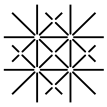

Universität
Basel

Gibt es Sudokus mit nur 16 Vorgaben?

Bachelorarbeit

Philosophisch-Naturwissenschaftliche Fakultät der Universität Basel
Department Mathematik und Informatik
Künstliche Intelligenz
http://ai.cs.unibas.ch

Beurteiler: Prof. Dr. Malte Helmert
Zweitbeurteiler: Florian Pommerening und Thomas Keller

Philipp Oldenburg
philipp.oldenburg@stud.unibas.ch
2013-061-064

14. August 2016

Danksagung

An dieser Stelle möchte ich mich bei meinen Assistenten Florian Pommerening und Thomas Keller bedanken, die mir während der gesamten Erarbeitungsphase mit konstruktivem Feedback zur Seite standen.
Dank gilt weiterhin Prof. Dr. Malte Helmert für die Möglichkeit in seiner Forschungsgruppe meine Bachelorarbeit zu absolvieren.
Weiterer Dank gilt Prof. Gary McGuire, für das zur Verfügung stellen von Mustern. Das sind Datenstrukturen, welche für meine Implementation von Bedeutung sind.

Abstract

Die Frage ob es gültige Sudokus – d.h. Sudokus mit nur einer Lösung – gibt, die nur 16 Vorgaben haben, konnte im Dezember 2011 mithilfe einer erschöpfenden Brute-Force-Methode von McGuire et al. verneint werden.
Die Schwierigkeit dieser Aufgabe liegt in dem ausufernden Suchraum des Problems und der dadurch entstehenden Erforderlichkeit einer effizienten Beweisidee sowie schnellerer Algorithmen.
In dieser Arbeit wird die Beweismethode von McGuire et al. bestätigt werden und für $2^2 \times 2^2$ und $3^2 \times 3^2$ Sudokus in C++ implementiert.

Inhaltsverzeichnis

1 Motivation

Der Begriff Sudoku kommt aus dem Japanischen und bedeutet übersetzt: Isoliere die Zahlen. Es soll ein 9×9 Gitter mit Ziffern von 1-9 gefüllt werden, sodass jede Ziffer in jeder Spalte, Zeile oder Box nur einmal vorkommt. Ausgangspunkt dafür ist ein Gitter bei dem schon mehrere Ziffern vorgegeben sind [1].
Die Frage ob es gültige Sudokus mit 16 Vorgaben gibt, stellt sich beispielsweise, wenn es darum geht neue Sudokus zu erstellen. Denn die Anzahl der vorgegebenen Zahlen ist, neben der Anordnung dieser, ein Maß dafür wie schwierig das Rätsel zu lösen ist. Wenn man von der Anordnung absieht gilt, dass es umso mehr Spielraum beim Ausfüllen des Sudokus gibt, je weniger Zahlen vorgegeben sind.
Bei der von McGuire et al. beschriebenen Beweisidee wird das ursprüngliche Problem, mithilfe von sogenannten unausweichlichen Mengen, in ein Hitting-Set-Problem transformiert [6]. Dies schmälert einerseits den Suchraum und andererseits wird der Weg geebnet für bereits bekannte Backtracking Algorithmen.
Ich werde in dieser Arbeit zunächst das Problem formell definieren. Danach werde ich ein Überblick auf die Beweismethode geben. Nach dem Überblick wird auf die einzelnen Beweisschritte genauer eingegangen. Weiterhin werden Resultate der eigenen Implementation besprochen und zuletzt ein Fazit gezogen.

2 Definition des Problems

Bis wir das eigentliche Problem definieren können, werden wir zuerst einige Grundlagen festhalten. Wenn in diesem Paper von einem Sudoku gesprochen wird ist, wenn nicht anders erwähnt, ein 9×9 Sudoku gemeint, d.h. ein Zellenfeld mit 9 Zeilen und 9 Spalten. Es lässt sich als Funktion $S : D \to \{1 \dots 9\}$ mit $D \subseteq \{0 \dots 80\}$ auffassen, d.h. eine partielle Zuweisung der insgesamt $9 \cdot 9$ Sudoku Felder zu einer Zahl.
Wir nennen ein Sudoku *konsistent* falls für die Funktion gilt, dass $\forall i, j \in \{1, ..., 80\}, i \neq j : S(i) \neq S(j)$, falls sich i und j in der gleichen Senkrechten, Waagerechten oder im gleichen 3×3 Zellenblock befinden. Formell befinden sich i und j in der gleichen Senkrechten falls $i \bmod 9 = j \bmod 9$, der gleichen Waagerechten wenn $\lfloor i \div 9 \rfloor = \lfloor j \div 9 \rfloor$ und im gleichen 3×3 Zellenblock falls $(\lfloor (i \bmod 9) \div 3 \rfloor = \lfloor (j \bmod 9) \div 3 \rfloor$ und $\lfloor \lfloor i \div 9 \rfloor \div 3 \rfloor = \lfloor \lfloor j \div 9 \rfloor \div 3 \rfloor)$.
Ein *Band* ist ein Tripel von Zeilen. Wenn die Zeilen in einem Sudoku, von oben nach unten, von 0 bis 8 durchnummeriert sind, bestehen die Bänder aus den folgenden Tripeln: $\{0, 1, 2\}, \{3, 4, 5\}, \{6, 7, 8\}$. Analog dazu gibt es in einem Sudoku 3 *Stapel*, welche Tripel von Spalten sind.
Allgemein werden wir Zeilen, Spalten, Boxen, Stapel und Bänder immer von links nach rechts bzw. von oben nach unten durchzählen. Wenn wir beispielsweise von der 3. Box im 1. Stapel sprechen ist dabei die unterste Box im linken Stapel eines Sudokus gemeint.
Unter einem Sudoku mit n *Vorgaben* verstehen wir ein Sudoku dessen Definitionsbereich eine Kardinalität von n hat.
Wir werden außerdem ein Sudoku, um darauf mit Mengenoperatoren sinnvoll arbeiten zu können, als eine zur Funktion korrespondierende Teilmenge von $\mathcal{P}(\{0 \dots 80\} \times \{1 \dots 9\})$ auffassen. $\mathcal{P}$ ist hierbei die Potenzmenge.
Das Lösen eines Sudoku S_1 ist demnach das Finden einer konsistenten Obermenge S_L mit $S_1 \subseteq S_L$, die allen 81 Feldern eine Zahl zuordnet. S_L nennen wir eine *Lösung* des Sudokus S_1.
Ein Sudoku ist *gültig*, falls nur genau eine Lösung existiert.
Wir nennen ein Sudoku *lösbar*, falls mindestens eine Lösung existiert.
Unter einem gelösten Sudoku verstehen wir eine beliebige Lösung eines beliebigen Sudoku.
Es ist offensichtlich, dass jedes gelöste Sudoku außerdem gültig sein muss, da es als einzige Lösung sich selbst besitzt, da keine weiteren Zellen ausgefüllt werden können.
Wir werden weiterhin davon sprechen, dass sich ein Sudoku S_1 *in* einem Sudoku S_2 befindet, falls $S_1 \subseteq S_2$ gilt.
Unter dem (relativen) Komplement $S_1 \setminus S_2$ zweier Sudokus S_1 und S_2 verstehen wir das Entfernen von Zellen aus S_1, welche in S_2 belegt sind.

Die Zahl der *minimalen Vorgaben* der $n \times n$ Sudokus gibt an, welches die kleinste Zahl x ist, für die es noch mindestens ein gültiges $n \times n$ Sudoku mit x Vorgaben gibt.
Die minimale Anzahl der Vorgaben eines gültigen $n \times n$ Sudokus steigt mit der Größe von n. Beispielsweise gibt es gültige 4×4 (Box: 2×2 Zellen), 6×6 (Box: 2×3 Zellen) und 8×8 (Box: 2×4 Zellen) Sudokus mit nur minimal 4, 8 und 14 Vorgaben [6, Table 1].
Die Frage um die es in diesem Papier geht ist, welches die minimale Anzahl an Vorgaben für 9×9 Sudokus ist.
Zuletzt möchte ich hier noch aufzeigen, dass es, falls es keine gültigen 9×9 Sudokus mit 16 Vorgaben gibt, auch keine geben kann mit weniger als 16. Dafür wird zuerst dieses Lemma eingeführt.

Lemma 1. *Gegeben sei ein gültiges Sudoku S_1 mit n Vorgaben und seine einzige Lösung S_L. Dann ist jedes Sudoku S_2 mit $S_1 \subseteq S_2 \subseteq S_L$ auch gültig.*

Beweis. Angenommen unsere Behauptung, dass S_2 auch gültig ist, sei falsch, das heißt, es gibt mindestens zwei Lösungen S_{L1} und S_{L2}. Diese Lösungen wären aber gleichzeitig auch Lösungen für S_1, denn nach $S_1 \subseteq S_2$ sowie $S_2 \subseteq S_{L1}$ und $S_2 \subseteq S_{L2}$ gilt $S_1 \subseteq S_{L1}$ und $S_1 \subseteq S_{L2}$. Dies widerspricht sich aber damit, dass S_1 gültig ist und demnach ist unsere Behauptung, dass S_2 ungültig ist falsch.

