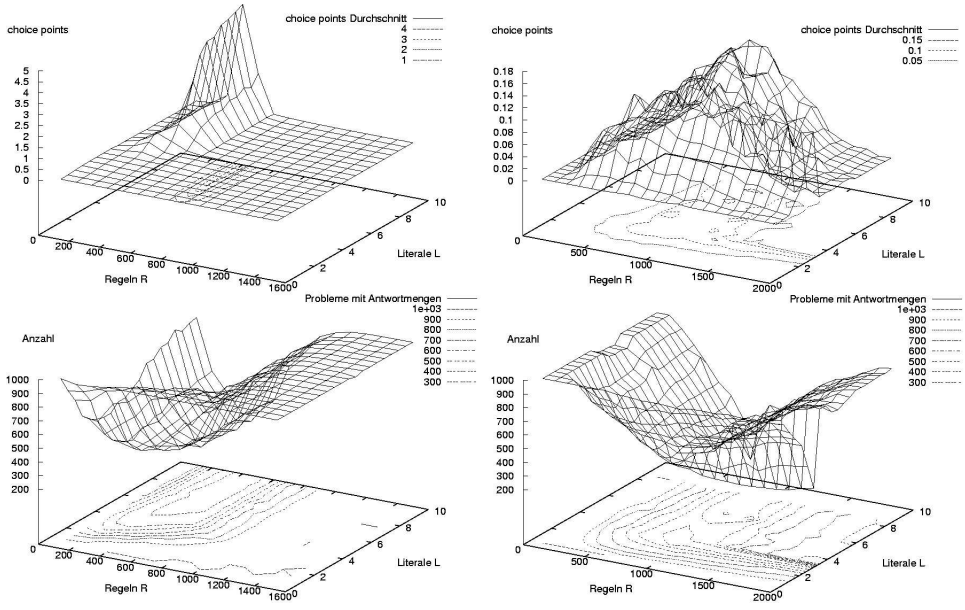

Abbildung 6.40: Durchschnittliche choice points und Erfüllbarkeit bei casesCon für 50 und 300 Atome

Die in Abbildung 6.40 gezeigten Diagramme gehören zur Testreihe casesCon. Sie sind vertikal genauso wie in Abbildung 6.39 geordnet, allerdings befinden sich jetzt links die Diagramme für 50 Atome und rechts die für 300 Atome.

In den beiden oberen choice point Diagrammen ist wieder der mit der Literalanzahl wachsende Höhenzug zu sehen. Wobei er links bei 50 Atomen noch wesentlich höher und ausgeprägter ist, als rechts bei 300 Atomen. Für ein besseres Verständnis der Zusammenhänge, müssen wohl die durchschnittlichen choice points zusammen mit der Atom-, der Regelanzahl und der durchschnittlichen Anzahl von Literalen gesehen werden, was allerdings schwierig ist. Die Höhenlinien in einer solchen Darstellung sind wahrscheinlich kegelartige Gebilde, die in Richtung der Literaleachse offen sind.

Die unteren Erfüllbarkeitsdiagramme zeigen wieder grabenartige Strukturen, die, von Anfangswerten bei wenigen Literalen abgesehen, fast parallel zur Literalachse verlaufen. Es ist einsichtig, dass bei sehr wenigen Literalen mehr Programme erfüllbar sind, da bei ihnen auch ein sehr hoher Faktenanteil existiert. Die Tiefe der Gräben ist ungefähr gleich. Der Graben für 300 Atome ist weniger ausgeprägt und nach rechts, in den Bereich von mehr Regeln, verschoben.

Desweiteren ist beim Diagramm für 300 Atome, beim Graben ein Seitenarm zwischen 2 bis 4 Literalen zu sehen, der sich in Richtung der Regelachse allmählich abflacht. Ein entsprechender Seitenhöhenzug ist darüber im zugehörigen choice point Diagramm für 300 Atome zu sehen. Die Erfüllbarkeit ist also mit dem Berechnungsaufwand gekoppelt, da bei beiden Atomanzahlen die Probleme um so schwieriger sind, um so geringer der Anteil der erfüllbaren Probleme ist.

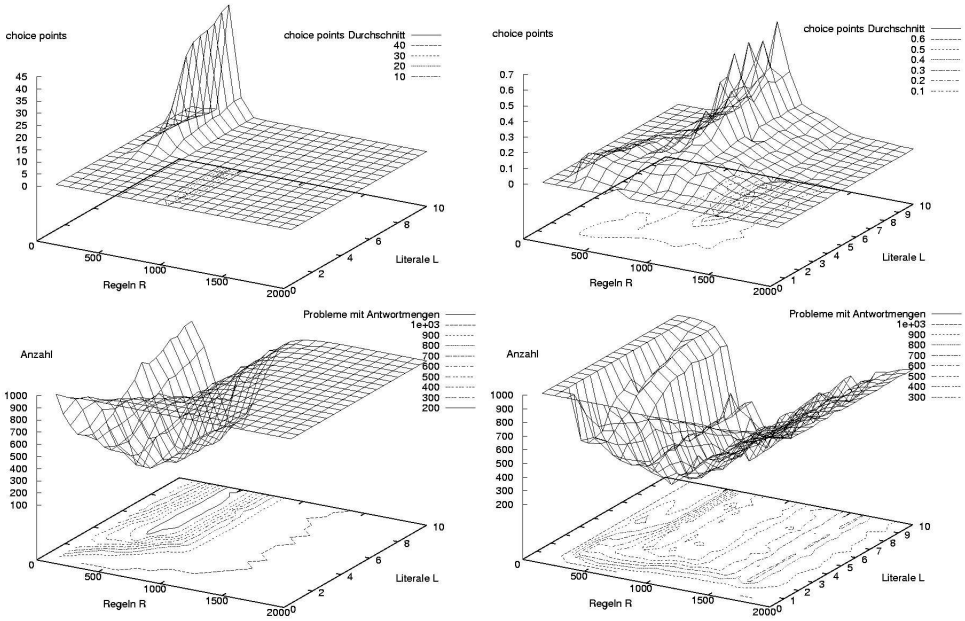

Abbildung 6.41: Durchschnittliche choice points und Erfüllbarkeit bei casesConR für 50 und 300 Atome

Die Aufteilung in Abbildung 6.41 entspricht der aus Abbildung 6.40, links be-

finden sich die Diagramme für 50 Atome und rechts die für 300 Atome. In beiden Abbildungen wurde die linke Seite jeweils so gewählt, dass es beim Berechnungsaufwand einen deutlichen Ausschlag gibt. Bei casesConR sind die Maximalausschläge des Berechnungsaufwands höher als bei casesCon. Dies könnte daran liegen, dass bei casesCon, im Gegensatz zu casesConR, Atome auch in keinem Kopf vorkommen können, wodurch sie von vornherein mit Falsch belegt werden. Dadurch werden schon am Anfang zusätzliche Informationen bereitgestellt und die Suche wird einfacher.

Ansonsten ähnelt das Verhalten von casesConR sehr dem von casesCon.

6.4 Hamiltonsche Zyklen

Bei der Testreihe graph1, wurden die Programme gemäß Abschnitt 5.2 auf Seite 42 generiert. Hier wurde also nach Hamiltonschen Zyklen in Graphen gesucht. Gerechnet wurde auf Universitätsrechnern und gesucht wurde eine Antwortmenge.

