Klassifikation der Schwierigkeitsgrade von Sudokus mit Methoden des maschinellen Lernens

Classification of Sudoku difficulty levels usining methods of machine learning
Bachelor-Thesis von Michael Bräunlein

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Informatik
Knowledge Engineering Group
Betreuer Prof. Dr. Johannes Fürnkranz

Zusammenfassung

bla

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Das Zahlenrätsel Sudoku ist weltweit bei Rätsel-Liebhabern bekannt und beliebt. Nach seiner Veröffentlichung 1979 dauerte es bis 1986, bis sich Sudoku wie ein Lauffeuer über den Globus verbreitete. Seitdem begeistern sich immer mehr Menschen für mehr oder weniger schwierige Exemplare. Sudokus finden sich im Internet, in der Rätselecke der Tageszeitungen und sogar als ganze Bücher, um nur einige Erscheinungsorte zu nennen.
Laut [5] kommt das Wort Sudoku ursprünglich aus dem japanischen, 'Su' bedeutet 'Zahl' und 'Doku' bedeutet 'einzeln'. Weiter liest man dort, dass der Grundstein zu Sudokus von dem Mathematiker Leonhard Euler gelegt worden sei. Das Puzzle sei über Frankreich und Amerika nach Japan gewandert, wobei es sich kontinuierlich von lateinischen Quadraten[1] zu dem heute bekannten Sudoku entwickelt habe. Die erste Veröffentlichung von Sudoku erfolgte im Mai 1979 in der Zeitschrift 'Dell Pencil Puzzles & Word Games', allerdings unter dem Namen 'Number Place', wie man [7] entnehmen kann.
Die Regeln sind sehr simpel, denn es gibt nur eine. Sudoku ist daher sehr einfach zu lernen. Dennoch kann ein Spieler nicht jedes Sudoku auf Anhieb lösen, da für manche Sudokus sehr komplexe Techniken nötig sind. Das Erlernen der Techniken und das Finden der benötigten Muster in den Zahlen stellt beim Meistern des Spiels die größte Herausforderung dar. Manche Spieler versuchen auch, Sudokus möglichst schnell zu lösen.
Verschiedene Sudokus werden von Spielern als unterschiedlich schwer empfunden, daher ist annähernd jedes veröffentliche Sudoku mit einem Schwierigkeitsgrad versehen. Je nachdem, wo Sudokus auftauchen unterscheiden sich auch die Schwierigkeitsgrade, obwohl sie manchmal sogar den selben Namen haben. Das liegt daran, dass es kein Vereinheitlichstes Vorgehen bei der Bewertung des Schwierigkeitsgrades gibt. Daher haben sich eine Menge unterschiedliche Vorgehensweisen und Bewertungsskalen entwickelt. Manche Skalen haben nur die Einträge 'Leicht', 'Mittel' und 'Schwer', andere unterteilen Sudokus in sieben verschiedene Schwierigkeitsstufen.
Der Spielspaß ist aber stark davon abhängig, dass ein Spieler die zu ihm passende Schwierigkeitsstufe findet. Wenn ein Spieler ein zu leichtes Sudoku löst, dann stellt es keine Herausforderung dar und er hat kein Erfolgserlebniss, wenn er fertig ist. Ist das Sudoku allerdings zu schwer, dann kommt schnell ein Gefühl der Frustration auf, da keine Zahlen gefunden werden, die in das Spielfeld eingetragen werden können.
Die verschiedenen Skalen der unterschiedlichen Veröffentlichungsorte machen einen Wechsel zum Beispiel von der Tageszeitung zu einem Buch unnötig kompliziert. Spieler müssen wieder den zu ihnen passenden Schwierigkeitsgrad finden. Daher befasst sich diese Arbeit unter anderem damit, wie zwei Skalen miteinander verglichen werden können.
Sudokus sind für die Bearbeitung mit Computern sehr gut geeignet. Da das Spielfeld aus 81 Felder besteht, die jeweils 10 verschiedene Einträge haben können[2], ist die interne Repräsentation in einem Computerprogramm sehr leicht. Weiterhin gibt es keine anderen Spieler, keinen Zufall und die Informationen über das Spiel sind immer vollständig.
Der neue Ansatz dieser Arbeit ist das Einteilen von Sudokus in Schwierigkeitsgrade mit Methoden des maschinellen Lernens. Dabei soll eine Methode entwickelt werden, die aus einem Sudoku Informationen gewinnt und mit Hilfe dieser Informationen eine Aussage über den Schwierigkeitsgrad treffen kann. Ausserdem soll festgestellt werden, welche Informationen zur Einteilung relevanter sind als andere.

[1] http://www.statistik.tuwien.ac.at/public/dutt/vorles/statistII_10/compstat/node33.html
[2] leer oder eine der Ziffern 1 bis 9

1.2 Struktur

Nach der Einleitung in Kapitel 1 werde ich einen Überblick über die Grundlagen geben, auf denen diese Arbeit aufbaut. Zuerst werde ich in Kapitel 2.1 auf die Regeln für Sudokus eingehen. Das beinhaltet die Regeln, die der Spieler beachten muss und auch die Regeln, die ein Sduoku erfüllen muss, um als Sudoku zu gelten.
Anschließend werden in Kapitel 2.2 einige Begriffe definiert, die im weiteren Verlauf der Arbeit auftauchen werden.
Danach werde ich in Kapitel 2.3 beschreiben, welche Ansätze es zur Ermittlung des Schwierigkeitsgrades bereits gibt und auf deren Vor- und Nachteile eingehen.
Da, für die in dieser Arbeit vorgestellte Methode, Lösungstechniken für Sudokus essentiell sind, werde ich in Kapitel 2.4 die verwendeten Lösungsmethoden erklären. Sie sind innerhalb des Kapitels nach ihrer Schwierigkeit geordnet. Das sind nicht alle existierenden Lösungsmethoden, aber Weitere wurden im Rahmen dieser Arbeit nicht benötigt.
In Kapitel 3 möchte ich die von mir entwickelte Methode vorstellen. Dazu werde ich in Kapitel 3.1 darauf eingehen, was Klassifikation im Kontext von maschinellem Lernen bedeutet und wozu Klassifikation in der Methode benötigt wird.
Anschließend wird in Kapitel **??** erklärt, was ein Featurevector ist und wozu er in dieser Arbeit verwendet wird.
In Kapitel 3.2 werde ich darauf eingehen, wie genau der Featurevector aus dem Sudoku gewonnen wird und welche Eintrage darin enthalten sind.
Das Kapitel 3.3 beschäftigt sich mit einer sehr nützlichen Optimierung des Featurevectors, die speziell Sudokus betrifft und die im Rahmen dieser Arbeit entwickelt wurde.
Um die entwickelte Methode zu testen, wurde Software entwickelt, die in Kapitel 4.1 beschrieben wird. Da einige Funktionalität bereits in open source Projekten implementiert wurde, wurden diese Projekte bei der Entwicklung der Software verwendet. Welchen Zweck diese Projekte innerhalb des selbst entwickelten Programms erfüllt, wird in Kapitel 4.2 erklärt.
Um die Methode zu testen, werden neben einer Implementierung auch Testdaten benötigt. Welche Testdaten verwendet wurden und woher diese stammen, wird in Kapitel 4.3 erklärt.
Die Ergebnisse der Tests, werde ich in Kapitel **??** vorstellen. Dazu zählen die Güte der Klassifikation, der Einfluss der Parameter der Klassifiikationsalgorithmus und der Einfluss der verwendeten Lösungsmethoden. Weiterhin werde ich in diesem Kapitel auf die Ergebnisse der Methode zum Vergleich verschiedener Bewertungsskalen für den Schwierigkeitsgrad von Sudokus eingehen.
Das letzte Kapitel (6) fasst die Arbeit zusammen und gibt einen Ausblick auf weitere, mögliche Forschung.