□

Nun können wir die ursprüngliche Behauptung beweisen:

Theorem 1. *Falls es keine gültigen Sudokus mit 16 Vorgaben gibt, dann gibt es auch keine mit $n < 16$ Vorgaben.*

Beweis. Wir nehmen an es gibt ein Sudoku S_1 das weniger als 16 Vorgaben hat und gültig ist, obwohl es keinerlei Sudokus mit 16 Vorgaben gibt. Wenn S_1 ein gültiges Sudoku ist, dann bedeutet dies nach Definition, dass es eine Lösung S_L mit $S_1 \subseteq S_L$ gibt. Nun können wir uns ein gültiges Sudoku S_2 mit 16 Vorgaben und $S_1 \subseteq S_2 \subseteq S_L$ bauen, indem wir einfach solange Zellenbelegungen aus S_L zu S_1 hinzufügen, bis wir bei einem Sudoku mit 16 Vorgaben angekommen sind. Mit Lemma 1 erhalten wir, dass S_1 dann auch ein gültiges Sudoku sein muss. Dies ist ein Widerspruch und wir können folgern, dass das Theorem gilt. □

3 Beweisvorgang

Die nun vorgestellte Beweisidee von McGuire et al. ist eine Brute-Force-Methode. Es werden alle möglichen Sudokus mit 16 Vorgaben aufgezählt und dann getestet ob es sich bei mindestens einem von den Kandidaten tatsächlich um ein gültiges Sudoku handelt.
Die vermutlich einfachste Methode die dafür benötigten Sudokus aufzustellen wäre mit einem leeren Sudoku zu beginnen und dann für jede mögliche Kombination von 16 Zellen alle konsistenten Zahlenbelegungen durchzuprobieren.
Dies ist aber eine ziemlich ungünstige Methode. Ein Grund dafür ist, dass bei dieser Suche auch Sudokus gefunden werden welche gar keine Lösung besitzen. Zum Beispiel das Sudoku in Abbildung 1, welches keine Lösung besitzen kann, weil beim Zellenausfüllen nicht gleichzeitig die Konsistenz der ersten Spalte und der letzten Zeile gewährleistet werden kann. Da wir auf der Suche nach Sudokus mit genau einer Lösung sind interessieren uns diese nicht.

1								
2								
3								
4								
5								
6								
7								
8								
	2	3	4	5	6	7	8	9

Abbildung 1: Beispiel eines konsistenten Sudoku, welches nicht lösbar ist

Eine bessere Methode ist die Suche innerhalb eines bereits gelösten Sudoku. Auf diese Weise ist gewährleistet, dass es immer mindestens eine Lösung des gefundenen Sudoku gibt. Felgenhauer and Jarvis konnten 2006 zeigen, dass es $6670903752021072936960 \approx 6.671 \times 10^{21}$ gelöste Sudokus gibt [2]. Angenommen wir könnten diese mit einer Frequenz von 1 GHz durchsuchen, was sehr optimistisch ist, würde dies immer noch ungefähr 210 000 Jahre dauern.

Um die Anzahl zu beschränken werden nun Sudokugruppen eingeführt. Diese Gruppen haben alle gemeinsam, dass jedes Sudoku darin in jedes Andere mithilfe von Symmetrieoperatoren umgewandelt werden kann. Die Symmetrieoperatoren haben alle die Eigenschaft, dass bei der Operation sämtliche Eigenschaften des Sudoku, wie Konsistenz, Lösbarkeit, Gültigkeit und so fort erhalten bleiben. Das bedeutet auch, dass wenn ein Element der Gruppe kein gültiges 16-Vorgaben-Sudoku beinhaltet, dann gilt dies auch für jedes andere Element der Gruppe. Der Grund dafür ist, dass wenn in einem Feld der Gruppe sich tatsächlich ein solcher Fund befinden würde, dann wäre es auch möglich diesen mit den Operatoren zu einem gültigen 16-Vorgaben-Sudoku innerhalb aller anderen Elemente der Gruppe zu transformieren. Dies hat für uns zur Folge, dass wir nun nur noch für jede Symmetriegruppe ein Sudoku durchsuchen müssen. Wir werden uns mit den folgenden 6 Symmetrieoperatoren beschäftigen [6, 3.1. Equivalence Transformations]:

- Vertauschen der Ziffern
- Permutieren der Reihen innerhalb eines Bands
- Permutieren der Spalten innerhalb eines Stapels
- Permutieren der Bänder
- Permutieren der Stapel
- Transponieren

Russell and Jarvis haben 2006 gezeigt, dass es genau 5472730538 Symmetriegruppen von Sudokus gibt [7]. Für den Beweis muss für jede Gruppe ein (kanonischer) Vertreter durchsucht werden.
Wir werden uns nun mit dem Suchen eines 16-Vorgaben-Sudoku innerhalb dieser Vertreter beschäftigen. Wenn jede mögliche Kombination von 16 Zellen aus den insgesamt 81 Zellen ausgewählt werden sind dies insgesamt $\binom{81}{16} \approx 3.36 \times 10^{16}$ Kombinationen.
Um diesen Problem aus dem Weg zu gehen haben sich McGuire et al. ein bekanntes Konzept zu Nutze gemacht. Dieses Konzept wurde von Slaney 2014 verallgemeinert, sodass es für sämtliche Kombinationsprobleme angewandt werden kann [9]. Slaney nennt dieses Konzept Mengen-Dualität. Was die Mengen Dualität ist und wie diese auf unser Problem bezogen benutzt werden kann sei hier im folgenden geschildert.

Sei S_L ein gelöstes Sudoku und θ die Menge aller gültigen Sudokus in S_L. Die duale Menge θ^* ist definiert als $\{x \subseteq S_L : \overline{x} \notin \theta\}$ mit $\overline{x} := S_L \setminus x$. In Worten ausgedrückt heißt das, dass eine Zellenkombination in θ^*, aus unserem ursprünglichen Sudoku S_L entfernt, ein Sudoku ergibt, welches nicht gültig ist. Diese Mengen spielen auf unserer Suche nach gültigen Sudokus mit wenigen Zellen eine wichtige Rolle, um Gültigkeit zu bewahren. Wir nennen diese *unausweichliche Mengen*[1]. In diesem Kontext ist unter einer Menge eine Menge an Zellenbelegungen zu verstehen.
Unter einer minimalen unausweichlichen Menge verstehen wir eine unausweichliche Menge deren echte Teilmengen nicht mehr unausweichlich, also nicht mehr in θ^* sind.
In Abbildung 2 befinden sich zwei Sudokus, die jeweils eine unausweichliche Menge enthalten.

2	7	3	9	6	4	8	5	1
4	6	9	1	5	8	7	2	3
1	8	5	2	7	3	4	6	9
8	2	1	3	4	6	5	9	7
5	4	6	7	9	2	3	1	8
3	9	7	8	1	5	2	4	6
7	1	8	5	2	9	6	3	4
6	3	2	4	8	1	9	7	5
9	5	4	6	3	7	1	8	2

2	7	3	9	6	4	8	5	1
4	6	9	1	5	8	7	2	3
1	8	5	2	3	7	4	6	9
8	2	1	3	4	6	5	9	7
5	4	6	7	9	2	3	1	8
3	9	7	8	1	5	2	4	6
7	1	8	5	2	9	6	3	4
6	3	2	4	8	1	9	7	5
9	5	4	6	7	3	1	8	2

Abbildung 2: Die beiden Sudokufelder sind Lösungen des Sudokus, welches nur aus den schwarzen Zahlen besteht. Die Zellen der gefärbten Zahlen gehören jeweils zu einer minimalen unausweichlichen Menge

Wenn die Zellen mit den gefärbten Zahlen aus dem Sudoku entfernt würden, ergäbe dies ein ungültiges Sudoku, da es zwei Lösungen gibt bei denen jeweils die eingefärbten 7er mit den eingefärbten 3ern die Plätze getauscht haben.
Wir werden nun eine Verbindung herstellen zwischen den unausweichlichen Mengen in θ^* und unseren gesuchten Mengen in θ mithilfe des folgenden Lemmas und des darauf aufbauenden Theorem.

[1] Weshalb die Mengen unausweichlich sind, klärt sich in Theorem 2.