Bei den nachfolgenden dreidimensionalen Diagrammen, die eine Kurvenfläche enthalten, gibt es am oberen linken Rand, in dem Bereich mit wenigen Knoten aber mehr Kanten, auch eine Fläche, obwohl dort keine Daten vorhanden sind. Bei N Knoten (nodes) ist die maximale Anzahl der Kanten E (edges) gleich $E = N^2$, also alle möglichen Paare von Knoten. Diese Unstimmigkeit wird von mir in Kauf genommen, da Kurvenflächen anschaulicher sind und Gnuplot sie dann so generiert.

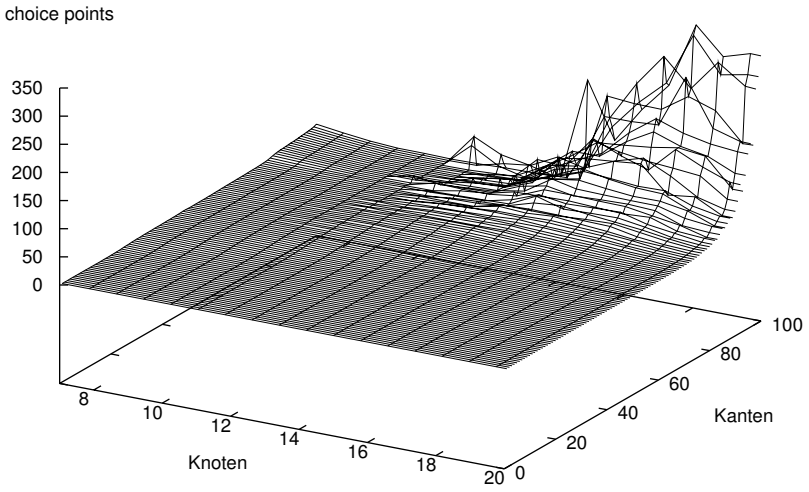

Abbildung 6.42: Durchschnittliche choice points bei graph1

Der Höhenzug aus Abbildung 6.42 für die durchschnittlichen choice points, weist Ähnlichkeiten mit den bisher behandelten Höhenzügen für die choice points von cases und casesMix auf, nur zeigt diesmal die x-Achse die Anzahl der Knoten und die y-Achse die Anzahl der Kanten. Auch zeigt der Höhenzug mehr Unebenheiten auf, als die vorangegangenen.

Die Abbildung 6.43 zeigt für die unterschiedlichen Testpunkte, die Anzahl der Probleme die erfüllbar waren, beziehungsweise die Anzahl der Graphen die mindestens einen Hamiltonschen Zyklus besaßen. Wie zu erwarten, steigt die Wahr-

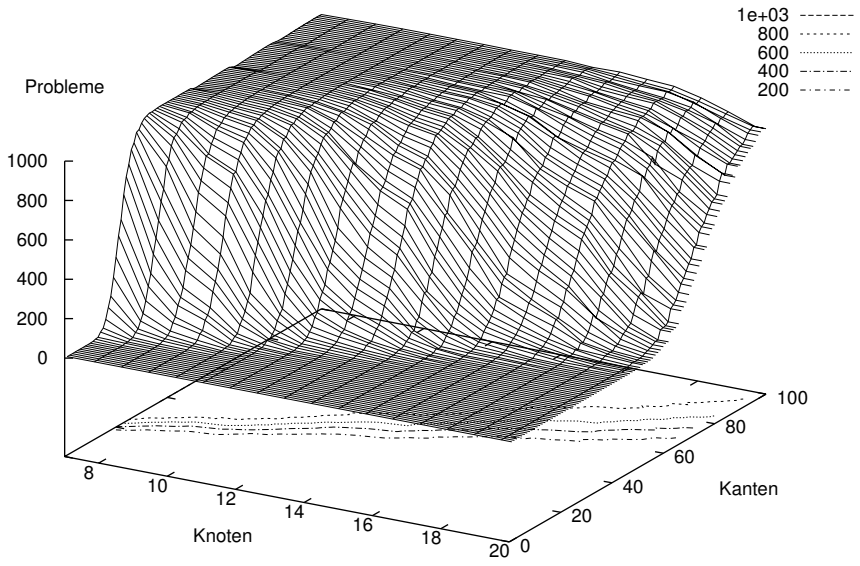

Abbildung 6.43: Erfüllbare Probleme bei graph1

scheinlichkeit, dass Graphen einen Hamiltonschen Zyklus haben, ab einen bestimmten Kanten/Knoten (E/N) Verhältnis plötzlich stark an, es gibt also einen Phasenübergang. Allerdings sind die Höhenlinien nahezu gradlinig, was auf eine Skalierungsformel ungleich $\frac{E}{N \log N}$ schließen lässt, wie sie in Abschnitt 3.1 Seite 16 beschrieben ist. Aus den erhobenen Daten kann geschlossen werden, dass die Skalierungsformel irgendwo zwischen $\frac{E}{N \log N}$ und $\frac{E}{N}$ liegt. Für die Formeln bewegt sich die 50 Prozent Marke der Erfüllbarkeit in unterschiedliche Richtungen, wenn die Knotenzahl erhöht wird. Der Grund dafür ist die Generierungsart der Graphen, denn die $\frac{E}{N \log N}$ Skalierungsformel gilt für ungerichtete Graphen, hier werden aber gerichtete Graphen verwendet.

Abbildung 6.44 zeigt die verschiedenen Parameter der Programme. All diese Parameter zeigten innerhalb der Testpunkte nur wenig Varianzen, was an der Art der Kodierung liegt.

Links oben ist die durchschnittliche Anzahl der Atome der Programme aufgeführt. Zu sehen ist, dass sie linear mit der Anzahl der Kanten und Knoten wächst. Ein ähnliches Bild ergibt sich für die Anzahl der Regeln im Diagramm links unten, nur ist deren Wachstum etwas höher. Daraus resultiert die Kurve rechts unten für der R/A Faktoren. Ihr Verhalten ist nicht linear.

Da die Programme nicht nur normale Regeln enthalten, ist die durchschnittliche Anzahl von Körperliteralen nicht (ohne weiteres) messbar. Um dennoch einen

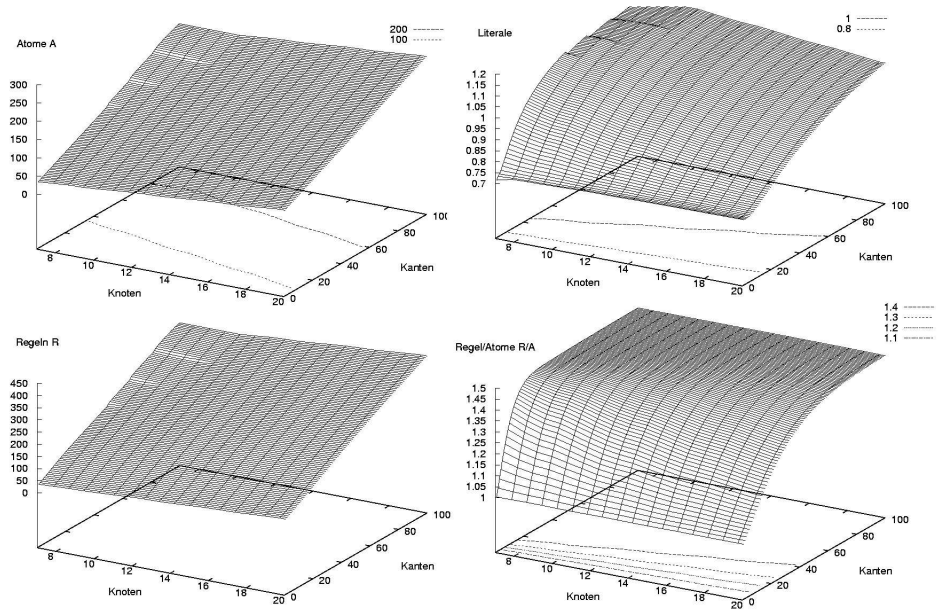

Abbildung 6.44: Durchschnittliche Atome A, Anzahl von Literalen L pro Regelkörper, Regeln R und dem Verhältniss von R/A bei graph1