2 Grundlagen

2.1 Die Regeln

Diese Arbeit beschäftigt sich nur mit der meist verbreiteten Art von Sudokus. Dieses Sudoku spielt man auf einem Spielfeld der Größe 9x9. Das beudeutet, dass jede Seite des Sudokus 9 Felder lang ist. Das Spielfeld ist in 9 Blöcke der Größe 3x3 unterteilt. Es hat ausserdem 9 Zeilen und 9 Spalten. Insgesamt besteht das Sudokus also aus 81 Felder. In das Sudoku werden im Spielverlauf Ziffern eingetragen, so dass am Ende des Spiels jedes Feld gefüllt ist. Dabei gilt ein Sudoku als gelöst, wenn in jeder Zeile, in jeder Spalte und in jedem Block die Ziffern 1 bis 9 jeweils einmal vorkommen. Das ist die einzige Regel des Sudokuspiels.

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

Abbildung 2.1: Sudoku

In **Abbildung 2.1** sieht man ein unfertiges Sudoku. Im mittleren Block kann man sehen, dass in einigen Feldern mehrere kleine Zahlen eingetragen sind. Diese dienen als Gedankenstütze des Spielers und helfen ihm beim Finden komplexer Muster, die für einige Lösungstechniken benötigt werden. Das notieren dieser Zahlen ist allerdings keine Pflicht.
Wichtig ist bei Sudokus, dass ein Sudoku nur als solches gilt, wenn es genau eine Lösung hat. Es sind keine Anordnungen der Zahlen erlaubt, die mehrere Lösungen zulassen, oder die schon von Anfang an im Widerspruch zur Sudokuregel stehen oder unvermeidbar zu einem Widerspruch führen.

2.2 Begriffserklärung

Ein Sudoku besteht aus 81 *Feldern* oder *Zellen*. In diese werden die *Ziffern* oder *Zahlen* eingetragen und sie bilden ein Quadrat der Größe 9x9, das *Grid*. Aufgrund dieser Aufteilung hat ein Sudoku 9 *Zeilen* und 9 *Spalten*. Das Grid wird in 9 Unterquadrate geteilt, die jeweils 3x3 Felder groß sind. Diese werden *Blöcke* genannt. Zeilen, Spalten und Blöcke werden unter dem Begriff *Figur* zusammengefasst. Die Nummerierung der Blöcke erfolgt zeilenweise von links oben nach rechts unten.
Vorgaben sind Zahlen, die schon von Anfang an gegeben sind.

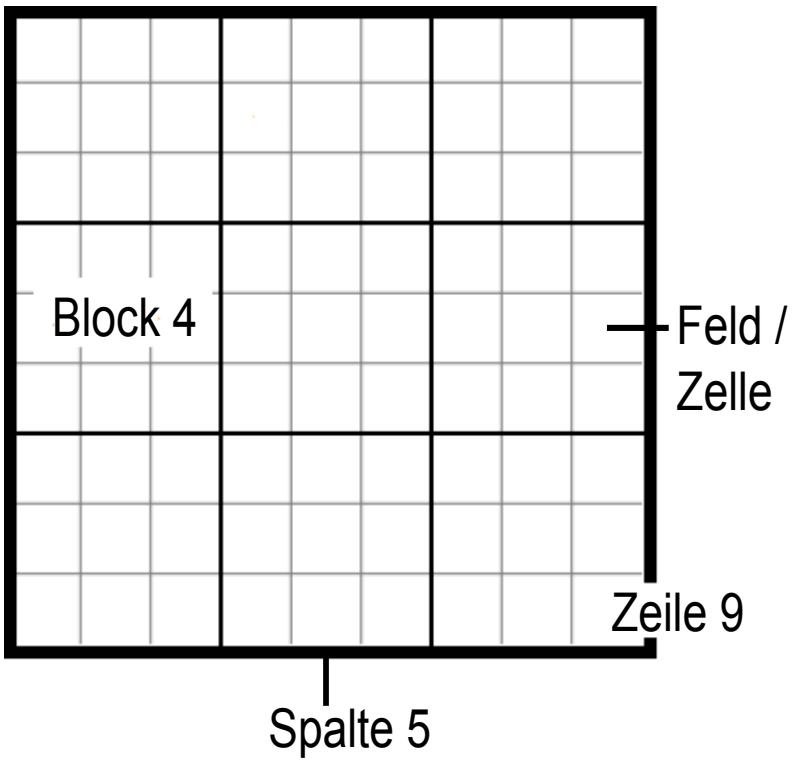

Abbildung 2.2: Sudoku

In **Abbildung 2.2** sieht man im mittleren Block sogenannte *Kandidaten*. Ein Kandidat ist eine Zahl, die in der Zelle noch möglich ist. Jede Zelle hat ihre eigene Liste mit Kandidaten.
In der Beschreibung der Lösungstechniken ist es notwendig bestimmte Felder zu betrachten. Hierzu wird eine Abkürzung verwendet, die Zeile und Spalte enthält und somit eine Zelle eindeutig indentifiziert. *z2s3* meint zum Beispiel die Zelle in Zeile 2 und Spalte 3.
In der folgenden Abbildung sind die erläuterten Begriffe zum besseren Verständniss eingetragen.

2.3 Existierende Einteilungsschemata

Für die Zuordnung eines Schwierigkeitsgrades zu einem Sudoku existieren bereits verschiedene Methoden. Die einfachsten orrientieren sich nur an Merkmalen, die auf den ersten Blick sichtbar sind, wie etwa die Anzahl der vorgegebenen Ziffern. Diese Methoden haben den Vorteil, dass das Sudoku bewertet werden kann, ohne es zu lösen. Ausserdem sind sie sehr einfach zu implementieren und auch sehr schnell. Die Genauigkeit solcher Methoden ist allerdings nicht besonders hoch, da die offensichtlichen Merkmale nicht mit einbeziehen, wie schwer das Lösen eines Sudokus ist, da es ja während der Bewertung des Sudokus nicht gelöst wird und somit dem Lösungsweg kein Schwierigkeitsgrad zugeordnet werden kann.
Daher ist es sinnvoll, das Sudoku bei der Bewertung zu lösen und die Schwierigkeit des Lösungsweges zu schätzen. Ein Lösungsweg besteht aus einer Reihe von Lösungsschritten. Mit jedem Schritt erfährt der Spieler mehr über das Sudoku, das bedeutet, dass er eine Ziffer einsetzen kann oder dass er ausschließt, dass eine Ziffer in einem bestimmten Feld stehen kann. Gegebenenfalls kann er mit einem Lösungsschritt sogar mehrere Ziffern in einem Feld, oder eine Ziffer in mehreren Felder ausschließen. Sehr komplexe Methoden erlauben auch den gleichzeitigen Ausschluss von mehreren Ziffern in mehreren Feldern.
Es gibt also verschiedene Arten von Lösungsschritten, diese haben auch eine verschiedene Schwierigkeit. Dabei entscheidend ist für diese Arbeit die vom menschlichen Spieler empfundene Schwierigkeit und nicht etwa die Komplexität der Implementierung oder die Laufzeit der Lösungsmethode auf einem Rechner, da die Bewertung später als Orrientierung für Menschen dienen soll.
Die Website `http://sudoku.soeinding.de/strategie/strategie03.php` ist ein Beispiel für ein solches Vorgehen.
Eine weitere Methode ist die Messung der Zeit, die menschliche Spieler zum Lösen von Sudokus brauchen. Hierbei ist das Problem, dass die Klassifikation eines Sudokus sehr lange dauert, da es erst von mehreren Spielern gelöst werden muss. Ausserdem ist die Erfahrung und damit die Spielstärke von Sudoku Spielern sher unterschiedlich und sollte in die Bewertung mit einfließen.
Ein Beispiel für diesen Ansatz der Klassifikation ist diese Website `http://sudokugarden.de/de/online/dstats`.
In dieser Arbeit wird versucht, ein Einteilungsschma zu finden, dass auf der Schwierigkeit des Lösungsweges beruht, aber auch offensichtliche Merkmale einbezieht.