Lemma 2. *Gegeben sei ein gelöstes Sudoku S_L, die Menge θ aller gültigen Sudokus in S_L, sowie die duale Menge θ^*. Wenn eine Menge M_1 nicht in θ^* ist, dann befinden sich auch auch keine Teilmenge M_T von M_1 in θ^*.*

Beweis. Da $M_1 \notin \theta^*$ gilt nach Definition auch $\overline{M_1} \in \theta$. Wir nehmen nun an, es befindet sich doch ein M_T in θ^* und folgern $\overline{M_T} \notin \theta$ und $\overline{M_T} \in \theta^*$. Da $M_T \subseteq M_1$ gilt auch $\overline{M_1} \subseteq \overline{M_2}$. Mit $\overline{M_1} \subseteq \overline{M_T} \subseteq S_L$ und Lemma 1 lässt sich folgern, dass $\overline{M_T} \in \theta$. Damit haben wir einen Widerspruch und können folgern, dass jedes $M_T \notin \theta^*$. □

Theorem 2. *Gegeben sei ein gelöstes Sudoku S_L und die dazugehörigen Mengen θ und θ^*. Dann gilt für jedes $M_1 \in \theta$, das dieses ein Hitting-Set von θ^* ist (das bedeutet, dass jedes $M_2 \in \theta^* : M_1 \cap M_2 \neq \emptyset$).*

Beweis. Nach Definition von θ^* gilt $\overline{M_1} \notin \theta^*$. Mit Lemma 2 können wir folgern, dass es keine Teilmenge S von $\overline{M_1}$ in θ^* gibt. Da S genau die Mengen sind die M_1 nicht schneidet, da $S \subseteq \overline{M_1} = S_L \setminus M_1$, können wir folgern, dass M_1 alle Mengen in θ^* trifft. □

Wir wissen nun, dass jedes gültige 16-Vorgaben-Sudoku alle unausweichlichen Mengen schneiden muss. Dies können wir nutzen indem wir während unserer Suche innerhalb eines Vertreters zuerst unausweichliche Mengen finden und dann nach Hitting-Set mit Größe 16 suchen. Unter den gefundenen Hitting-Set sind dann unweigerlich alle gültigen Sudokus mit 16 Vorgaben vorhanden. Da die Zeit für das Aufzählen der Hitting-Sets mit der Anzahl der gefundenen unausweichlichen Mengen steigt, werden wir nicht versuchen alle unausweichlichen Mengen zu finden, sondern nur eine genügend große Teilmenge von θ^*.
Das Arbeiten auf einer Untermenge von θ^* ist möglich, ohne dabei einen möglichen 16-Vorgaben-Kandidaten zu übersehen, denn jedes Hitting-Set von θ^* ist auch ein Hitting-Set jeder Teilmenge von θ^*.
Mit Theorem 2 können wir nicht folgern, dass jedes Hitting-Set auch ein gültiges Sudoku ist. Deshalb müssen wir nach einem Fund eines Hitting-Sets noch testen ob das korrespondierende Sudoku gültig ist. Dazu werden wir die entsprechenden Kandidaten mit einem Sudoku-Solver lösen.
Damit haben wir das ursprüngliche Problem, nämlich dem Finden von gültigen Sudokus mit 16 Vorgaben aus θ, nach Slaney in sein duales Äquivalent, dem Finden von Hitting-Sets von θ^* umgewandelt. Der Suchraum konnte somit wesentlich eingeschränkt werden.

Um den vollständigen Beweis zu vollbringen müssen wir:

- einen Vertreter für jede Gruppe von gelösten Sudokus finden
- minimale unausweichlichen Mengen innerhalb der Vertreter aufzählen
- alle Hitting-Sets mit Größe 16 für die unausweichlichen Mengen suchen
- alle Kandidaten mit einem Sudoku-Solver lösen um auf Gültigkeit zu prüfen

Mit dem ersten Schritt werden wir uns in dieser Arbeit nicht weiter auseinandersetzen. McGuire et al. haben bei ihrem Brute-Force-Beweis ein Programm namens *sudoku* [3] benutzt, welches in der Lage ist Representanten der Sudokugruppen aufzuzählen.

4 Details zu den Schritten

In dieser Sektion wird es darum gehen welche Konzepte McGuire et al. verwendet haben um einen möglichst schnellen Suchlauf zu erhalten. Wir werden uns jedes dieser Konzepte anschauen und auf Konsistenz prüfen. Weiterhin werden wir uns im letzten Unterkapitel mit dem Lösen von Sudokus beschäftigen.

4.1 Aufzählen der minimalen unausweichlichen Mengen

Um minimale unausweichliche Mengen aufzuzählen werden wir Muster, d.h. eine Menge von Mengen von Zellen $M \subseteq 2^{\{0,\dots,80\}}$ verwenden. Ein Muster $\{M_0, M_1, \dots, M_i\}$ mit $M_i \subseteq \{0, \dots, 80\}$ trifft ein (gelöstes) Sudoku S_L genau dann, wenn für alle $j \in \{0, \dots, i\}$ und alle $m_1, m_2 \in M_j$ gilt $S_L(M_j(m_1)) = S_L(M_j(m_2))$. Das Muster gibt also an welche Zellen in unserem Sudoku die gleiche Ziffer beinhalten, damit das Muster auf diese *zutrifft*.
Falls das Muster auf ein Sudoku zutrifft ist die Vereinigung der Zellen des Sudoku, welche sich mit dem Muster überlappen eine *minimale* unausweichliche Menge. Beispielsweise ist das Muster, das auf das gelöste Sudoku in Abbildung 2 zutrifft: $\{\{22, 77\}, \{23, 76\}\}$.
Muster lassen sich genau wie Sudokus als Teilmenge von $\mathcal{P}(\{0 \dots 80\} \times \{1 \dots 9\})$ auffassen, wobei alle Zellen, die sich in der selben Menge des Musters befinden, mit derselben Zahl belegt werden. Dadurch lassen sich ebenfalls alle Symmetrieoperatoren auch auf Muster anwenden.
Wir werden auch hier davon sprechen, dass sich ein Muster M_1 in einem anderen Muster M_2 befindet, wenn gilt $M_1 \subseteq M_2$.
Weiterhin sprechen wir von einem $n \times m$ Muster, wenn dieses Zellen in n Bändern und m Stapeln hat.
Wir gehen nun folgend davon aus, dass $n \leq m$, ohne dabei an Generalität zu verlieren, da entsprechende Muster mit $m > n$ einfach transponiert werden können.
Die in diesem Projekt verwendeten Muster stammen aus dem Programm *checker* [5], welches von McGuire et al. benutzt wurde um die erschöpfende Suche auf einem Computercluster zu vollbringen. Wie bereits erwähnt ist es nicht das Ziel alle minimalen unausweichliche Mengen mit diesen zu finden. Je mehr unausweichliche Mengen gefunden werden, desto länger braucht der Hitting-Set Algorithmus zum Aufzählen. Dieses führt allerdings zu einer verringerten Anzahl an 16-Vorgaben-Sudoku Kandidaten, die geprüft werden müssen. McGuire et al. haben empirisch bestimmt, dass der Algorithmus die kürzeste Laufzeit hat, wenn alle unausweichliche Mengen bis Größe 12

gesucht werden. Deshalb finden sich in ihrer Liste auch nur Muster mit maximal 12 Elementen.
Die verwendeten Muster sind asymmetrisch untereinander in Bezug auf unsere Symmetrieoperatoren aus Kapitel 3. Um mit diesen alle möglichen unausweichlichen Mengen bis Größe 12 zu finden müssen alle möglichen Transformationen auf den Vertreter angewandt werden. Danach wird jeweils getestet, ob eines der Muster darauf zutrifft. Falls dies der Fall ist, werden die Operationen die auf das Sudoku angewandt wurden, umgekehrt auf das Muster angewandt um die entsprechende unausweichliche Menge zu finden, die auf den ursprünglichen Vertreter zutrifft.
Dieser Prozess lässt sich beschleunigen, indem nicht alle transformierten Vertreter auf Muster geprüft werden müssen. In *checker* wurden alle Muster mit Äquivalenzumformungen in bestimmte Formen gebracht, die sich mit den Mustern in Abbildung 3 und 5 überlappen. Die Beweisideen für die folgenden Theoreme, Lemmata und Korollare stammen teilweise aus dem Papier von McGuire et al. [6].
Für ein Muster M mit Definitionsbereich D gibt es eine bijektive Abbildung $P : Z_1 \mapsto Z_2$ mit $Z_1, Z_2 \in D$ welche jede Zelle von M auf eine weitere Zelle verweist. Wir nennen P eine *Permutation*. Wenn S ein gelöstes Sudoku ist, auf das M zutrifft, nennen wir eine Permutation P, die nach ihrer Ausführung auf S zu einem weiteren gelösten (konsistenten) Sudoku führt *Musterpermutation*.

Korollar 1. *Gegeben sei ein gelöstes Sudoku S_L und eine minimale unausweichliche Menge U mit $U \subseteq S_L$ und Definitionsbereich D. Das zu U korrespondierende Muster nennen wir M. Weiterhin seien $S_0, \ldots, S_{n-1}$ alle n Lösungen von $S_L \setminus U$. Dann gibt es für jedes Paar $i, j \in \{0, \ldots, n-1\}$ eine Musterpermutation P_Z mit denen sich S_i in S_j ineinander umwandeln lassen und weiterhin für jedes $x \in D$ gilt, dass x und $P_Z(x)$ in derselben Zeile sind. Außerdem gilt für M, dass es mindestens eine Musterpermutation gibt.*

Beweis. Mithilfe von Permutationen lassen sich alle möglichen Zeilenvervollständigungen von $S_L \setminus U$ finden, welche keine Ziffer doppelt enthalten. In Kombination erhält man damit alle Vervollständigungen V mit $\{(S_L \setminus U) \subseteq V\}$ für die jeweils nur Konsistenz innerhalb jeder Zeile gewährleistet ist. Unter diesen Permutationen befindet sich auch unweigerlich alle P_Z, da dies ein Spezialfall ist bei dem Konsistenz innerhalb Zeilen, Spalten und den Boxen gleichzeitig gegeben ist.
$S_L \setminus U$ hat aufgrund der Eigenschaften von U mindestens zwei Lösungen. Es gibt also mindestens immer eine Musterpermutation. □

Es lässt sich analog zeigen, dass das Korollar 1 auch für Musterpermutionen gilt, welche Zellen nur innerhalb von Spalten oder Boxen permutieren.
Weiterhin gilt für jedes Muster M und entsprechender Musterpermutation P_M mit Definitionsbereich D, dass für jedes $Z_1 \in D : Z_1 \neq P_M(Z_1)$. Im gegenteiligen Fall gäbe es einen Widerspruch zur Minimalität der gefundenen unausweichlichen Mengen von M.