aussagekräftigen Wert zu erhalten, wurden nur die Körperliterale der normalen Regeln gezählt und deren Gesammtanzahl durch die Anzahl der normalen Regeln geteilt. Das Wachstum der durchschnittlichen Anzahl von Körperliteralen pro Regelkörper im Diagramm rechts oben, ist auch nicht linear. Sie wächst mit der Anzahl der Kanten, aber sinkt schwach mit der Anzahl der Knoten. Die linearen Höhenlinien, die in Richtung Koordinatenursprung verlaufen, lassen auf eine Skallierungsformel $E/N = Kanten/Knoten$ schließen.

Aufgrund der Veränderung der durchschnittlichen Anzahl von Literalen, lässt sich die Testreihe graph1 nur schwer mit den zuvor beschriebenen Testreihen, z.B. casesMix, vergleichen.

Wenn die Werte für die durchschnittliche Anzahl von Literalen und R/A von graph1 auf casesMix bezogen werden, befinden sie sich im leichten Bereich, denn beide Parameter nehmen nur geringe Werte an.

In Abbildung 6.45 sind einige choice point Kurven und die Kurve der erfüllbaren Probleme für Graphen mit 20 Knoten gegenüber gestellt.

Das Diagramm oben links zeigt die durchschnittlichen choice points und die Anzahl der erfüllbaren Probleme. Die Achse für die Erfüllbarkeit ist links, die Achse für die choice points rechts, diese ist auch logarithmisch unterteilt. Während der Phasenübergang bei der Erfüllbarkeit ungefähr zwischen 50 und 100 Kanten liegt, liegt das Maximum der choice points ungefähr bei 180 Kanten. Das Wachstum

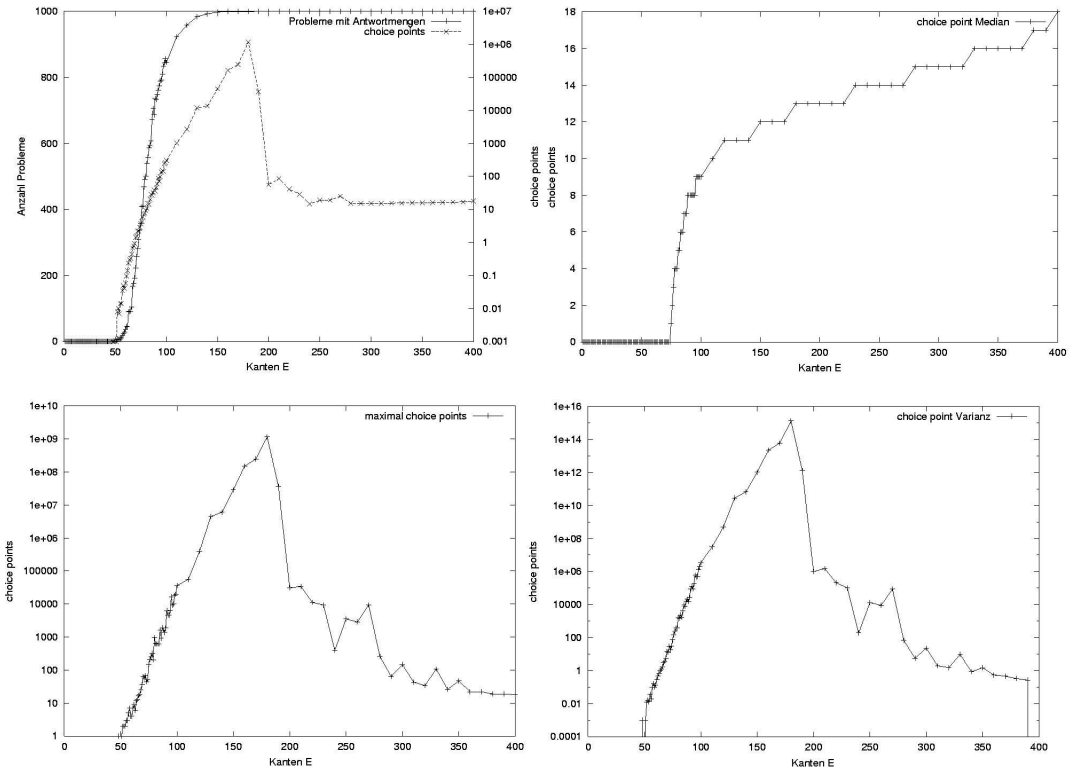

Abbildung 6.45: Durchschnittliche, median, maximale choice points und choice point Varianz bei graph1

der choice point Kurve beginnt mit dem Phasenübergang der Erfüllbarkeitskurve. Nachdem die choice point Kurve das Maximum durchlaufen hat, sinkt sie relativ schnell von einem Wert von rund 10^6, auf einen Wert knapp über 10 durchschnittliche choice points.

Das Diagramm rechts oben zeigt den Median der choice points. Zu beachten ist, dass dies das Einzige der vier Diagramme aus 6.45 ist, bei dem die choice point Skala nicht logarithmisch unterteilt ist. Nach einem relativ starken Wachstum der Kurve, in der Nähe des Phasenübergangs der Erfüllbarkeit, bei rund 60 bis 150 Kanten, geht die Kurve in einen langsameren Anstieg über, der relativ linear ist. Dabei erreicht sie am Ende gerade mal 18 choice points. Ab ungefähr 250 Kanten sind die Werte für die durchschnittlichen und die median choice points ungefähr gleich. Davor fehlt der median choice point Kurve aber völlig eine schwere Phase, beziehungsweise ein lokales Maximum.

Der Ursprung des Maximums der Kurve für durchschnittliche choice points, wird bei der Betrachtung der Kurve links unten für die maximalen choice points ersichtlich. Sie zeigt annähernd den Kurvenverlauf der durchschnittlichen choice

points. Das Maximum der Kurve für die maximalen choice points liegt ungefähr bei 10^9, also rund um den Faktor 1000 höher, als das Maximum der durchschnittlichen choice points. Bei 1000 generierten Problemen für einen Testpunkt, ergeben sich die durchschnittlichen choice points im schweren Bereich fast ausschließlich aus einigen wenigen sehr schweren Problemen.

Diese Tatsache wird auch von der, im Diagramm rechts unten dargestellten, choice point Varianz bestätigt, die innerhalb des schweren Bereiches sehr hoch ist.

Ob der Median der choice points, bei Problemen aus der Testreihe graph1, auch ein anderes Verhalten zeigen kann, dürfte schwer zu ermitteln sein, da bei mehr Knoten der schwere Bereich so gut wie nicht mehr zugänglich ist.