2.4 Lösungsmethoden

Alle in dieser Bachelorarbeit beschriebenen Techniken sind nicht im Rahmen dieser Arbeit entwickelt worden, sondern wurden aus verschiedenen Quellen zusammengetragen. Die Beschreibung der Lösungstechniken lehnt sich an die Beschreibung der Quellen an. Teile der Beispiele wurden aus den Quellen entnommen, dies ist entsprechend gekennzeichnet.
Grob kann man die Techniken zum Lösen von Sudokus in zwei Kategorien einteilen. Die erste Kategorie findet Zahlen heraus, die direkt in das Sudoku eingetragen werden können. Die Techniken der zweiten Kategorie entfernen Bedingungen in einzelnen Zellen des Sudokus.

2.4.1 Kandidatenlisten

Beim Lösen von Sudokus ist es üblich, in jedes Feld die Kandidaten einzutragen, die dort stehen können. Dabei wird vorerst nur die Sudoku Regel berücksichtigt, die besagt, dass in jeder Figure die Zahlen 1 bis 9 vorkommen müssen. Wenn in einer Zeile nun die Zahl 3 vorkommt, dann kann sie in der selben Zeile nicht nochmal vorkommen, daher kann sie aus allen Kandidatenlisten der Zellen in der selben Zeile gelöscht werden. Dasselbe gilt für Spalten und Blöcke. Immer, wenn eine Ziffer in ein Feld eingetragen wird, dann muss der Spieler die Liste der Kandidaten aktualisieren.
Kandidatenlisten sind keine eigene Lösungstechnik, sind aber wesentlicher Bestandteil vieler Techniken.

2.4.2 Full House

Wenn in einer Figur 8 Zahlen eingetragen sind, dann kann die Technik *Full House* angewendet werden. Da in jeder Figur die Zahlen 1 bis 9 stehen müssen, kann die fehlende Zahl einfach per Ausschluss ermittelt werden.

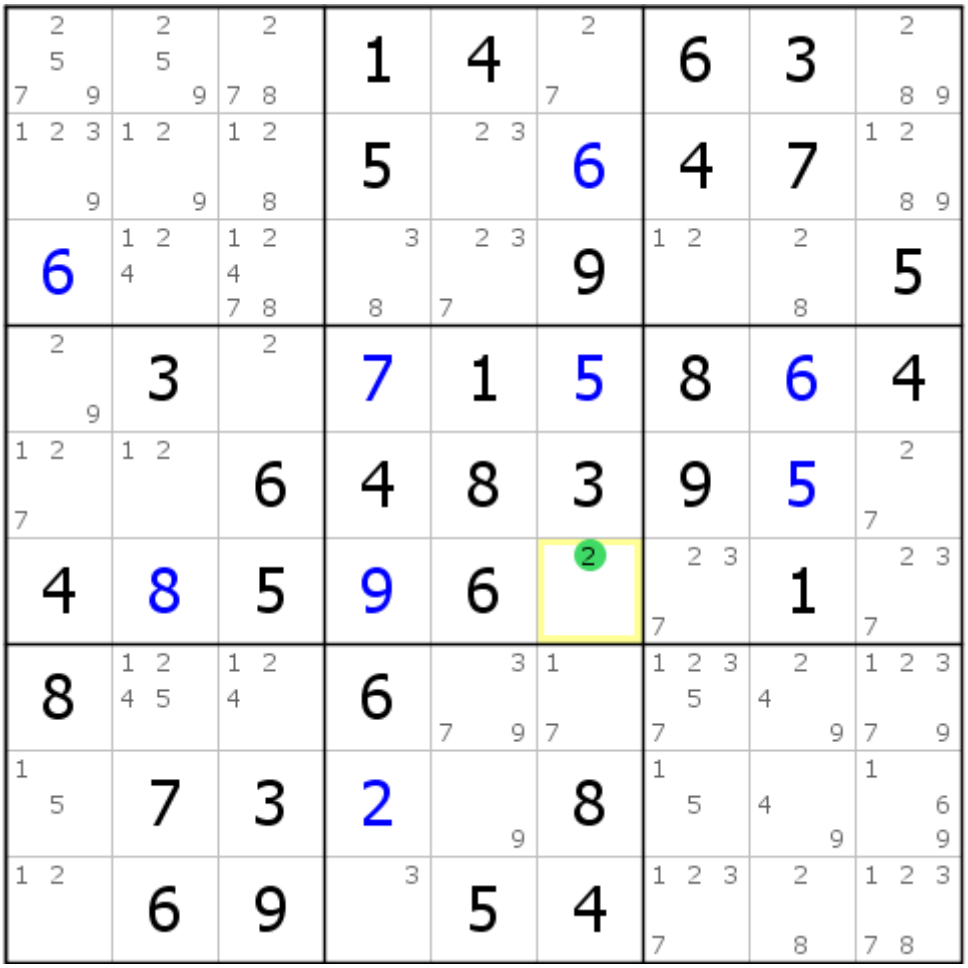

Abbildung 2.3: Full House

In **Abbildung 2.3** fehlt im fünften Block nur noch die Ziffer 2. Alle anderen Ziffern sind bereits in diesem Block eingetragen. Die Sudokuregel besagt, dass in jedem Block die Ziffern 1 bis 9 jeweils einmal vorkommen muss. Daher muss in z6s6 die Ziffer 2 stehen.

2.4.3 Naked Single

Bei der Technik *Naked Single* werden Kandidatenlisten verwendet. Diese Technik kann angewendet werden, wenn in der Kandidatenliste einer Zelle nur noch ein Kandidat steht. Dieser Kandidat kann dann in die Zelle eingetragen werden. Das funktioniert aufgrund des Aufbaus der Kandidatenlisten. Diese enthalten zuerst alle Kandidaten und es werden immer dann Kandidaten entfernt, wenn dieser Kandidat nicht mehr als Ziffer in der Zelle stehen könnte wel er dort eine Regel verletzen würde. Wenn also nur noch ein Kandidat in der Kandidatenliste steht, dann bedeutet das, dass dieser Kandidat die einzige Ziffer zwischen 1 und 9 ist, die in der Zelle stehen kann ohne eine Regel zu verletzen.