Lemma 3. *Gegeben sei ein gelöstes Sudoku S_L. Dann kommt in jeder beliebigen minimalen unausweichliche Menge M aus S_L jede in M enthaltene Ziffer mindestens zweimal vor.*

Beweis. Angenommen es gibt so eine Menge M, die eine Ziffer nur ein mal besitzt. Das heißt für unser Sudoku $S_L \setminus M$, dass diese Ziffer nur einmal aus S_L entfernt wird. Da jedes gelöste Sudoku jede Zahl genau 9 mal enthalten muss, um konsistent zu sein, gibt es dann in $S_L \setminus M$ bereits 8 mal diese Ziffer. Wenn man nun nach der Lösung dieses Sudoku sucht wird man feststellen, dass es nur eine Möglichkeit gibt die Ziffer in eine Zelle einzutragen, da bereits 8 Zeilen und Spalten für die Besetzung ausgeschlossen sind. Da M unausweichlich ist muss es aber mind. zwei unterschiedliche Lösungen geben und da sie sich in dieser Ziffer nicht unterscheiden muss der Unterschied in den restlichen Zellen in M liegen. Damit ist M, ohne die Zelle mit der einzigartigen Ziffer, immer noch eine unausweichliche Menge. Dies widerspricht sich mit der Minimalität von M. Wir folgern also, dass in M jede Ziffer mindestens zwei mal vorkommt. □

Aus Lemma 3 lässt sich auch schließen, dass auch für Muster gelten muss, dass diese Ziffern mehrmals enthalten. Da dieses sonst nicht auf eine minimale unausweichliche Menge zutrifft.
Wenn wir davon sprechen, dass ein Muster sich in einer bestimmten Form befindet ist damit gemeint, dass sich die Zellenbelegungen der Form in diesem befinden.

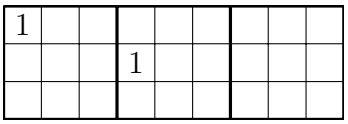

Abbildung 3: Allgemeine Form des oberen Bandes von Mustern in *checker*

Theorem 3. *Jedes obere Band aller Muster lässt sich mit Äquivalenzumformungen in die Form aus Abbildung 3 bringen.*

Beweis. Wir werden für diesen Beweis zuerst zeigen, dass jede Ziffer in einem Muster nochmals in demselben Band oder Stapel vorkommt. Um einen Widerspruch zu erhalten gehen wir nun aber vom Gegenteiligen aus. Bei eindimensionalen Mustern gibt es sofort einen Widerspruch mit Lemma 3. Mit Lemma 3 und einigen Äquivalenzumformungen können wir jedes (mehrdimensionale) Muster so umformen, dass sich die folgenden Zellenbelegungen in diesem befinden:

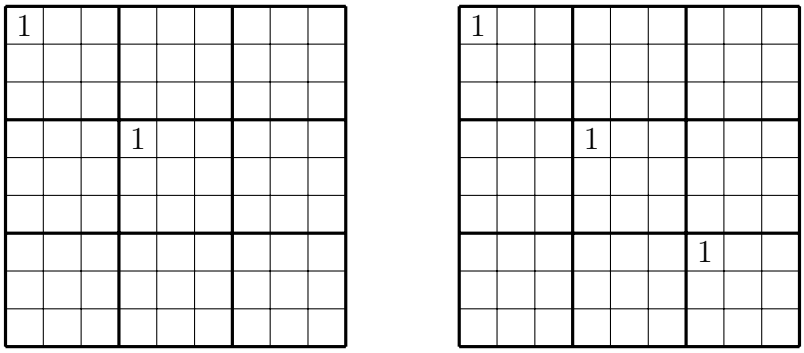

Abbildung 4: Jedes $m \times n$ Muster mit $n \geq m \geq 2$ und der Annahme, dass jede Ziffer eines Musters nur einmal im gleichen Stapel oder Band vorkommt, lässt sich in eine äquivalente Form bringen welche die obigen Zellenbelegungen enthält.

Mit Korollar 1 wissen wir, dass es eine Musterpermutation gibt, welche Zellen eines gelösten Sudokus nur innerhalb ihrer Boxen permutiert. Eine solche Musterpermutation kann es aus Konsistenzgründen gar nicht geben. Wenn die Musterpermutation eine Ziffer in eine andere Zelle permutieren würde, gäbe es nach Durchführung der kompletten Musterpermutation in entweder der Zeile oder Spalte der Zelle in der sich die Ziffer ursprünglich befunden hat, diese Ziffer nicht mehr. Der Grund dafür ist, dass bei der Musterpermutation innerhalb der Boxen für diese fehlende Ziffer nicht aufgekommen werden kann. In Abbildung 4 ist zu sehen, dass die Boxen der belegten Zellen sich nicht mit den Zeilen bzw. Spalten der anderen belegten Zellen schneiden. Damit müssten die Ziffern, um zu einem konsistenten Sudoku zu führen, an ihrem Platz bleiben, was sich aber wiederum mit der Minimalität unserer Muster widerspricht.
Damit hätten wir gezeigt, dass eine Ziffer unserer Muster mindestens zwei mal innerhalb eines Bands oder Stapels vorkommen muss. Im ersten Fall ist nach einer Bandpermutation, einer Zeilenpermutation und einem umbenennen der Ziffern das zu beweisende bereits gezeigt. Für letzteren Fall, dass dieselben Ziffern nur in Stapeln vorkommen, können wir in den Fällen von

2×2 und 3×3 Mustern einfach transponieren und landen wieder im ersten Fall. Weiterhin erinnern wir uns, dass wir nur $n \times m$ Muster mit $n \leq m$ betrachten und deshalb die einzige Möglichkeit in der wir tatsächlich, auch durch transponieren, nicht in die gewünschte Form kommen können ist, wenn wir ein 2×3 Muster haben, das nur Ziffern enthält, welche nur mehrfach in Stapeln, nicht aber in Bändern, vorkommen.
Ich möchte zeigen, dass es gar keine solchen 2×3 Muster geben kann. Wir gehen nun davon aus, dass es ein solches 2×3 Muster mit den erwähnten Eigenschaften gibt. Mit Korollar 1 wissen wir, dass es eine Musterpermutation gibt, welche Zellen eines gelösten Sudokus nur innerhalb ihrer Zeilen permutiert. Da es keine Ziffern gibt, welche in mehreren Stapeln vorkommen, kann bei dieser Musterpermutation nicht stapelübergreifend permutiert werden. Es gäbe sonst nach der Musterpermutation Inkonsistenzen, da für das Fehlen einer Ziffer in ihrer ursprünglichen Box nicht aufgekommen werden kann. Dann gilt auch, dass sich durch das Permutieren der Zellen einer Musterpermution, welche nur innerhalb eines Stapels vorkommen, keine Inkonsistenzen außerhalb des Stapels erzeugt werden würden. Es würde dementsprechend reichen nur die Zellen der Musterpermutation innerhalb eines Stapels zu verändern, um zu einem weiteren konsistenten Sudoku zu kommen, ohne dabei die anderen Stapel zu verändern. Dies ist ein Widerspruch zur Minimalität des Musters und wir folgern, dass es keine solchen 2×3 Muster, mit Ziffern die nur innerhalb eines Stapels vorkommen, gibt.
Damit wären für alle möglichen Fälle gezeigt, dass diese in die Form in Abbildung 3 gebracht werden können. □

Der Vorteil, dass sich die Muster in diesen allgemeinen Formen befinden entsteht dadurch, dass die transformierten gelösten Sudokus noch bevor sie mit allen Mustern durchsucht werden aussortiert werden können. Denn falls sich das transformierte gelöste Sudoku nicht in dieser Form befindet trifft auch keines der Muster auf dieses zu.
Um diesen Vorteil noch weiter auszubauen haben McGuire et al. alle $n \times m$ Muster mit $n \geq m \geq 2$ in die Formen aus Abbildung 5 gebracht.

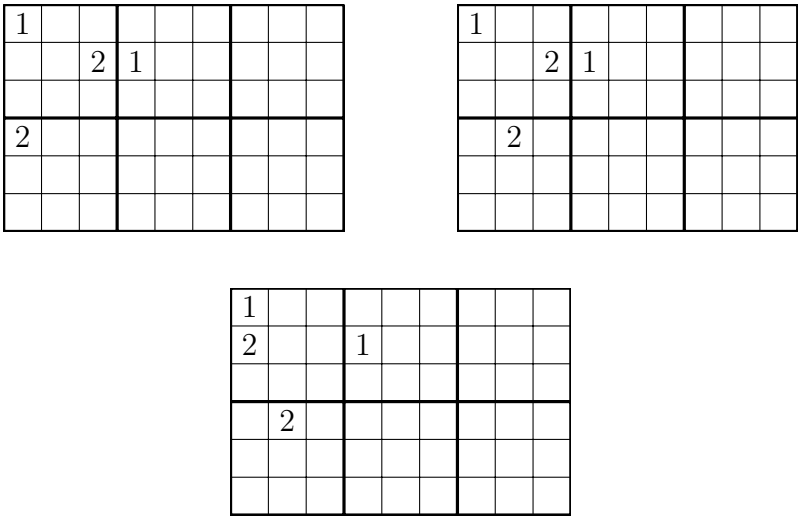

Abbildung 5: Drei Formen der oberen Bänder von $n \times m$ Mustern mit $n \geq m \geq 2$.