Das gezeigte Verhalten der choice points lässt darauf schließen, dass die choice point Verteilung, innerhalb eines Testpunktes im schweren Bereich, sehr asymmetrisch ist. Es gibt wahrscheinlich viele Probleme mit relativ wenigen choice points, so dass dort ein starkes Maximum oder eine starke Ansammlung auftritt. Dann gibt es noch einige wenige Probleme mit sehr vielen choice points, die den Durchschnitt der choice points dominieren.

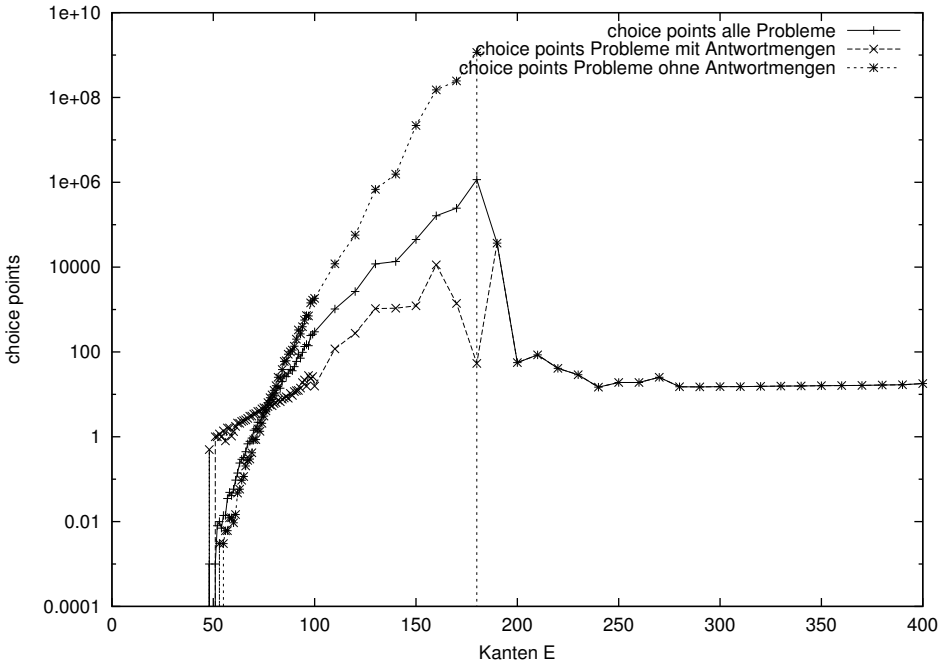

Abbildung 6.46: Durchschnittliche choice points bei erfüllbaren, unerfüllbaren und allen Graphen

Woher die sehr schweren Probleme kommen, wird aus Abbildung 6.46 ersichtlich. In ihr sind die durchschnittlichen choice points für die erfüllbaren, die unerfüllbaren und alle Probleme dargestellt. Die Kurve für die erfüllbaren Probleme

fängt erst bei rund 50 Kanten an und die der unerfüllbaren hört bei rund 180 Kanten auf, da außerhalb dieser Bereiche keine erfüllbaren beziehungsweise unerfüllbaren Probleme vorhanden waren. Deutlich zu sehen ist, dass im schweren Bereich die Kurve der erfüllbaren Probleme deutlich unter der von den unerfüllbaren liegt. Die durchschnittlichen choice points werden also von den wenigen unerfüllbaren Problemen dominiert. Mit steigender Kantenanzahl wird es demnach immer schwerer zu beweisen, dass ein Graph keinen Hamiltonschen Zyklus besitzt. Im schweren Bereich hat auch die Kurve der erfüllbaren Probleme einen Ausschlag nach oben, nur ist dieser deutlich schwächer, als der Ausschlag bei den unerfüllbaren Problemen. Die Kurve der unerfüllbaren Probleme hat einen sehr starken Anstieg, der nur abbricht, weil für die Testpunkte keine unerfüllbaren Probleme mehr vorhanden waren. Interessant wäre noch herauszufinden, wie sich diese Kurve fortsetzt, indem für die Testpunkte mehr Probleme generiert werden. Auch das Maximum der durchschnittlichen choice points aller Probleme könnte dadurch weiter nach rechts, in den Bereich von mehr Kanten, verschoben werden.

Insgesamt zeigt das Verhalten der Testreihe graph1 deutliche Unterschiede zu dem Verhalten von den anderen hier untersuchten Testreihen für zufällig generierte Programme. Es gibt aber auch viele Parallelen, wie z.B. das leicht-schwer-leicht Muster beim Berechnungsaufwand oder die starke Veränderung des Anteils der erfüllbaren Probleme in einem Bereich (bei graph1 beim Phasenübergang).

6.5 Vergleich von verschiedenen Solvern

Im nachfolgenden wird der Nomore++ und der Smodels-Solver verglichen. Beim Vergleich von unterschiedlichen Solvern ist dabei Folgendes zu beachten.

Die Höhe der Kurvenwerte ist aussagekräftig über den tatsächlichen benötigten Aufwand an choices oder Zeit der Solver, sie kann aber keine Aussagen über die Güte der zugrunde liegenden Ideen treffen. Da gute Ideen bei einem Solver in einer schlechten Implementation umgesetzt worden sein können und andersherum. Der hier verwendete Nomore++ Solver verfolgt beispielsweise einen relativ neuen Ansatz, bei der verwendeten Version handelt es sich wohl um eine der ersten überhaupt lauffähigen. Dem gegenüber ist die verwendete Smodels-Version schon wesentlich ausgereifter. Es ist sehr wahrscheinlich, dass Nomore++ allein durch ein paar Implementationsänderungen verbessert werden kann, ohne dabei die grundlegende Arbeitsweise zu ändern.

Deshalb können wohl allein mit den choices oder den Zeiten, die ein Solver für ein Beispiel oder auch einen Testpunkt benötigt, keine wirklichen Aussagen darüber getroffen werden, wie gut der verwendete Ansatz ist. Denn schlechtere Zeiten können allein durch eine langsamere Implementierung entstehen und mit choices allein kann noch keine Aussage über die benötigte Berechnungszeit getroffen werden.

Der Anstieg einer geeigneten Kurve (z.B. bei konstanten R/A Faktor) ist eher ein Indiz für die Güte der zugrunde liegenden Ideen bei einem Solver, da er eher

das Wachstum des Berechnungsaufwands darstellt. Mit ihm lassen sich auch Abschätzungen für Beispiele mit anderen Parametern machen.

Da der Nomore++ und der Smodels-Solver ähnliche Ansätze verfolgen, sollte das Berechnungsaufwandsverhalten beider ähnlich sein.

6.5.1 Vergleich von Smodels und Nomore++

Die Testreihe casesN entspricht der Testreihe cases (k-LP) aus dem Abschnitt 6.2.2 Seite 58 mit dem Unterschied, dass als ASP-Solver Nomore++ Version 0.4.10 verwendet wurde. Gerechnet wurde auf meinen Heimrechner und gesucht wurden alle Antwortmengen. Wegen der großen Ähnlichkeit des Verhaltens der Testreihen cases und casesN, wird casesN hier im Vergleich zu cases betrachtet.

Da sowohl die Suche von Smodels als auch Nomore++ vollständig sind, sind die Kurven der Testreihen für die Anzahl der erfüllbaren und damit auch unerfüllbaren Probleme, von statistischen Ungenauigkeiten abgesehen, gleich.

Im Allgemeinen ähnelt sich das Verhalten der Kurven für cases und casesN sehr, weshalb hier vor allem die Unterschiede betrachtet werden.