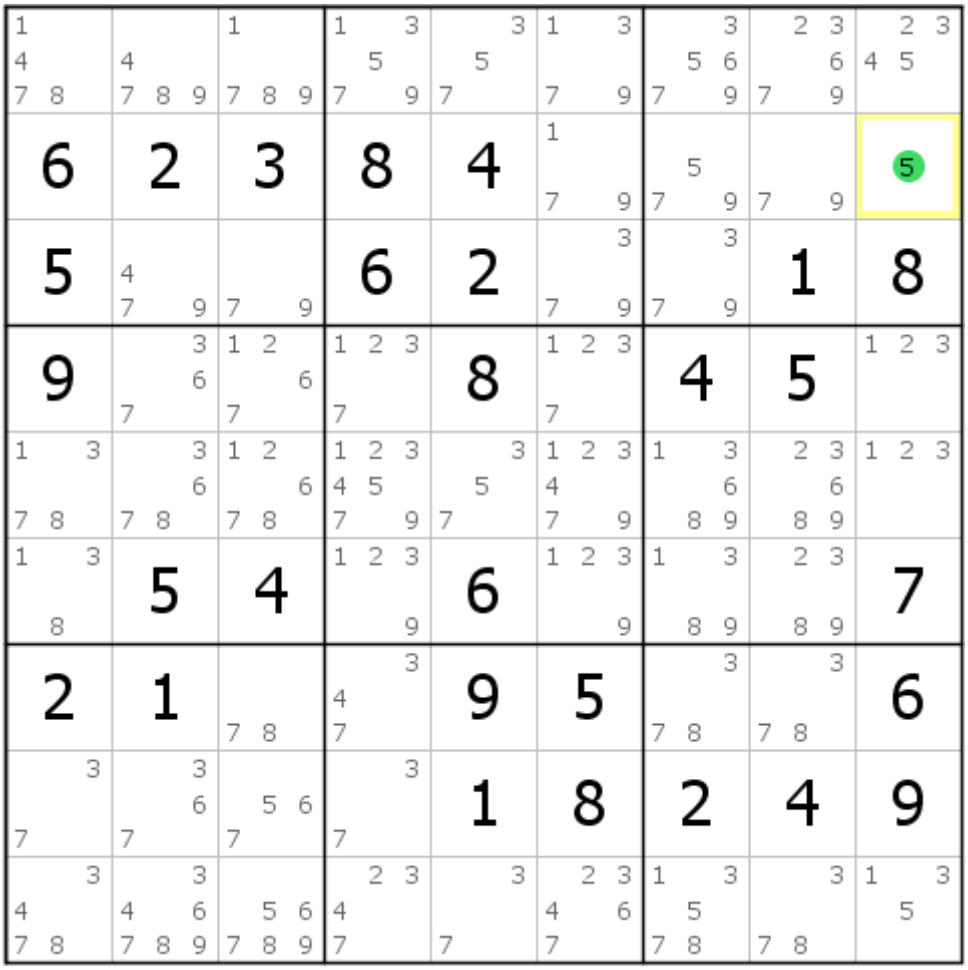

Abbildung 2.4: Naked Single

Im oben stehenden Beispiel **Abbildung 2.4** sieht man sofort, dass die Kandidatenliste in z2s9 nur noch einen Eintrag enthält. Dieser kann nun einfach eingetragen werden. Alle anderen Möglichkeiten dieses Feldes werden durch die Zeile, Spalte und den Block ausgeschlossen, in denen die Zelle enthalten ist, da in jeder dieser Figuren eine Ziffer nicht zweimal vorkommen darf.

2.4.4 Hidden Single

Auch die Technck *Hidden Single* arbeitet wieder mit Kandidatenlisten. Wenn in einer Figur eine Kandidatenliste die einzige ist, in der eine bestimmte Zahl vorkommt, dann kann diese Zahl direkt in die Zelle eingetragen werden. Wenn in dieser Zelle die Zahl nicht stünde, dann gäbe es in der Figur keine Möglichkeit mehr, dass die Zahl auftaucht und damit wäre die Sudoku Regel verletzt, nach der jede Zahl genau einmal enthalten sein muss.

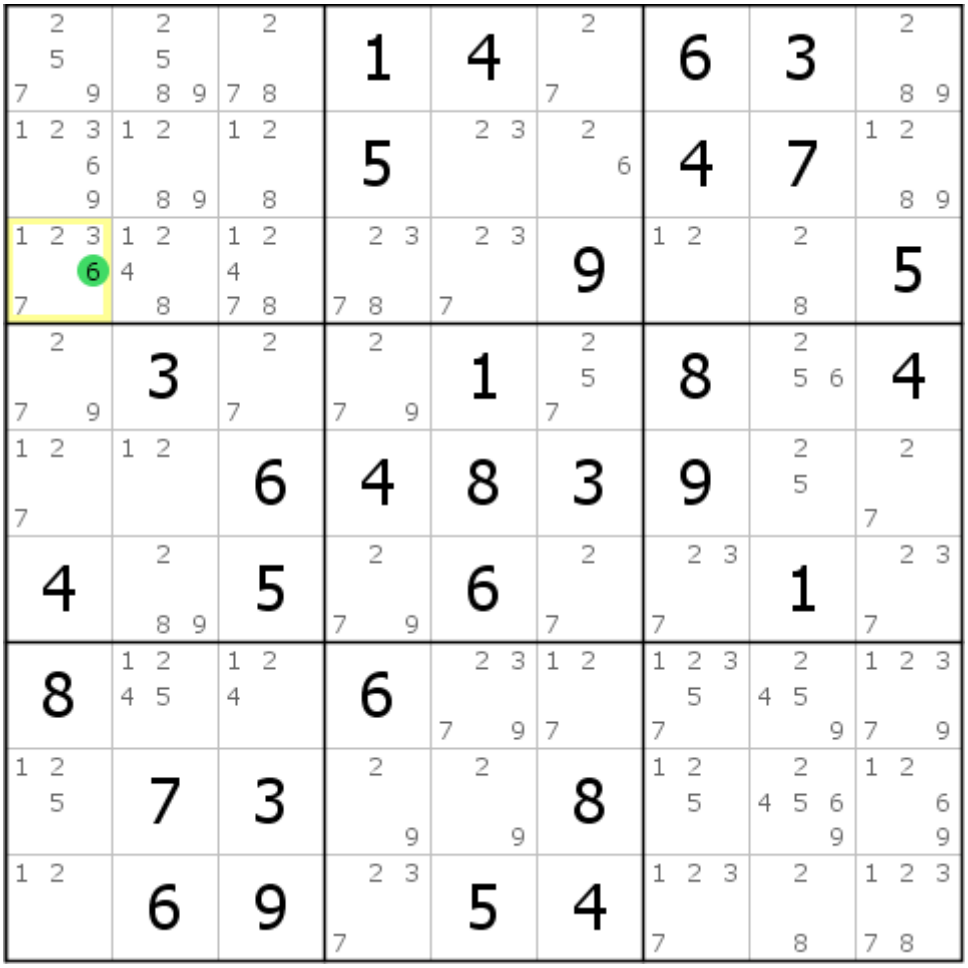

Abbildung 2.5: Hidden Single

In **Abbildung 2.5** sieht man, dass die Ziffer 6 in der Zeile 3 nur in z3s1 vorkommen kann. Da in Zeile 3 die Ziffer 6 genau einmal vorkommen kann und sie nur an einer Position in dieser Zeile stehen kann, bleibt keine andere Möglichkeit, als sie dort einzutragen.

2.4.5 Pointing Pair / Triple

Bei der Technik *Pointing Pair / Triple* müssen zum ersten mal die Kandidatenlisten mehrerer Felder gleichzeitig betrachtet werden, was diese Technik etwas schwerer macht. Ausserdem ist diese Technik die erste, die Kandidaten aus Kandidatenlisten entfernt und nur bedingt zum Einsetzen von Zahlen in das Sudoku führt.
Es werden die Kandidatenlisten in Blöcken jeweils zeilen- und spaltenweise betrachtet. Die Technik *Pointing Pair / Triple* kann angewendet werden, wenn in einem Block eine Kandidat nur in Kandidatenlisten der selben Zeile oder Spalte vorkommt. Dann kann jedes weitere vorkommen der Zahl in einer Kandidatenliste der selben Zeile oder Spalte entfernt werden. Das gilt, da die Zahl genau einmal in dem Block vorkommen muss. Da alle möglichen Vorkommen der Zahl in der selben Zeile oder Spalte liegen ist klar, dass die Zahl in dieser Zeile oder Spalte vorkommt. Da sie aber kein zweites mal in der Zeile oder Spalte vorkommen darf muss sie aus den Kandidatenlisten entfernt werden, die nicht im selben Block liegen.