Ich habe versucht via einer Fallunterscheidung zu zeigen, dass dies auch für jedes Muster tatsächlich möglich ist. Dies ist mir jedoch nicht ganz gelungen, ich möchte hier jedoch die Idee vorstellen.
Ausgehend von der Konfiguration aus Abbildung 3 werden wir nun zeigen, dass mit Äquivalenzumformungen, eine weitere andere Ziffer, welche wir 2 nennen werden, jeweils in den oberen zwei Boxen mindestens eines Stapels vorkommen muss.
Wir wissen, dass es mindestens eine Ziffer im oberen Band geben muss, welche bei einer Musterpermutation, innerhalb von Spalten (Korollar 1), das Band verlässt, da sich sonst zeigen lässt, analog wie im Beweis von Theorem 3, dass das obere Band an sich schon ein Muster ist und die unausweichliche Menge, welche zum kompletten Muster gehört nicht mehr minimal. Dies bedeutet auch, dass es ein weiteres Vorkommen dieser Ziffer im selben Stapel geben muss. Falls diese im untersten Band vorkommt, lässt sich das untere Band einfach via Bandpermutation mit dem Mittleren tauschen.
Weiterhin gilt auch, dass wenn diese beiden Ziffern bei einer Musterpermutation innerhalb von Spalten ihre Bänder verlassen, es dann ein weiteres Paar von Ziffern gibt, welche in denselben Boxen vorkommen, da unsere Musterpermutation, innerhalb von Spalten, sonst inkonsistent wäre.
Wir können schließen das mindestens eins von diesen beiden Zahlenpaaren verschieden von 1 sein muss. Damit wissen wir, dass jedes Muster folgende Eigenschaft hat: Es befinden sich mindestens ein 2-er Paar in den ersten beiden Boxen des 1., 2. oder 3. Stapels. Dabei kann ein Muster ein 2-er Paar

auch in mehreren Stapeln haben, aber in mindestens einem muss es der Fall sein.
Im Fall, bei dem das 2-er Paar sich im 1. Stapel befindet gibt es folgende 11 konsistente Möglichkeiten, nachdem die 2 in der 2. Box des 1. Stapels in die beiden möglichen Zellen aus den drei Formen in Abbildung 5 gebracht wurde:

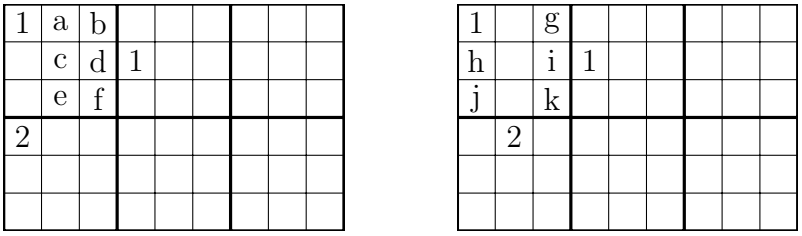

Abbildung 6: 11 Äquivalenzformen der Muster bei denen das Zweierpaar im 1. Stapel vorkommt. Die Buchstaben stehen für die möglichen Zellen der zweiten 2.

Von diesen 11 Möglichkeiten sind wiederum 3 äquivalent. Fall a und b, c und d sowie e und f sind jeweils äquivalent, da diese mit einer Spaltenpermutation ineinander umgeformt werden können (siehe Abbildung 6). Damit bleiben insgesamt noch 8 Fälle, wobei sich von diesen 5 Fälle (solche in denen es zwei Ziffern gibt welche gleichzeitig in derselben Zeile oder Spalte sind und sich außerdem in unterschiedlichen Boxen befinden) mit Äquivalenzumformungen in eine der drei Formen aus Abbildung 5 bringen lassen. Bei den restlichen 3 Fällen bin ich mir nicht sicher, aber ich vermute, dass sich das Muster wieder in zwei Teile aufteilen lässt und dies ein Widerspruch zur Minimalität gäbe. Alle Muster, welche die 2-en im mittleren Stapel hat, können mit Stapeltausch von dem 1. und 2. Stapel und einem Zeilentausch von der 1. und 2. Zeile in den Fall mit den 2-en im ersten Stapel (Abbildung 6) transformiert werden.
Im letzten Fall, in dem die beiden 2-er im rechten Stapel auftauchen, gibt es 6 mögliche Äquivalenzformen (siehe Abbildung 7), von denen wieder 3 äquivalent sind. Hier ist mir keine gute Begründung eingefallen, warum diese auch äquivalent zum Fall mit den 2-en im 1. Stapel ist. Ich vermute, dass sich zeigen lässt, dass in diesem Fall eine weitere 1 in der 1. Box des 3. Stapels vorkommen muss, da sonst das Muster eventuell nicht mehr minimal wäre. Dann wäre es wieder äquivalent zu Fall 1.
Hiermit haben wir also gezeigt, dass sich alle Muster auf 11 Fälle zurückführen lassen, von denen sich 5 in entsprechende Formen aus Abbildung 5 überführen

lassen. Ich hoffe, dass diese Fallunterscheidungen einige Anhaltspunkte geben konnte, weshalb jedes Muster in diese Form gebracht werden kann.

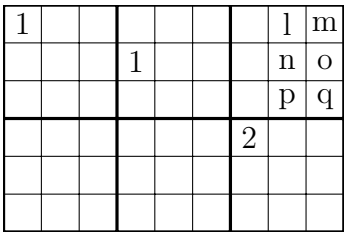

Abbildung 7: 6 Äquivalenzformen der Muster bei denen das Zweierpaar im 3. Stapel vorkommt. Die Buchstaben stehen für die möglichen Zellen der zweiten 2.

4.2 Transformationen der Vertreter

Kommen wir nun zu den nötigen Transformationen, welche wir auf den gelösten Sudokus ausführen, um zu garantieren, alle möglichen minimalen unausweichlichen Mengen bis maximal 12 Elementen mit unseren asymmetrischen Mustern zu finden. Bei diesem Prozess wird versucht durch die Struktur der Muster die Anzahl der Transformationen so gering wie möglich zu halten.
Wir beginnen mit dem einfachsten Fall, für Muster die sich nur innerhalb eines Bands oder Stapels befinden (Abbildung 3). Da sich in der Musterliste nur Muster befinden mit mehr Stapel als Bänder muss mit Transponieren sichergestellt werden, dass auch unausweichliche Mengen, bei denen dies nicht der Fall ist, gefunden werden können. Dazu kommen drei Bandpermutationen, sodass jedes Band sich einmal an oberster Stelle befindet, da sich alle Zellen unserer Muster dort befinden. Dazu kommen alle möglichen Zeilenpermutationen des ersten Bandes.
Nun folgen die Spalten- und Stapelpermutationen. Zum einen werden alle möglichen Stapelpermutationen, sowie Spaltenpermutationen des 1. Stapels durchgegangen. Bei den Spaltenpermutationen im 2. Stapel werden nur solche in Betracht gezogen, sodass jede Spalte einmal auf der linken Seite ist. Eigentlich würde man dadurch die Hälfte der unausweichlichen Mengen nicht finden, jedoch befinden sich in unserer Musterliste solche, die in mehr als einer Spalte des mittleren Stapels eine Ziffer enthalten doppelt, mit jeweils der mittleren und rechten Spalten permutiert. Der Grund warum dies gemacht wird ist, dass es nur sehr wenige Muster gibt, die Ziffern in zwei Spalten des mittleren Stapels haben, aber dafür die Hälfte an Permutationen gespart

werden kann ($3 \div 3! = \frac{1}{2}$). Analog wird mit Mustern mit nur einem Band und 3 Stapeln verfahren. Bei diesen gibt es auch wieder nur 3 Spaltenpermutationen im rechten Stapel. Diese Muster kommen dann falls sie Ziffern in mehr als einer Spalte im 2. und 3. Stapel haben sogar 4 mal in der Liste vor:

1			2		3	4		5
6			1			2		
5					4	6		3

1			2		3	4	5	
6			1			2		
5					4	6	3	

1			2	3		4	5	
6			1			2		
5				4		6	3	

1			2	3		4	5	
6			1			2		
5				4		6	3	

Abbildung 8: 1×3 Muster mit Ziffern in 2 Spalten jeweils in der mittleren und rechten Box