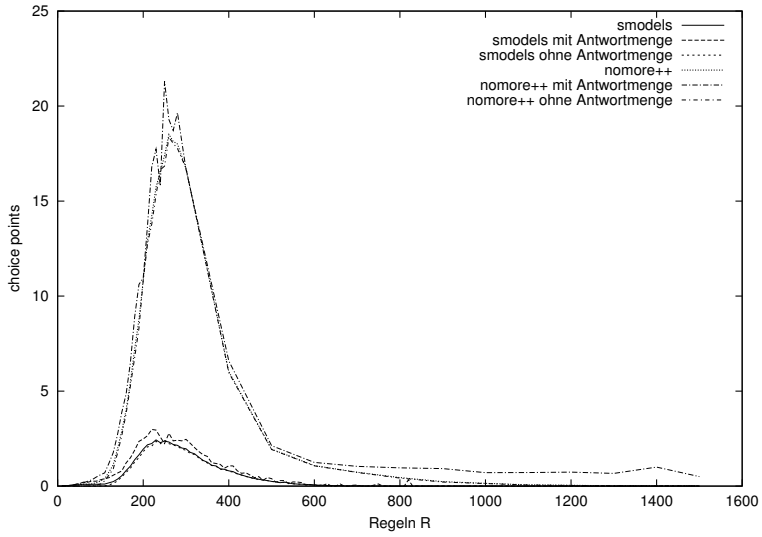

Abbildung 6.47: Durchschnittliche choice points bei erfüllbaren, unerfüllbaren und allen Problemen für cases und casesN bei 50 Atomen

Abbildung 6.47 zeigt die durchschnittlichen choice points der erfüllbaren, unerfüllbaren und aller Probleme für die Testreihe cases mit Smodels und casesN mit Nomore++ bei 2-LP und 50 Atomen. Der Verlauf der Kurven ist unabhängig vom verwendeten Solver, nur die Höhe der konkreten Werte unterscheidet sich. So ist die Anzahl der durchschnittlichen choice points für Nomore++ durchweg höher als für Smodels.

Abbildung 6.48 zeigt das Wachstum der durchschnittlichen choice points bei

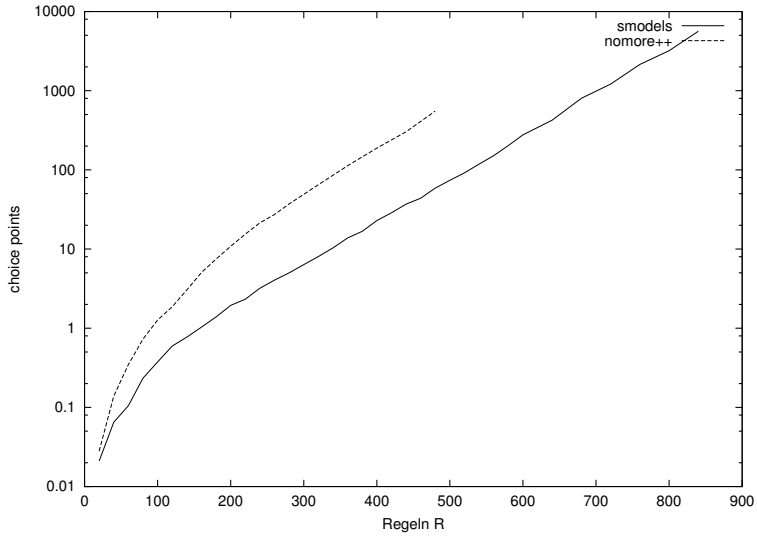

Abbildung 6.48: Durchschnittliche choice points für cases und casesN bei $R/A = 4$

2-LP für den Faktor $R/A = 4$. Die Kurven für Smodels und Nomore++ sind vom Verlauf her sehr ähnlich. Die Werte der Nomore++ Kurve sind aber überall wesentlich höher, als die Werte der Smodels Kurve.

Leider sind die Werte von Nomore++ schon sehr früh so hoch, dass eine weitere Berechnung des Kurvenverlaufs zu aufwändig war und abgebrochen werden musste. Deshalb sind keine sicheren Aussagen über die stabile Phase möglich.

Es sieht aber so aus, als ob die Kurve von Nomore++ bei höheren Werten, in der stabilen Phase, annähernd parallel zu der von Smodels verläuft. Zu beachten ist, dass in einer exponentiellen Darstellung ein konstanter Abstand bedeutet, dass der Quotient der Werte ein konstanter Faktor ist. Dieser Faktor ist hier rund 10, dass heißt Nomore++ braucht durchschnittlich für Probleme ungefähr 10 mal mehr choice points als Smodels.

Weitergehende Untersuchungen wären auf jeden Fall interessant.

Abbildung 6.49 zeigt die durchschnittliche Zeit bei den Testreihen cases und casesN für 2-LP beim Faktor $R/A = 4$. Hier schneidet Nomore++ noch schlechter als bei den choice points ab. Nur bei sehr kleinen Werten liegt Nomore++ kurzzeitig vorn. Die Endwerte der Nomore++ Kurve liegen bei rund 140 Sekunden, was bei 1000 Problemen pro Testpunkt rund 39 Stunden Laufzeit für den Testpunkt bedeutet.

Allerdings beschreibt die Kurve von Smodels einen nach oben offenen und die von Nomore++ einen nach unten offenen Bogen. Beide Kurven enden ungefähr parallel, also mit ähnlichem exponentiellem Wachstum (bei der Approximation 2^{x*a+b} wäre der Faktor a für beide ähnlich). Es ist also durchaus möglich, dass sich Nomore++ beim Wachstum gegenüber Smodels noch verbessert. Wenn das Wachstum bei Nomore++ irgendwann geringer als bei Smodels wird, werden die

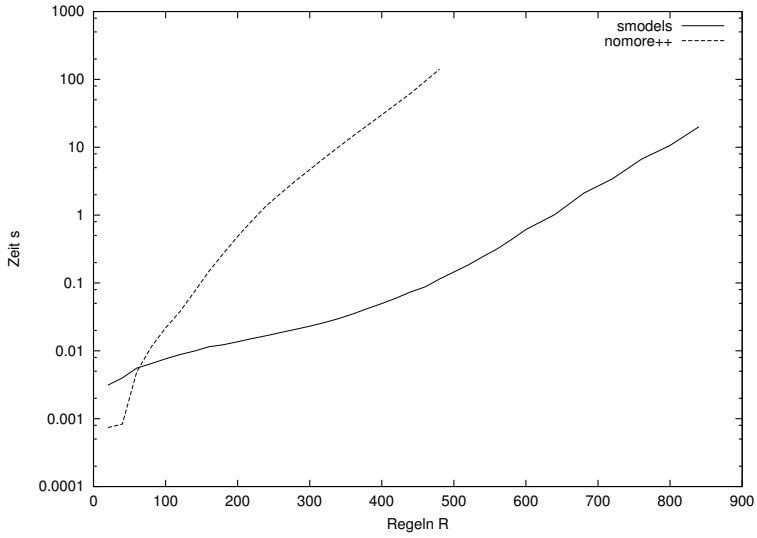

Abbildung 6.49: Durchschnittliche Zeit für cases und casesN bei $R/A = 4$

Werte bei Nomore++ auch irgendwann kleiner, als die von Smodels sein. Da es sich bei den Problemen um noch relativ kleine Probleme, im Sekunden- bis Minutenbereich der Berechnungszeit handelt, ist es durchaus möglich, dass Nomore++ bei Problemen im Stundenbereich gegenüber Smodels nicht mehr ganz so schlecht abschneidet.