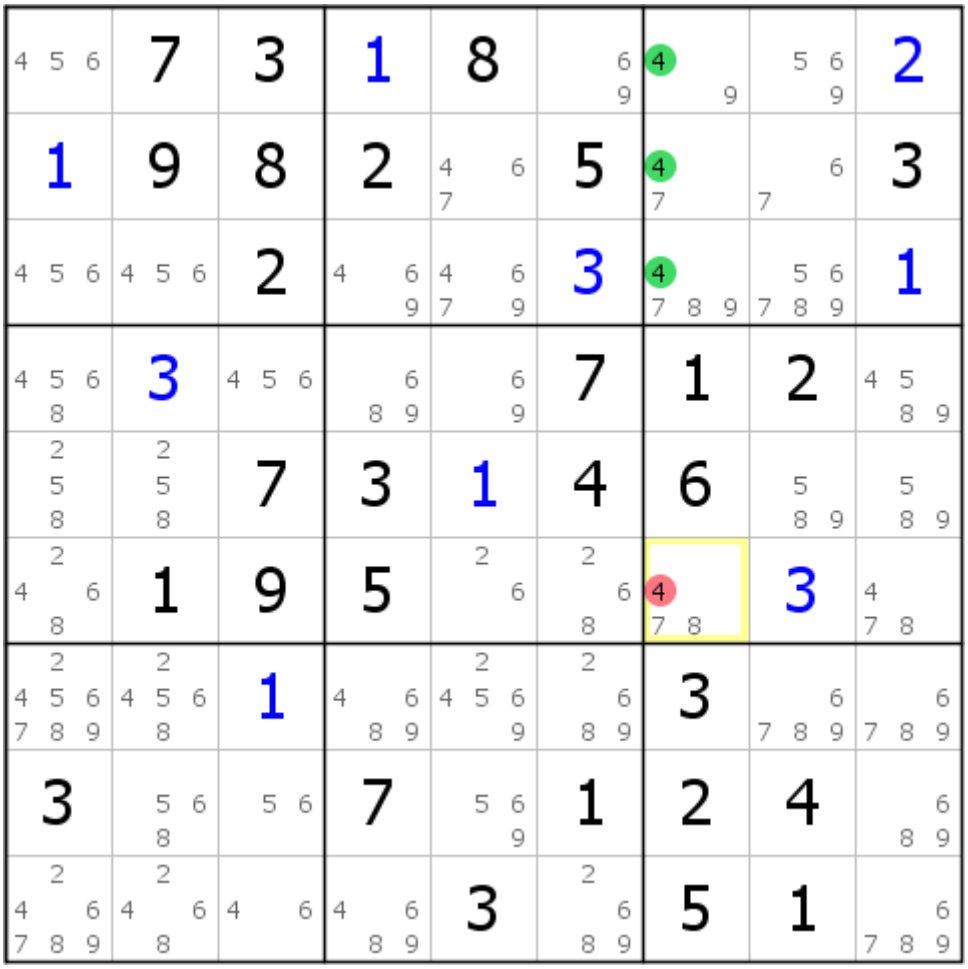

Abbildung 2.6: Pointing Triple

In **Abbildung 2.6** betrachten wir Block 3. Hier ist das Vorkommen der Zahl 4 in den Kandidatenlisten auf Spalte 7 beschränkt. Wie oben beschrieben können nun alle weiteren Vorkommen in der selben Spalte, die nicht in Block 8 liegen aus den Kandidatenlisten entfernt werden. Diese sind in der Abbildung rot markiert. Das führt nicht dazu, dass eine neue Zahl in das Sudoku eingetragen wird. Dennoch ist das Sudoku nun genauer bestimmt, da weniger Möglichkeiten übrig sind.

2.4.6 Box-Line Reduction

Die Technik *Box-Line-Reduction* ist verwandt mit der Technik *Pointing Pair / Triple*. Hier wird das Sudoku zeilen- und spaltenweise betrachtet. Ist das Vorkommen einer Zahl in den Kandidatenlisten auf einen Block beschränkt, dann kann jedes weitere Vorkommen der Zahl aus den Kandidatenlisten der Zellen des selben Blocks gestrichen werden, die nicht in der Zeile oder Spalte liegen. Die Begründung dafür ist ähnlich der Begründung bei *Pointing Pair / Triple*. Da die Zahlen 1 bis 9 jeweils genau einmal in der Zeile oder Spalte vorkommen müssen und dieses Vorkommen auf einen Block beschränkt ist, ist klar, dass die Zahl letztendlich in diesem Block vorkommt und zwar in der gefundenen Zeile oder Spalte. Die Zahl kann aber nicht zweimal in dem Block vorkommen, daher kann sie aus den Kandidatenlisten des Blocks gelöscht werden, deren Zellen sich nicht in der Reihe oder Spalte befinden.

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

Abbildung 2.7: Box-Line Reduction

Wir betrachten Spalte 7 in **Abbildung 2.7**. Hier sieht man, dass das Vorkommen der Ziffer 9 in dieser Spalte auf Block 3, den oberen Block, beschränkt ist. Anhand dieser Spalte sieht man also, dass die Ziffer 9 entweder in z1s7 oder in z3s7 steht, also in jedem Fall in Block 3. Daher kann die Ziffer 9 aus den Kandidatenlisten der anderen Zellen in Block 3 gestrichen werden.

2.4.7 Naked Subset

Die Technik *Naked Subset* ist ein Überbegriff für die Techniken *Naked Pair*, *Naked Triple* und *Naked Quadruple*. Alle Techniken Arbeiten nach dem selben Prinzip, der Unterschied liegt in der Anzahl der verwendeten Kandidatenlisten. Bei *NakedSubsets* sucht man nach Paaren, Tripeln oder Quadrupeln von Zellen in Figuren, nach Kandidatenlisten einer bestimmten Eigenschaft. Die Vereinigung der Listen muss eine bestimmte Anzahl Elemente enthalten. Bei Paaren sind das zwei, bei Tripeln drei und bei Quadrupeln vier Einträge in den Kandidatenlisten.
Findet man zum Beispiel ein Paar, das nur noch die selben beiden Zahlen enthalten kann dann ist klar, dass keine der Zahlen anderswo in der Figur stehen kann, da sonst für eine der Zellen keine Zahl mehr übrig bleibt. Daher können die beiden Zahlen dann aus den Kandidatenlisten aller anderen Zahlen aus der Figur entfernt werden. Die Begründung für Tripel und Quadrupel ist analog.
Es ist nicht nötig nach mehr als Quadrupeln zu suchen, da für jedes Naked Quintupel ein Hidden Quadrupel existiert.