Zusammenfassend kommt man damit auf $2 \cdot 3 \cdot 3! \cdot 3! \cdot 3! \cdot 3 = 3\,888$ Transformationen für die 1×2 Muster. Für die 1×3 Muster kommt man auf $3\,888 \cdot 3 = 11\,664$ Kombinationen. Jedoch werden wir, wie bereits erwähnt, vor dem testen, welche Muster auf ein transformierten Vertreter zutreffen, noch sicherstellen, dass die transformierten Vertreter sich auch in der Form aus Abbildung 3 befinden. Damit reduziert sich die Anzahl jeweils noch einmal auf ein Sechstel. Der Grund dafür ist, dass sich die Ziffer in der ersten Zelle des Sudokus in 6 verschiedenen Stellen in der zweiten Box des Sudokus aufhalten kann, da 3 der Stellen der zweiten Box durch eine sonstige Inkonsistenz ausgeschlossen sind.
Die Permutationen für 2×2 und 2×3 Muster werden beinahe analog zu den 1×2 und 1×3 gemacht, jedoch müssen nun sämtliche Bandpermutationen betrachtet werden und zusätzlich alle Reihenpermutationen im zweiten Band.
Bei den Kombinationen für 3×3 Muster muss nichts zusätzlich zu den 2×3 Mustern gemacht werden, da bereits alle Bandpermutationen gemacht werden.
Es ist sinnvoll bei dem Transformieren der Vertreter zuerst solche Zellen zu transformieren, die in den Formen aus Abbildung 3 und 5 vorkommen. Dadurch kann schon frühzeitig auf diese Form getestet werden und einige Transformationen können gespart werden. Dies ist möglich, da alle unsere Transformationen bis auf die Transponierung assoziativ sind. Jedoch muss darauf geachtet werden, dass die Transponierung entweder am Anfang

oder am Schluss stattfindet, da es sonst zu äquivalenten Transformationen kommt. Wenn wir beispielsweise zuerst eine Bandpermutation, gefolgt von einer Transponierung auf einem Sudoku durchführen, führt dies zur selben Transformation als wenn wir zuerst transponieren und dann eine analoge Stapelpermutation ausführen.
Wie bereits erwähnt müssen in dem Fall einer Mustererkennung die angewandten Transformationen invers auf das Muster angewandt werden um zu der erwünschten minimalen unausweichlichen Menge zu gelangen. Invers bedeutet in unserem Fall, dass wir erneut alle angewandten einzelnen Transformationen in umgekehrter Reihenfolge anwenden, da das Inverse aller unserer Transformationen gleich der jeweiligen Transformationen selbst ist.

4.3 Das Suchen der Hitting-Sets

Der nun vorgestellte Weg um alle Hitting-Sets bis zur Größe 16 aufzuzählen wurde von McGuire et al. über mehrere Jahre hinweg weiter verbessert bis die Performance an einem Punkt war, dass es möglich war die ganze Suche innerhalb eines knappen Jahres zu vollbringen, was dann auch im Jahre 2011 getan wurde. Wir werden uns hier nun die Version im Endstadium anschauen.
Der Grundbaustein ist ein einfacher Backtracking Algorithmus, der sich eine noch nicht getroffene Menge aussucht und dann rekursiv verzweigt und jeweils eines der Elemente aus der Menge zu dem aufbauenden Hitting-Set hinzugefügt wird. Dieser Vorgang wird solange wiederholt bis alle Mengen getroffen sind.
Für unseren Beweis müssen wir alle Hitting-Sets der Größe 16 finden. Das heißt wir sind bei unserem Algorithmus nicht daran interessiert möglichst kleine Hitting-Set zu finden. Stattdessen versuchen wir eine möglichst schnelle vollständige Suche nach Hitting-Sets der Größe 16 zu machen. Wir müssen falls wir ein Hitting-Set mit einer Größe ≤ 16 gefunden haben dieses mit jeder möglichen Kombination an zusätzlichen Zellen auffüllen, damit gewährleistet ist sämtliche Hitting-Sets mit Größe 16 zu finden.

Lemma 4. *Der Backtracking Algorithmus zum Aufzählen der Hitting-Sets findet alle minimalen Hitting-Sets.*

Beweis. Für jedes beliebige minimale Hitting-Set H_M gilt, dass der Algorithmus eine Teilmenge $T \subseteq H_M$ erstellt, da der Algorithmus zu Anfang mit einer leeren Menge startet, welche Teilmenge von jeder Menge ist.
Jedes $T \subset H_M$ wird außerdem von dem Algorithmus rekursiv zu H_M vervollständigt werden, denn für jedes $T \subset H_M$ wird der Algorithmus ein

$T_2 \subseteq H_M$ mit $T \subset T_2$ finden. Der Grund dafür ist, dass der Algorithmus, nachdem T erstellt wurde noch kein Hitting-Set gefunden hat, da jede Teilmenge von H_M kein Hitting-Set ist. Deshalb wird eine noch nicht getroffene Menge ausgesucht und aus dieser mindestens ein Element an T angehängt welches sich in H_M befindet, da H_M jede Menge trifft. Da diese Eigenschaften für jeden Schritt innerhalb des Algorithmus gegeben sind wird der Algorithmus nicht aufhören, bevor nicht jedes H_M aufgezählt wurde.

□

Lemma 5. *Jedes Hitting Set H einer Menge von Mengen ist eine Obermenge von einem minimalen Hitting-Set H_M.*

Beweis. Angenommen es gibt ein Hitting-Set H welches keine Obermenge von einem minimalen Hitting-Set H_M ist.
Dies bedeutet, dass keine Untermenge $U \subseteq H$ ein minimales Hitting-Set ist. Nun gibt es zwei Möglichkeiten. Einerseits könnte es eine oder mehr Hitting-Sets in den Teilmengen geben. Dies würde aber bedeuten, dass zumindest eines von diesen Hitting-Sets ein minimales sein muss, da es nur eine begrenzte Anzahl an Teilmengen gibt und mindestens die leere Menge würde ein minimales Hitting-Set sein. Die andere Möglichkeit ist, dass es unter den Teilmengen keine Hitting-Sets gibt. Dies ist aber auch die Definition eines minimalen Hitting-Set. Wir folgern also in beiden Fällen, dass unsere Annahme falsch ist. □

Theorem 4. *Der Backtracking Algorithmus findet alle Hitting-Sets der Größe 16.*

Beweis. Angenommen der Backtracking Algorithmus würde nicht alle Hitting-Sets der Größe 16 finden.
Der letzte Schritt des Algorithmus besteht aus dem Finden aller Obermengen der Größe 16 von den aufgezählten Hitting-Sets. Mit Lemma 5 lässt sich schließen, dass unser Backtracking Algorithmus nicht alle minimalen Hitting-Sets findet. Dies ist ein Widerspruch zu Lemma 4 und wir schließen, dass unsere Annahme falsch sein muss. □

Ein Problem dieses Algorithmus ist in Abbildung 9 ersichtlich.

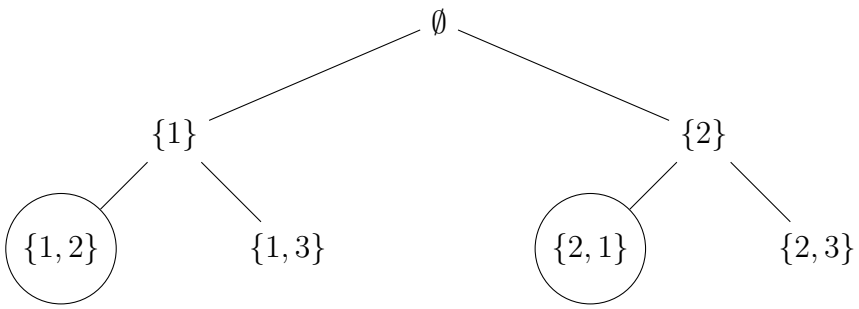

Abbildung 9: Suchbaum des Backtracking Algorithmus auf der folgenden Menge von gefundenen unausweichlichen Mengen: $\{\{1,2\},\{1,3\},\{2,3\}\}$. Alle Blätter entsprechen gefundenen Hitting-Sets.

Die im Suchbaum umkreisten Blätter entsprechen äquivalenten Hitting-Sets, da die Reihenfolge keine Rolle spielt. Falls es noch weitere unausweichliche Mengen gegeben hätte, welche noch nicht getroffen wären, würden ganze äquivalente Teilbäume mehrfach besucht werden. Es ist also ersichtlich, dass dies einen beträchtlichen Mehraufwand bedeuten kann. Um diese Redundanz zu sparen werden bei einem Hinzufügen eines Elements aus einer noch nicht getroffenen Menge, alle Elemente aus dieser Menge, die kleiner als das ausgewählte Element sind, aus der weiteren Suche in diesem Teilbaum ausgeschlossen. In unserem Beispiel würden wir wenn wir die 2 in unsere Menge aufnehmen, die 1 innerhalb dieses Teilbaums ignorieren, was dazu führen würde, dass unser Duplikat im rechten Teilbaum nicht mehr aufgezählt wird.

Theorem 5. *Diese Beschränkung unseres Suchraums lässt sich durchführen, ohne dabei ein minimales Hitting-Set auszulassen.*

Beweis. Gegeben sei in einem Schritt des Algorithmus die bereits erstellte Menge X, welche noch nicht unbedingt ein Hitting-Set ist. Falls X noch kein Hitting-Set ist wird eine weitere noch nicht getroffene Menge ausgesucht und für alle n Elemente $X \cup E_i$ mit $i = \{0 \dots n\}$ und $E_0 \leq E_2 \cdots \leq E_n$ verzweigt. $X \cup Y$ ist ein mögliches minimales Hitting-Set, mit $Y \cap X = \emptyset$, welches in einem der n rechten Teilbäumen auftaucht (siehe Abbildung 10).