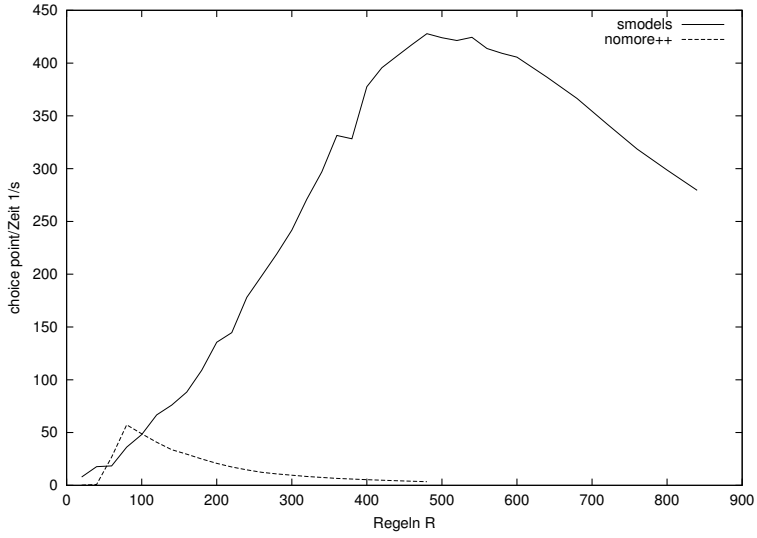

Abbildung 6.50: Durchschnittlicher choice points Zeit Quotient für cases und casesN bei $R/A = 4$

Das unterschiedliche Verhältnis von Zeit zu choice points wird auch aus Abbildung 6.50 ersichtlich. Sie zeigt den choice point (/) Zeit Quotient für cases und casesN bei 2-LP und dem Faktor $R/A = 4$. Die Nomore++ Kurve scheint eine kleinere Version der Smodels Kurve zu sein. Dies erklärt, dass sich die durchschnittliche Zeit bei Nomore++ noch schlechter verhält, als die durchschnittlichen choice points, da bei Nomore++ auf einen choice point durchschnittlich viel mehr Zeit kommt.

Es ist also sinnvoll, die Zeit und die choice points für einen Solver zueinander in Beziehung zu bringen. Denn die choice points allein helfen bei der Frage, wie lange die Lösung eines Problems dauert, nicht weiter. Außerdem können so weitere Zusammenhänge erkannt und für die Verbesserung der Solver genutzt werden.

Die dargestellten Zusammenhänge zeigen, dass es durchaus Unterschiede zwischen den Solvern beim Wachstum des Berechnungsaufwands gibt, die aus dem leicht-schwer-leicht Muster nicht ersichtlich werden.

Kapitel 7

Abschluss

In der vorliegenden Arbeit wurde das Berechnungsaufwands- und Erfüllbarkeitsverhalten von Antwortmengensolvern untersucht.

Dabei sind die verschiedenen Problemstellungen bei der Untersuchung von Suchstrategien und Suchstrategieimplementationen zu berücksichtigen. Theoretische Untersuchungen können nur auf einfachen Annahmen beruhen, da sie sonst viel zu komplex werden, um noch handhabbar, verständlich und aussagekräftig zu sein. Praktische Untersuchungen sind dagegen immer auf einige Problembereiche beschränkt, so dass die Aussagekraft dieser im Hinblick auf das allgemeine Verhalten ungewiss ist.

Ich habe mich aus beiden Richtungen der Problemstellung genähert und dabei auch neue Herangehensweisen entwickelt. Einige dieser neuen Herangehensweisen haben auch durchaus neue Aspekte aufgezeigt, z.B. die Untersuchung des Berechnungsaufwands bei einem konstanten Wert für die problemspezifische Skalierungsformel.

7.1 Zusammenfassung

Zunächst wurde im Kapitel 2 eine Einführung in das Gebiet der Antwortmengenprogrammierung gegeben. Bei den dort vorgestellten Definitionen von Operatoren und Suchschematas, wurden schon erste Überlegungen bezüglich des Berechnungsaufwands angestellt.

In Kapitel 3 wurden einige zum Thema relevanten Arbeiten zusammengefasst. Die vorgestellten Arbeiten über Graphen- und SAT-Probleme zeigten ähnliche Muster dahingehend, dass es bei den untersuchten Problemen einen Phasenübergang bei der Erfüllbarkeit gibt, zwischen der Phase, in der die meisten Probleminstanzen erfüllbar sind, und der Phase, in der die meisten Probleminstanzen unerfüllbar sind. Zu den ähnlichen Mustern gehört auch, dass es beim Berechnungsaufwand ein leicht-schwer-leicht Muster gibt und in der Nähe des Phasenübergangs bei der Erfüllbarkeit der schwere Bereich zu finden ist.

Weiterhin wurden auch Arbeiten über den Berechnungsaufwand bei der Antwortmengenprogrammierung betrachtet. Sie unterschieden sich von den Graphen- und SAT-Untersuchungen dahingehend, dass kein Phasenübergang mehr vorhanden ist, da es nur eine Phase von größtenteils unerfüllbaren Problemen gibt. Das leicht-schwer-leicht Muster ist aber auch bei ihnen zu finden.

In Kapitel 4 wurden einfache theoretische Modelle vorgestellt, um damit die Erklärung des Verhaltens bei den Testreihen zu erleichtern.

Danach spielen die Informationen, wie Vorwissen, abgeleitetes Wissen oder Problemwissen, die zur Lösung eines Problems zur Verfügung stehen, eine zentrale Rolle. Einerseits können mehr Informationen den Suchvorgang vereinfachen, andererseits wird, um mehr Informationen zu berücksichtigen und zu finden, mehr Zeit benötigt. Daher ist es wahrscheinlich nicht möglich einen Antwortmengensolver zu bauen, der in allen Problemklassen der Beste ist, da für verschiedene Problemklassen verschiedene Arten von Informationen unterschiedlich nützlich sind. Es ist daher vorteilhaft, sich bei der Wahl des Solvers an der Problemklasse zu orientieren und bei speziellen Problemen auch spezielle Suchalgorithmen zu verwenden.

In Kapitel 6 beinhaltet die Ergebnisse der gemachten statistischen Untersuchungen. Die dafür benötigten Generierungsmodelle sind in Kapitel 5 zusammengefasst.

Bei allen untersuchten Testreihen gibt es wiederkehrende Muster, in fast allen Bereichen. Allerdings treten diese Muster erst bei größeren Werten, beziehungsweise Problemen, auf, bei kleineren Werten gehen sie oft im statistischen Rauschen unter.

Eines dieser Muster ist, dass es Skalierungsformeln gibt, z.B. R/A, bezüglich derer das Wachstumsverhalten der Kurven ähnlich ist. Diese Skalierungsformeln können für die Bestimmung des Berechnungsaufwandswachstums genutzt werden. Wenn der Wert der Skalierungsformel (der Faktor) konstant gehalten wird, zeigte sich meist ein monotones Wachstum. Bei den Testreihen mit fester Körperlänge (k-LP) und gemischter Körperlänge (k-pLP) ist die Skalierungsformeln R/A, dabei zeigt sich, bei konstanter Anzahl von Literalen und bei $k - LP$ für $k > 1$ und einem konstantem Wert größer 1 für die Skalierungsformel ($R/A > 1$), ein ähnliches, annähernd exponentielles Wachstumsverhalten. Das exponentielle Wachstum erschwert allerdings eine genaue Untersuchung, da es dadurch sehr schnell zu aufwändig wird weitere Testpunkte zu generieren und es nur schmale aussagekräftige Bereiche gibt, die nicht immer leicht zu finden sind. Um verlässliche Aussagen zu treffen, ist es zwingend notwendig die Programme so zu generieren, dass das Wachstum, wenn vorhanden, von der exponentiellen Komponente dominiert wird und nicht von linearen oder polynomiellen Komponenten.