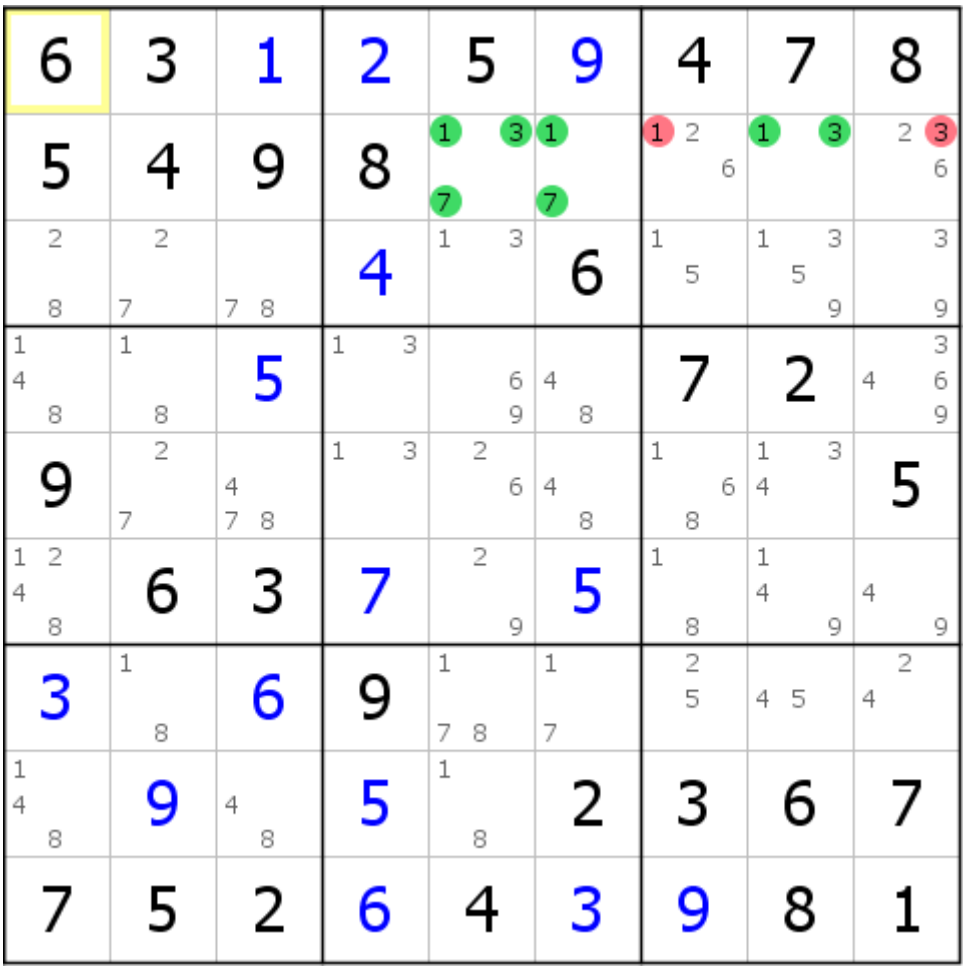

Abbildung 2.8: Naked Subset - Naked Triple

In **Abbildung 2.8** findet man das *Naked Subset* in Zeile 2. Genauer gesagt handelt es sich um ein *Naked Triple*. Hier hat die Vereinigung der Kandidatenlisten der Zellen z2s5, z2s6 und z2s8 genau drei Einträge: 1, 3 und 7. Es gibt offensichtlich keine andere Möglichkeit, als die drei Ziffer auf diese zellen zu verteilen. Demnach können sie in der Zeile sonst nicht vorkommen und können aus den Kandidatenlisten der anderen Zellen entfernt werden.

2.4.8 Hidden Subset

Analog zu den *Naked Subset*-Techniken ist auch *Hidden Subset* ein Sammelbegriff. Er beinhaltet die Techniken *Hidden Pair*, *Hidden Triple* und *Hidden Quadruple*. Auch hier ändert sich nur die Anzahl der betrachteten Kandidatenlisten. Hier soll exemplarisch die Technik *Hidden Tuple* erklärt werden, im folgenden Beispiel wird dann die Technik *Hidden Quadruple* angewendet.
Wenn man in einer Figur zwei Zahlen findet, die ausschließlich in den zwei gleichen Zellen vorkommen können, dann müssen diese beiden Zahlen in die beiden Zellen gesetzt werden. Daher kann man alle anderen Kandidaten in den Zellen von der Kandidatenliste streichen.

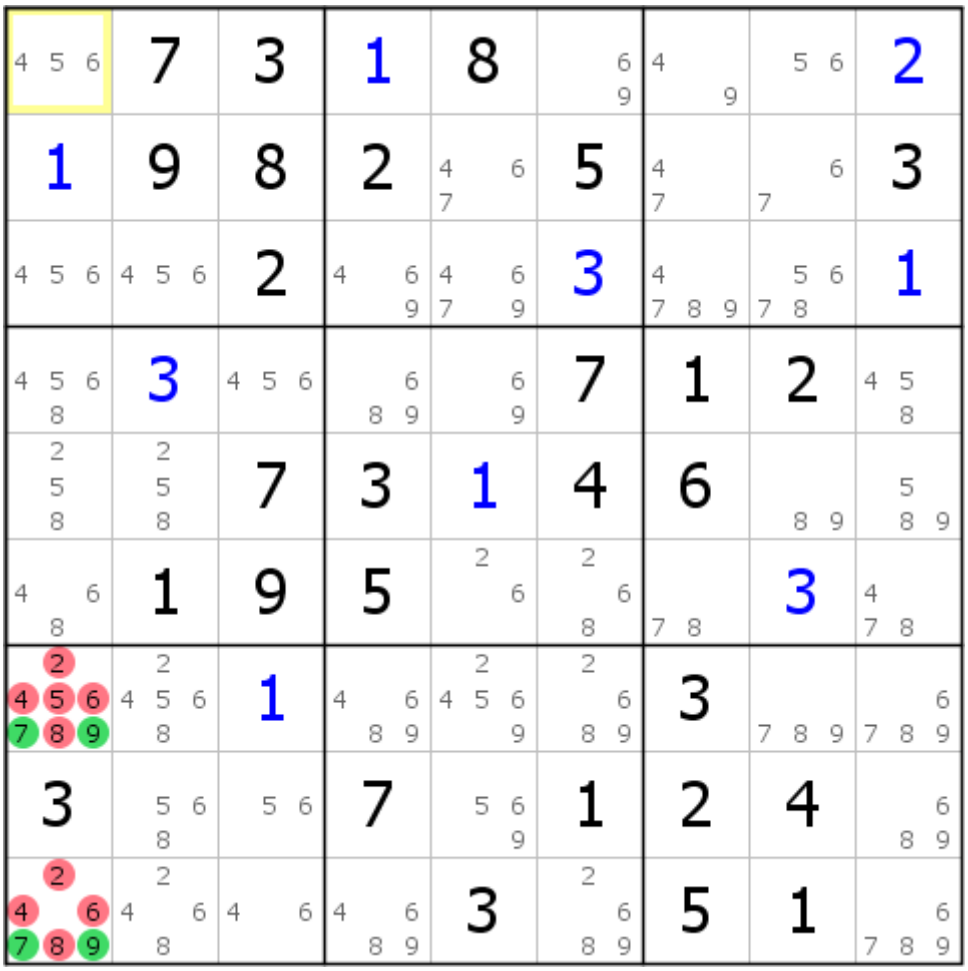

Abbildung 2.9: Hidden Subset - Hidden Quadruple

In **Abbildung 2.9** betrachten wir den Block 7 und in diesem Block die Zellen z7s1 und z9s1. Nur in diesen Zellen können die Ziffern 7 und 9 in Block 7 vorkommen. Da wir diese zwei Ziffern nun auf die zwei Zellen verteilen müssen gibt, es dort keinen Platz mehr für andere Zahlen. Diese können also aus den Kandidatenlisten entfernt werden.