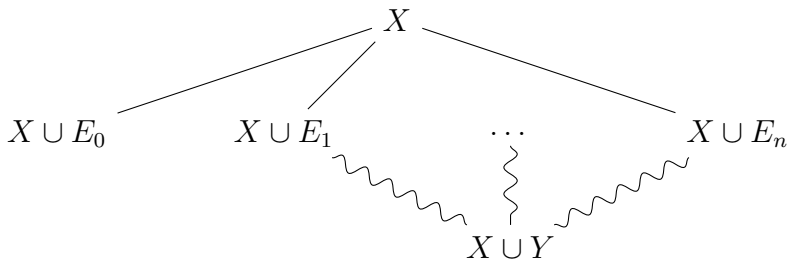

Abbildung 10: Schritt innerhalb des Backtracking Algorithmus, mit $X \cup Y$ einem minimalen Hitting-Set.

Wir werden nun zeigen, dass jedes minimale Hitting-Set einmal aufgezählt wird. In jedem der n Teilbäume werden wir durch unsere Einschränkung jeweils $\{E_1\}, \{E_1, E_2\}, \ldots, \{E_1, \ldots, E_{n-1}\}$ ignorieren. Angenommen wir würden auf Grund dieser Einschränkung ein minimales Hitting-Set $X \cup Y$ auslassen, dann würde sich mindestens eines der ausgeschlossenen E_i in Y befinden. Dies würde wiederum bedeuten, dass $X \cup Y$ sich zusätzlich auch in dem Teilbaum von $X \cup E_i$ mit $E_i \in Y$ befindet, da unser Algorithmus alle Teilmengen einer minimalen unausweichlichen Menge rekursiv zu dieser ausbaut (siehe Lemma 4). Falls dieses Exemplar jedoch ausgelassen wird lässt sich dasselbe Schema wieder auf den Teilbaum anwenden. Es lässt sich somit für jede Stufe zeigen, dass falls ein minimales Hitting-Set ignoriert wird, es in mindestens einem der anderen Teilbäume vorkommt. Dies lässt sich also von einer Tiefe von 0 beginnend, d.h. $X = \emptyset$, in deren Teilbäumen nach Lemma 4 alle minimalen Hitting-Set vorkommen, solange weiterführen bis der Algorithmus bei einer Tiefe von 16 angekommen ist und X das minimale Hitting-Set ist. □

Nun folgen noch zwei weitere Verbesserungen unseres Hitting-Set Algorithmus. Die erste beruht auf einer Form von höhergradigen unausweichlichen Mengen. Eine unausweichliche Menge n-ten Grades hat die Eigenschaft, dass ein Sudoku mindestens n Elemente aus dieser enthalten muss, um gültig zu sein.
Die minimalen unausweichlichen Mengen, welche mit den Mustern aus *checker* gefunden werden, sind alle 1-ten Grades. Höhergradige werden wir erstellen, indem wir mehrere disjunkte unausweichliche Mengen 1-ten vereinigen. Jedes Hitting-Set muss, um ein gültiges Sudoku zu sein, jede unausweichliche Menge treffen. Wenn also ein Hitting-Set nur weniger als n Elemente unserer n gradigen unausweichlichen Menge trifft wissen wir, dass mindestens eine disjunkte Menge, aus denen unsere unausweichliche Menge n-ten Grade aufgebaut ist, noch nicht getroffen sein kann.
McGuire et al. haben bei ihren Experimenten herausgefunden, dass es am

effizientesten ist, alle möglichen unausweichlichen Mengen bis zu Grad 5 aufzuzählen, welche aus gefundenen 1 gradigen Mengen zusammengesetzt werden können. Wenn der Backtracking Algorithmus in Tiefe 13 ist und wir noch wissen, dass eine unausweichliche Menge 4-ten Grades noch nicht getroffen ist, können wir abbrechen, da wir um zu einem Hitting-Set zu gelangen noch mindestens 4 Elemente hinzufügen müssten. Da wir aber nur an Hitting-Set mit Größe < 17 interessiert sind, können wir hier backtracken. Analog lässt sich mit unausweichlichen Mengen 2-ten, 3-ten oder 5-ten Grades verfahren.
Als letzte Verbesserung werden wir beim Auswählen der nächsten, noch nicht getroffenen Menge darauf achten, das mit der kleinsten Anzahl an Elementen auszuwählen. Dies hat zur Folge, dass der Suchbaum schrumpft. Eine Intuition warum dies so ist bekommt man am besten, wenn man sich vorstellt, dass es unausweichliche Mengen mit nur einem Element gibt, welche wenn man sie bevorzugt und zu Anfang des Algorithmus zur Menge hinzufügt zu einem kleinem Suchbaum führt, als wenn sie später im Baum in mehreren Teilbäumen hinzugefügt werden würden. Ähnlich verhält es sich mit Mengen mit mehr als einem Element.
McGuire et al. machen weiterhin die Variablenordnung von den bereits ignorierten Ziffern abhängig, welche wir zur Suchbaumverkleinerung eingeführt haben. Da Domänen, welche ignorierte Ziffern enthalten eigentlich kleiner sind.
In meiner Implementation hat sich jedoch heraus gestellt, dass eine statische Variablenordnung nach der Domänengröße zu Anfang (unabhängig von den ignorierten Ziffern) am effektivsten ist. Dies liegt eventuell daran, dass McGuire et al. eine effizientere Methode benutzt haben um die Variablen in Abhängigkeit der ignorierten Ziffern zu ordnen.

4.4 Sudoku-Solver

Takayuki and Takahiro konnten 2003 zeigen, dass das Lösen von allgemeinen $n \times n$ Sudokus NP-vollständig ist [10]. Dies spielt in unserem Fall aber keine große Rolle, da wir ein fixes n haben.
Damit wir einen Backtrackingalgorithmus anwenden können werden wir nun das Sudokulösen in ein Constraint-Network $C = \langle V, D, R\rangle$ transformieren [8, Kapitel 6]. V ist die Menge der Variablen. In unserem Fall entspricht dies der Menge $\{0 \dots 80\}$ der Zellen in einem Sudoku. D steht für Domäne und ist eine Funktion $V \to M \subseteq \{1 \dots 9\}$, welches die möglichen Belegungen unserer Sudokuzellen sind. Als letztes ist $R : V \times V \to \mathcal{P}(\{1 \dots 9\} \times \{1 \dots 9\})$ eine Relation zwischen den Zellen die mögliche Belegungen angibt. In unserem Fall ist $\forall v_1, v_2 \in R : \langle v_1, v_2\rangle \mapsto \{1 \dots 9\} \times \{1 \dots 9\}$, auch triviale Constraints genannt, falls v_1, v_2 *nicht* in derselben Spalte, Zeile oder 3×3 Block sind (for-

male Definition auf Seite 2). Andernfalls gilt $R : \langle v_1, v_2 \rangle \mapsto \{\langle d_1, d_2 \rangle | d_1, d_2 \in \{1, \ldots, 9\}$ und $d_1 \neq d_2\}$.
Unser Algorithmus bedient sich neben dem Backtracking einer einfachen vorausschauenden Interferenzmethode. Jedes mal wenn unser Algorithmus einem Feld eine Zahl zugewiesen hat entfernen wir diese Zahl von den Domänen der entsprechenden Felder in derselben Reihe, Spalte oder 3×3 Block. Somit gewährleisten wir, dass bei einer nächsten Zuweisung eines Feldes es zu keinem Konflikt kommt, d.h., dass unsere Variablenzuweisungen immer konform zu R sind. Damit erübrigt sich für unseren Algorithmus das Verwalten von R.
Als eine weitere Variante wollte ich mit der Kantenkonsistenz arbeiten, von der die vorausschauende Interferenzmethode eigentlich nur ein Spezialfall ist. Bei der Kantenkonsistenz wird für jedes mögliche Variablenpaar überprüft ob es zu einer Belegung der einen Variable eine passende Belegung der Anderen gibt, d.h. einen Eintrag in R. Falls es keine passende Belegung gibt, kann die entsprechende Variable aus der einen Domäne entfernt werden, da diese Belegung niemals zu einem vollständigen Sudoku erweitert werden könnte. Bei der vorausschauenden Interferenz wird ähnlich vorgegangen, jedoch nicht für sämtliche Variablenpaare sondern immer nur in Bezug auf die Variable, welche zuletzt gesetzt wurde.
Jedoch stellte sich im Nachhinein heraus, dass Kantenkonsistenz im Falle des Sudokulösens äquivalent zur vorausschauenden Interferenz ist. Durch die Kantenkonsistenz wird aus einer Domäne nur ein Element gestrichen falls es zu irgendeiner anderen Zelle in der gleichen Zeile, Spalte oder Box keine passende Belegung gibt. Dies ist allerdings nur genau dann der Fall, wenn die entsprechende Domäne der Zelle nur noch ein Element enthält. Denn sobald es zwei Elemente in der Menge gibt, ist immer ein Element darin, ungleich zu jeder Zahl. Wenn sich aber in einer Domäne bereits nur ein Element befindet und man mit einer dynamischen Variablenordnung arbeitet, die kleine Domänen bevorzugt, wird man dieses als nächstes auswählen und darauf vorausschauende Interferenz anwenden, welches wie bereits erwähnt Kantenkonsistenz garantiert.