Die statistischen Nennwerte für die choice points, die wrong choices, die Wahrheitswertzuweisungen und der Zeit zeigen alle ein ähnliches Wachstumsverhalten, auch wenn sich das konkrete Wachstum unterscheidet. Dieser Unterschied verändert sich im allgemeinen geringer als die konkreten Werte. Danach ist es nicht mehr

so wichtig, welcher Parameter als Indikator für das Berechnungsaufwandswachstum genommen wird. Das Verhalten des Unterschieds (z.B. des choice point (/) Zeit Quotients) kann allerdings dennoch für die Untersuchung des Solvers nützlich sein, z.B. beim Unterschied von choice point Wachstum zum Zeitwachstum.

Wenn es um den Vergleich des konkreten Berechnungsaufwands geht, ist es unerlässlich auch die Zeit zu untersuchen. Dies kann auch geschehen, indem sie zu anderen Parametern, z.B. choice points, in Relation gesetzt wird. Wie der gemachte Vergleich zwischen Nomore++ und Smodels zeigt, können sich beispielsweise die durchschnittlichen choice points um einen deutlichen anderen Faktor unterscheiden als die durchschnittlichen Zeiten. Das choice point zu Zeit Verhältnis hängt sogar nicht nur von den Solvern ab, sondern variiert auch bei Programmen mit unterschiedlichen Parametern für einen Solver.

Weiterhin wird durch kleinere Unterschiede bei den Modellen zur Generierung der Programme das Ergebnis nicht sehr beeinflusst. Bei Untersuchungen ist es demnach durchaus zulässig, Modelle zur Generierung zu wählen, mit denen sich mit weniger Aufwand Programme generieren lassen.

Die großen Ähnlichkeiten bei verschiedenen Generierungsmodellen, Berechnungsaufwandsparametern und Solvern eröffnet eine Möglichkeit unterschiedliche Solver auf allgemeinerer Basis zu vergleichen. Dafür könnte beispielsweise beim 2-LP Generierungsmodell die choice point Kurven und Zeitkurven der Solver, für den konstanten Faktor $R/A = 4$, verglichen werden.

Die gemachten Untersuchungen deuten darauf hin, dass es ein paar allgemeine Parameter bei der Entwicklung von Programmen gibt, mit denen ein geringeres Wachstum des Berechnungsaufwands begünstigt werden kann. Es sollte demnach besser sein Programme so zu schreiben, dass die Körper möglichst kurz sind, möglichst wenig oder sehr viele Regeln auf ein Atom kommen und die Atom- und Regelanzahl im allgemeinen möglichst klein ist.

Ein Problem bei der Erzeugung von Testreihen ist, dass sie mit wenigen generierten Daten oder bei leichten Problemen (im leichten Bereich) leicht zu erheben sind, aber dafür auch ungenaue Ergebnisse (Rauschen stärker, weniger Daten) erzeugt werden. Auf der anderen Seite ist es auch schwerer (zeitaufwändiger) die Daten zu erheben, wenn mehr generierte Daten erzeugt werden oder die Probleme schwerer sind, dafür sind die Ergebnisse aber auch besser.

Dieser Sachverhalt erklärt auch zum Teil die falschen Schlussfolgerungen, die in früheren Untersuchungen ([29], [21], [24], [6], [25] und [5]) über den Berechnungsaufwand bei SAT und ASP gemacht wurden. Aus den untersuchten Bereichen konnte nicht ersehen werden, dass auch im leichten Bereichen oder bei gemischter Körperlänge schwere Probleme generiert werden können und das Wachstum auch dort annähernd exponentiell ist, daher wurde fälschlicherweise angenommen, dass dies nicht der Fall ist.

Mit den hier aufgezeigten Methoden, z.B. der Untersuchung des Berechnungsaufwands bei einem konstanten Wert für die problemspezifische Skalierungsformel oder Erzeugung von Testreihen für dreidimensionale Diagramme, sollten aber solche Fehler besser vermieden werden können.

7.2 Ausblick

Die Leistung von Antwortmengen- und auch SAT-Solvern könnte dadurch gesteigert werden, dass der Berechnungsaufwand der eingesetzten Heuristiken, Inferenz- und lookahead-Operatoren so angepasst wird, dass sie rund eine Größenordnung weniger Zeit benötigen, als der restliche Algorithmus.

Interessant wäre auch ein Vergleich mehrerer Solver bezüglich eines konstanten R/A Faktors und konstanter Literalanzahl, bei fester oder gemischter Körperlänge. Eine weitere Fortführung der hier gemachten Untersuchungen könnte noch zusätzliche Einblicke verschaffen. Dies ist wegen des exponentiellen Wachstums, aber sehr zeitaufwändig.

Von Nomore++ sind mittlerweile auch neuere, ausgereiftere Versionen erschienen. Eine Untersuchung des Wachstumsverhaltens der neusten Version würde mehr zugunsten von Nomore++ ausfallen.

Weiterhin wäre es für die Untersuchung des Wachstumsverhaltens sehr hilfreich, für die unterschiedlichen Modelle, geeignete Skalierungsformeln zu haben, die z.B. auch die Literale berücksichtigen. Hilfreich dafür ist eine genauere Bestimmung der Maxima der Kurven für verschiedene Parameterwerte.

Die Formel für den Berechnungsaufwand für ein Generierungsmodell zu bestimmen, gestaltet sich noch schwieriger als das Finden von Skalierungsformeln. Mit einer solchen Formel, oder einer guten Approximation für sie, könnten allerdings Solver noch besser untersucht, eingeordnet und optimiert werden. Sie würde wahrscheinlich auch das Verständnis verbessern, wodurch Probleme schwerer lösbar werden.

Mit der statistischen Untersuchungen weiterer Strukturparameter für logische Programme, wie beispielsweise die Anzahl der negativen Zyklen und/oder die durchschnittliche Länge dieser, und deren Zusammenhänge zu Berechnungsaufwands- und Erfüllbarkeitsparametern, wird das Verständnis der Zusammenhänge verbessert und außerdem einige der hier aufgestellte Behauptungen überprüft.

Es wurde auch eine Testreihe unter Smodels ohne lookahead für gemischte Körperlänge mit den Namen casesMixNLA begonnen. Leider wurden innerhalb der Zeit keine aussagekräftigen Bereiche erreicht. Eine Fortführung dieser Testreihe wäre, im Hinblick auf die Analyse des Einflusses des lookaheads, sinnvoll.