2.4.9 Fish

Die *Fish* Methoden sind ein Sammelbegriff für eine ganze Grupppe von Methoden, die alle nach dem gleichen Prinzip arbeiten. Wie echte Fische hat dieses Prinzip eine sehr große Anzahl Unterarten hervorgebracht. Kleine Fische wie zum Beispiel X-Wing sind von geübten Sudoku Spielern noch zu finden, wenn die Fische allerdings größer werden, dann sind sie nur noch mit sehr hohem Aufwand manuell zu finden und daher eher zur Verarbeitung mit dem Computer gedacht.
Auf einer Internetseite, die sich unter anderem mit den Lösungsmethoden für Sudokus befasst, findet sich die folgende Erklärung zur Funktionsweise von Fischen.

> *[...] Man suche eine bestimmte Anzahl von Häusern, die sich nicht überschneiden. Diese Häuser werden als Base-Sets (Basismengen) bezeichnet (Set wird hier synonym für Haus verwendet), die in diesen Häusern enthaltenen Kandidaten sind die Basiskandidaten. Nicht überschneiden bedeutet hier, dass kein Basiskandidat in mehr als einem Haus enthalten sein darf, die Häuser selbst dürfen sich schon überlappen. Nun suche man eine gleiche Anzahl an sich nicht überschneidenden Häusern, die alle Basiskandidaten abdecken (engl.: cover). Diese neuen Häuser sind die Cover-Sets, sie enthalten die Coverkandidaten. Wenn eine solche Kombination existiert, können alle Coverkandidaten gelöscht werden, die nicht gleichzeitig Basiskandidaten sind.* [1] [2]

[1] Quelle: `http://hodoku.sourceforge.net/de/tech_fishg.php`
[2] Häuser stehen hier für Figuren

Basic Fish

Die Techniken *X-Wing*, *Swordfish* und *Jellyfish* sind die einfachsten Unterarten der Fische. Sie funktionieren nach dem selben Prinzip, nur dass eine unterschiedliche Anzahl an Figuren betrachtet wird, ähnlich zu *Naked Subset* und *Hidden Subset*. Hier wird stellvertretend die Technik *X-Wing* erklärt und am Beispiel gezeigt.
Dazu sucht man zwei Spalten oder Zeilen, die ausschließlich in den selben zwei Zellen einen bestimmten Kandidaten, die Fischziffer, beinhalten. Nun kann man aus dem jeweils anderen Paar von Figuren (Spalte oder Zeile), deren Position durch die zwei gefundenen Zellen festgelegt wird, alle Fischziffern löschen, die nicht gleichzeitig in einer der zuerst ausgesuchten Figuren liegen.
Da die Fischziffer in den beiden zuert ausgesuchten Figuren nur an jeweils zwei Stellen liegen kann und diese sich paarweise gegeinseitig ausschließen ist klar, dass jedes Vorkommen der Fischziffer in den zuletzt ausgesuchten Figuren in der Überschneidung mit den ersten Figuren liegen muss.

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

Abbildung 2.10: Basic Fish - X-Wing

Ein Beispiel für die Technik *X-Wing* findet sich in **Abbildung 2.10**. Hier wählen wir zuerst die Figuren Zeile 3 und 8. Diese enthalten die Fischziffer 5 nur an den Stellen 1 und 4. Wichtig ist, dass es in den beiden Zeilen die gleichen Positionen sind. Wir betrachten nun zwei Fälle. Steht in z3s4 die Ziffer 5, dann kann man die rot markierte Ziffer sofort löschen. In Fall zwei steht die Ziffer 5 in z3s1. Daher kann sie nicht in z8s1 stehen und müsste daher in z8s4 eingesetzt werden. Das würde ebenfalls die rot markierte Ziffer ausschließen. Da sie in beiden möglichen Fällen nicht in z1s4 stehen kann, kann sie gelöscht werden.

2.4.10 Single Digit Patterns

Single Digit Patterns ist ein Oberbegriff für Techniken, denen allen gemeinsam ist, dass sie nur eine Ziffer betrachten. Die Techniken *Skyscarper* und *2-String-Kite* sind Unterarten der Technik *Turbot Fish*, die aber für den Menschen einfacher zu finden sind. Da der Schwierigkeitsgrad des Sudokus für Menschen bewertet werden soll, werden sie hier mit aufgelistet und auch im Programm verwendet.

Skyscarper

Die Technik *Skyscarper* bedeutet übersetzt Wolkenkratzer und leitet sich von der Anordnung der betrachteten Ziffern ab. Gesucht werden zwei Zeilen oder Spalten, in deren Kandidatenlisten die Ziffer jeweils noch genau zwei mal auftaucht. Wenn nun zwei der Kandidaten in der selben anderen Figur (Spalte oder Zeile) sind, dann hat man einen Wolkenkratzer gefunden. Die beiden Zahlen, die in der selben anderen Figur sind, bilden das Fundament des Woleknkratzers, sie schließen sich gegenseitig aus. Das bedeutet wiederum, dass eine der beiden anderen gefundenen Ziffern dort stehen muss. Daher können alle Kandidaten, die von beiden Ziffern ausgeschlossen werden, aus den Kandidatenlisten gelöscht werden.

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

Abbildung 2.11: Skyscarper

In **Abbildung 2.11** betrachten wir die Spalten 5 und 8. Hier sind die Bedingungen für den Wolkenkratzer erfüllt, da in jeder Spalte die Ziffer 1 jeweils genau zwei mal vorkommt und sie in beiden Spalten an Position 5 auftaucht. Für das Feld z8s8 gibt es nun zwei Möglichkeiten. Entweder die Ziffer 1 steht in diesem Feld oder nicht. Diese beiden Möglichkeiten werden nun separat betrachtet. Wenn die Ziffer 1 in Feld z8s8 steht, dann schließt das bereits alle rot markierten Zahlen aus. Für den Fall, dass die Ziffer 1 nicht in z8s8 steht, muss sie in z5s8 stehen, das geht aus der Bedingung des Wolkenkratzers hervor. Da sich nun z5s8 und z5s5 in der selben Zeile befinden, kann die Ziffer 1 nicht in z5s5 vorkommen. Deshalb muss sie in z8s5 stehen, wo sie alle rot markierten Felder ausschließt. In jeder Möglichkeit werden die roten Felder ausgeschlossen, daher kann die Ziffer 1 daraus gelöscht werden.

2-String Kite

Die Technik *2-String-Kite* wird auch Paierdrache genannt, was sich, wie beim Wolkenkratzer, aus der Anordnung der Zahlen ableitet. Auch hier wird nur eine einzige Ziffer betrachtet. Gesucht werden eine Zeile und eine Spalte, die nur noch zwei Kandidaten der betrachteten Ziffer enthalten, so dass ein Kandidat der Spalte und ein Kandidat der Zeile im selben Block liegen. Die Zeile und die Spalte nennt man die *Schnüre* des Drachens. Die Enden der Schnüre liegen im gleichen Block, die Position des Anfangs ist relevant für das Löschen des Kandidaten. Gelöscht werden kann nämlich der Kandidat in der Zelle, die von beiden Anfängen der Schnüre ausgeschlossen wird. Das kommt zustande, da die betrachtete Ziffer in jedem Fall am Anfang einer der beiden Schnüre stehen muss.