5 Auswertung

In dieser Sektion werden die Resultate der Testläufe meiner Implementation ausgewertet. Zur effizienten Umsetzung der erwähnten Methoden aus der vorangegangenen Sektion verweise ich auf das Papier von McGuire et al. [6]. In dem Papier wird zu fast allen verwendeten Konzepten ein Pseudocode zur Implementation angegeben.
Die Testläufe wurden jeweils auf einem Kern des Intel Core i7-860 Prozessor ausgeführt.

5.1 Das Suchen der minimalen Vorgaben für 4×4 Sudokus

Bei der Suche nach den minimalen Vorgaben der 4×4 Sudokus lassen sich unsere Muster für die 9×9 Sudokus, wenn auch in abgewandelter Form, gut verwenden. Alle Muster für 4×4 Sudokus lassen sich, durch Anfügen von Zeilen und Spalten, in gültige Muster für 9×9 umwandeln lassen. Da unsere Musterliste für 9×9 Sudokus alle Muster für minimale unausweichlichen Mengen bis maximal Größe 12 beinhaltet, befinden sich unter diesen auch die entsprechenden Muster für minimale unausweichliche Mengen bis Größe 12 in 4×4 Sudokus.
Bevor wir diese jedoch auf 4×4 Sudokus anwenden können müssen wir sie auf entsprechende Größe kürzen. Dies funktioniert allerdings nur mit Mustern, welche in mehreren Zeilen und Spalten keine Elemente enthalten, sodass sich diese zusammenschrumpfen lassen (andernfalls würden diese auch gar nicht auf 4×4 Sudokus zutreffen).
Um zu allen gelösten 4×4 Sudokus zu kommen, werden wir den Sudoku-Solver verwenden indem wir ihn ein leeres 4×4 Sudoku übergeben und er dann entsprechend alle möglichen Vervollständigungen sucht. Man beachte, dass in den hierbei erstellten Sudokus auch Äquivalente vorkommen. Dies führt allerdings zu keinen Zeitproblemen, wegen der geringen Anzahl an Lösungen und der kurzen Laufzeit.
Bei meinen Berechnungen stellte sich heraus, dass die minimale Anzahl an Vorgaben bei 4 liegt. Um dies zu zeigen musste ich zuerst ein Sudoku finden, welches nur 4 Vorgaben besitzt (siehe Abbildung 11).

1		3	
	4		2

Abbildung 11: Gültiges 4×4 Sudoku mit nur 4 Vorgaben.

Der Sudoku-Solver konnte nach Eingabe eines leeren 4×4 Sudoku 144 Lösungen finden. In jedem dieser Lösungen wurde dann nach gültigen Sudokus mit 3 Vorgaben gesucht. Die meiste Rechenzeit wurde in dem Finden von höhergradigen unausweichlichen Mengen verbracht. In allen 144 Lösungen konnte mindestens eine unausweichliche Menge 4-ten Grades gefunden werden. Dadurch konnte der Backtracking Algorithmus direkt zu Beginn der Hitting-Set Suche backtracken. Entsprechend wurde auch in keinem der Fälle ein gültiges Sudoku mit nur 3 Vorgaben gefunden. Die durchschnittliche Rechenzeit pro Lösung betrug 11 ms.

5.2 Durchsuchen von gelösten 9×9 Sudokus, nach gültigen Sudokus mit 17 Vorgaben

Bei meinem Durchlauf habe ich die ersten hundert Sudokus aus Gordons Liste von gültigen Sudokus mit 17 Vorgaben verwendet [4]. Ich habe sie vorerst mit dem Sudoku-Solver gelöst und dann in den Lösungen nach darin enthaltenen gültigen Sudokus mit 16 Vorgaben gesucht. Die Lösungen scheinen sich intuitiv für einen Suchlauf zu eignen, da sich in diesen bereits ein Sudoku mit nur 17 Vorgaben befindet.
Es wurde in keinem der 100 Lösungen ein gültiges 16-Vorgaben-Sudoku gefunden. Die durchschnittliche Suchdauer innerhalb einer Lösung beträgt 2961 s. Dies ist etwas weniger als 50 Minuten. McGuire et al. optimierte Endversion brauchte im Vergleich nur etwa 3,6 s.
Es wurden durchschnittlich 322 unausweichliche Mengen gefunden. Interessanterweise scheint die Anzahl der gefundenen unausweichlichen Mengen keine große Rolle bei der Laufzeit zu spielen. Die Lösung mit der längsten Durchsuchungszeit (23061 s) liegt mit 339 darin gefundenen unausweichlichen Mengen relativ nah am arithmetischen Mittel. Die Differenz zum Mittel entspricht hierbei in etwa der halben Standardabweichung.
Auffällig beim längsten Suchlauf ist weiterhin, dass der Algorithmus knapp 80 % der aufgewendeten Zeit überprüft, ob ein gefundenes Hitting-Set der Größe 16 zu einem gültigen Sudoku gehört. Während im Durchschnitt lediglich ungefähr 28 % der Zeit zum Überprüfen der Hitting-Sets aufgewendet

wird.
Ich vermute, dass sich bei meiner Implementation des Sudoku-Solver auch noch an Performance gewinnen lässt, wenn man mit weniger komplexen Datenstrukturen arbeitet.
Bei dem Überprüfen, ob ein gefundenes Hitting-Set mit 16 Elementen zu einem gültigen Sudoku korrespondiert, reicht es aus zwei Lösungen zu finden und man hat gezeigt, dass das Sudoku nicht gültig ist. Eine Verbesserung des Algorithmus hin zum schnelleren Finden von Lösungen könnte eine Ordnung sein mit der man die Werte der Variablendomänen durchgeht. Man könnte beispielsweise die Auswahl der Werte der Variablen danach sortieren wie viele Elemente bei einem Forward Checking entfernt würden.
Der Spitzenwert des Speicherverbrauchs lag durchschnittlich bei ungefähr 27 MiB. Wobei durchschnittlich ca. 79 % des gebrauchten Speichers für die Verwaltung der unausweichlichen Mengen 2., 3., 4. und 5. Grades gebraucht wurden.

6 Fazit

In diesem Papier konnten sämtliche Methoden von McGuire et al. bestätigt werden.
Es konnte weiterhin berechnet werden, dass es keine gültigen 4×4 Sudokus mit weniger als 4 Vorgaben gibt.
Außerdem wurde berechnet, dass die Lösungen der ersten 100 Sudokus aus Gordons Liste keine gültigen Sudokus mit Größe 16 enthalten.
Der hierbei verwendete Backtracking Algorithmus lässt sich sehr gut auf alle Hitting-Set Probleme anwenden, bei denen es um eine vollständige Suche geht.
Was ich selber von diesem Projekt mitnehmen werde ist, dass man sich nicht einschüchtern lassen sollte von der anfänglichen Größe des Problems. Durch langanhaltenden Optimierungskampf (2006-2011) haben McGuire et al. es geschafft das vorerst nicht durchführbare Projekt an vielen Stellen zu optimieren, bis es am Schluss zum vollständigen Beweis innerhalb eines Jahres reichte.

Literatur

[1] Sudoku article on Wikipedia. https://de.wikipedia.org/wiki/Sudoku. Accessed in: August 2016.

[2] Bertram Felgenhauer and Frazer Jarvis. Mathematics of Sudoku I. *Mathematical Spectrum*, 39(1):15–22, 2006.

[3] Glenn Fowler. sudoku (program) website. http://gsf.cococlyde.org/download/sudoku. Accessed in: August 2016.

[4] Royle Gordon. 17 clue Sudokus website. Institution: The University of Western Australia. http://staffhome.ecm.uwa.edu.au/~00013890/sudokumin.php. Accessed in: August 2016. The list is licenced under Creative Commons Attribution 2.5 License: https://creativecommons.org/licenses/by/2.5/.

[5] Gary McGuire. checker website. http://www.math.ie/checker.html. Accessed in: August 2016.

[6] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue Sudoku: solving the Sudoku minimum number of clues problem via hitting set enumeration. *Experimental Mathematics*, 23(2):190–217, 2014.

[7] Ed Russell and Frazer Jarvis. Mathematics of Sudoku II. *Mathematical Spectrum*, 39(2):54–58, 2006.

[8] Stuart Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach*. 3rd edition, 2010.

[9] John Slaney. Set-theoretic duality: A fundamental feature of combinatorial optimisation. In *proceedings of the 21. European Conference on Artificial Intelligence (ECAI)*, pages 843–848, 2014.

[10] Yato Takayuki and Seta Takahiro. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.

Declaration on Scientific Integrity
Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor
Philipp Oldenburg

Matriculation number — Matrikelnummer
2013-061-064

Title of work — Titel der Arbeit
Gibt es Sudokus mit nur 16 Vorgaben?

Type of work — Typ der Arbeit
Bachelorarbeit

Declaration — Erklärung
I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognised scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, 14. August 2016

Philipp Oldenburg

Signature — Unterschrift