Im Zusammenhang mit dem Modell zur Suche einer Antwortmenge aus Abschnitt 4.1.2 auf Seite 34 wäre es interessant herauszufinden, wie hoch der Anteil der falschen Entscheidungen unter den Entscheidungen ist. Also wie oft bei einer Entscheidung bzw. einem choice von der Wurzel des Suchbaums zur Antwortmenge gleich der Pfad zur Antwortmenge gewält wird und wie oft nicht, und wie dieses Verhältnis mit unterschiedlichen Heuristiken beeinflußt werden kann. Dafür ist allerding eine Anpassung der Solver von nöten, da die richtigen und falschen Entscheidungen auf dem Pfad zur Antwortmenge bisher nicht ausgegeben werden und es auch keine Möglichkeit gibt, dies aus den jetzigen Ausgabewerten zu berechnen. In diesem Zusammenhang wäre auch eine Anpassung der Solver, so dass sie die Größe und die Anzahl der erfolglosen Untersuchbäume ausgeben, interessant.

Eine Untersuchung der Zusammenhänge zwischen SAT und ASP, könnte Klärung darüber bringen, warum bei ASP bei den untersuchten Generierungsmodellen für zufällig generierte Programme im Bereich des Maximum des Berechnungsaufwands kein Phasenübergang bei der Erfüllbarkeit mehr zu finden ist. Dafür können die logischen Programme eines Generierungsmodells von ASP (z.B. 2-LP) direkt in logische Formeln für SAT-Solver übersetzt werden, ähnlich wie es der ASSAT-Solver von Yuting Zhao macht, nur ohne die Einführung von Loop-Formeln. Wenn diese Formeln mit SAT-Solvern gelöst werden, können sie ähnliche Berechnungsaufwandsverläufe wie die äquivalenten Probleme bei ASP hervorbringen, aber andere Erfüllbarkeitsverläufe.

7.3 Eidesstattliche Erklärung

Ich versichere hiermit, dass die vorliegende Arbeit und die verwendeten Ideen, soweit nicht anders gekennzeichnet, von mir stammen.

Wörtliche oder sinngemäße Übernahmen aus anderen Werken wurden als solche gekennzeichnet.

Potsdam dem 1. Dezember 2005

Bernd Österholz

Literaturverzeichnis

[1] Anger, C., M. Gebser, T. Linke, A. Neumann und T. Schaub: *The nomore++ approach to answer set solving*. In: Sutcliffe, G. und A. Voronkov (Herausgeber): *Proceedings of the Twelfth International Conference on Logic for Programming Artificial Intelligence and Reasoning (LPAR'05)*, Band 3835 der Reihe *Lecture Notes in Artificial Intelligence*. Springer-Verlag, 2005. To appear.

[2] Baral, Chitta: *Knowledge Representation, Reasoning and Declarative Problem Solving*. CAMBRIDGE University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 2003. ISBN 0 521 81802 8.

[3] Ben-Naim, E. und P.L. Krapivsky: *Kinetic Theory of Random Graphs*. In: *AIP Conference Proceedings 776*, 2005.

[4] Cook, S.: *The Complexity of Theorem-Proving Procedures*. In: *Proceedings of the ACM Symposium on the Theory of Computing*, Seiten 151–158. ACM Press, 1971.

[5] Cook, Stephen A. und David G Mitchell: *Finding Hard Instances of the Satisfiability Problem: A Survey*. In: Du, Gu und Pardalos (Herausgeber): *Satisfiability Problem: Theory and Applications*, Band 35, Seiten 1–17. American Mathematical Society, 1997.

[6] Crawford, James M. und Larry D. Auton: *Experimental Results on the Crossover Point in Random 3-SAT*. Artificial Intelligence, 81(1-2):31–57, 1996.

[7] Davis, M., G. Logemann und D. Loveland: *A machine program for theorem-proving.* Communications of the ACM, 5:394–397, 1962.

[8] Davis, M. und H. Putnam: *A computing procedure for quantivication theory.* Jornal of the ACM, 7:201–215, 1960.

[9] Dowling, W. und J. Gallier: *Linear-time algorithms for testing the satisfiability of propositional horn formulae*. Journal of Logic Programming, 1:267–284, 1984.

[10] Erdogan, Selim T. und Vladimir Lifschitz: *Definitions in answer set programming*. Proc. Logic Programming and Nonmonotonic Reasoning 7, Seiten 114–126, 2004.

[11] Franco, J. und M. Paul: *Probailistic analysis of the Davis Putnam procedure for solving the satisfiability problem*. Discrete Applied Math., 5:77–87, 1983.

[12] Franco, John V. und R. P. Swaminathan: *Average Case Results for Satisfiability Algorithms Under the Random-Clause-Width Model*. Annals of Mathematics and Artificial Intelligence, 20(1-4):357–391, 1997.

[13] Frank, J. und C. Martel: *Phase transitions in the properties of random graphs*, 1995.

[14] Gelfond, M. und V. Lifschitz: *The Stable Model Semantics for Logic Programming*. In: *Proceedings of the International Conference on Logic Programming*, Seiten 1070–1080. The MIT Press, 1988.

[15] Georgii, Hans-Otto: *Gibbs Measures and Phase Transition*. de Gruyter, Berlin; New York, 1988. ISBN 0-89925-462-4.

[16] Irle, Alnrecht: *Wahrscheinlichkeitstheorie und Statistik, Grundlagen-Resultate-Anwendungen*. B.G. Teubner GmbH, Stuttgart/Leipzig/Wiesbaden, 2001. ISBN 3-519-02395-4.

[17] Kautz, Henry A. und Bart Selman: *Planning as Satisfiability*. In: *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI'92)*, Seiten 359–363, 1992.

[18] Lee, Joohyung: *A Model-Theoretic Counterpart of Loop Formulas*. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI'05)*, 2005. To appear.

[19] Lifschitz, V.: *Foundations of logic programming*. In: Brewka, G. (Herausgeber): *Principles of Knowledge Representation*, Seiten 69–127. CSLI Publications, 1996.

[20] Lin, Fangzhen und Xishun Zhao: *On Odd and Even Cycles in Normal Logic Programs.* AAAI 2004, Seiten 80–85, 2004.

[21] Lin, Fangzhen und Yuting Zhao: *Answer Set Programming Phase Transition: A Study on Randomly Generated Programs.* ICLP 2003, Seiten 239–253, 2003.

[22] Lin, Fangzhen und Yuting Zhao: *ASSAT: computing answer sets of a logic program by SAT solvers*. Artificial Intelligence, 157:115–137, 2004.

[23] Michael Greiner, Gottfried Tinhofer: *Stochastik für Studienanfänger der Informatik*. Carl Hanser Verlag München Wien, 1996. ISBN 3-446-18636-0.

[24] Mitchell, David G. und Hector J. Levesque: *Some Pitfalls for Experimenters with Random SAT*. Artificial Intelligence, 81(1-2):111–125, 1996.

[25] Mitchell, David G., Bart Selman und Hector J. Levesque: *Hard and Easy Distributions for SAT Problems*. In: Rosenbloom, Paul und Peter Szolovits (Herausgeber): *Proceedings of the Tenth National Conference on Artificial Intelligence*, Seiten 459–465, Menlo Park, California, 1992. AAAI Press.

[26] Scheid, Harald: *Wahrscheinlichkeitsrechnung*. Bibliographisches Institut & F.A. Brockhaus AG, Mannheim, 1992. ISBN 3-411-15841-7.

[27] Simons, Patrik: *Extending and Implementing the Stable Model Semantics.*, 2000. Doctoral dissertation. Research Report 58.

[28] van Gelder, Allen, Kenneth Ross und John S. Schlipf: *The Well-Founded Semantics for General Logic Programs*. Journal of the ACM, 38(3):620–650, 1991.

[29] Zhao, Yuting: *Answer set programming : SAT based solver and phase transition*. Hong Kong University, Hong Kong, 2003.