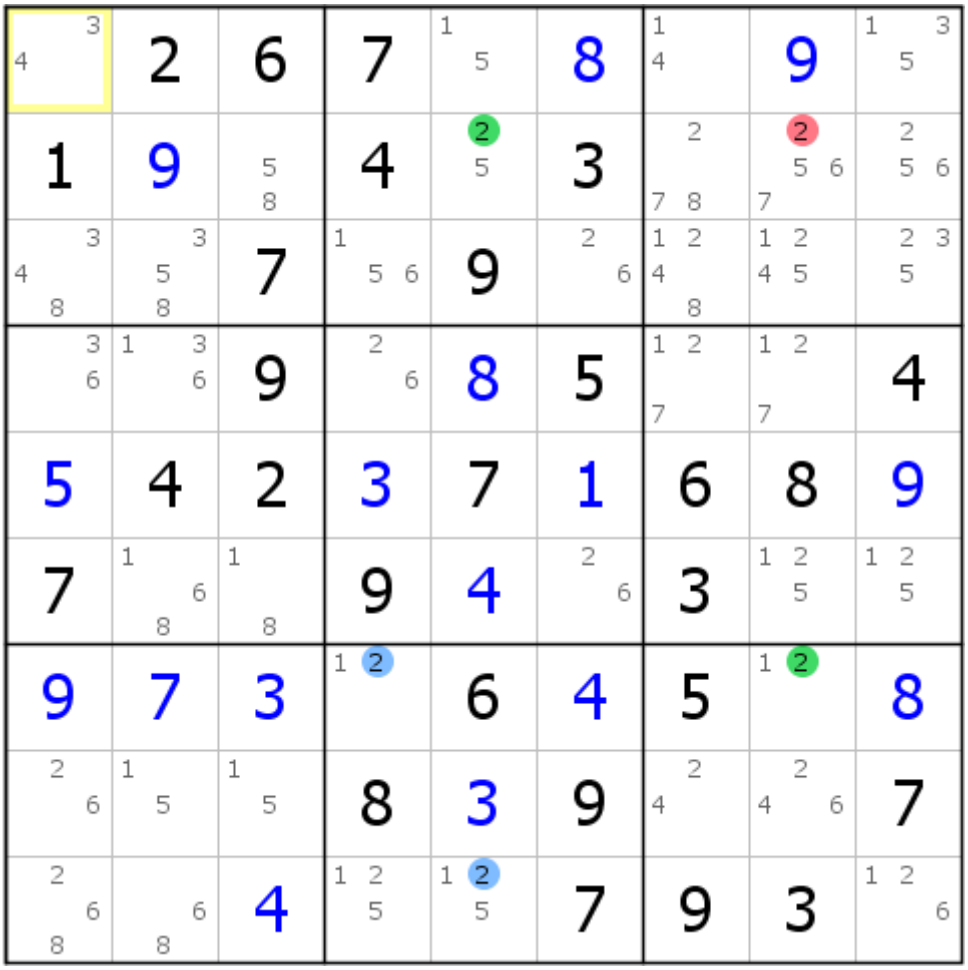

Abbildung 2.12: 2-String-Kite

Abbildung 2.12 zeigt einen *2-String-Kite*. Betrachtet wird die Ziffer 2, Zeile 7 und Spalte 5 fungieren als Schnüre des Drachen. In z2s5 betrachten wir zwei Fälle: Entweder die Ziffer 2 steht dort oder sie steht dort nicht. Für den ersten Fall gilt, dass dann die rot markierte Ziffer in z2s8 ausgeschlossen ist. Der zweite Fall ist etwas komplizierter. Wenn die Ziffer 2 nicht in z2s5 steht, dann muss sie an der einzig anderen möglichen Position der Spalte stehen, nämlich in z9s5. Daher kann sie nicht in z7s4 stehen, da diese Felder im selben Block liegen. Da in Zeile 7 auch nur noch zwei Kandidaten für die Ziffer 2 übrig waren, muss sie in z7s8 stehen, wo sie z2s8 ausschließt. In jedem der Fälle kann also die Ziffer 2 nicht in z2s8 stehen und daher kann sie dort gelöscht werden.

Turbot Fish

Wie auch bei den vorherigen *Single Digit Patterns* wird auch beim *Turbot Fish* nur eine Ziffer betrachtet. Gesucht wird eine Kette, die vier Ziffern lang ist, so dass Anfang und Ende eines Kettenglieds in einer gemeinsamen Figur liegen. Wichtig ist dabei, dass im ersten und dritten gemeinsamen Figur die beiden betrachteten Kandidaten die einzigen verbliebenen sind. Da die Kette vier Glieder lang ist, muss entweder der Anfang oder das Ende der Kette wahr sein, daher können die Kandidaten gelöscht werden, die von beiden Feldern ausgeschlossen werden.

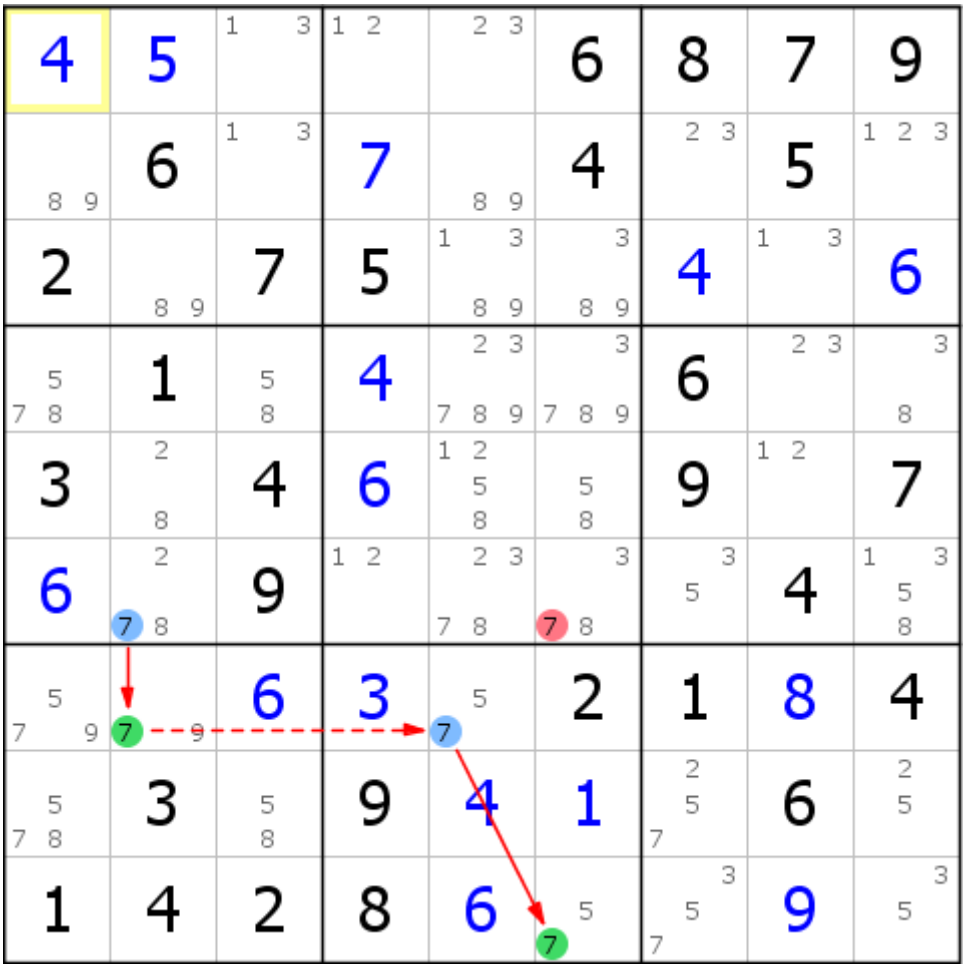

Abbildung 2.13: Turbot Fish

In der obigen **Abbildung 2.13** beginnt die Kette der Ziffer 7 im Feld z6s2. In der selben Spalte befindet sich die zweite Ziffer in z7s2, sie ist dort der einzige weitere Kandidat der Ziffer 7, was Voraussetzung für den Turbot Fish ist, da es sich hier um das erste Glied handelt. Die nächste Ziffer liegt in der gleichen Zeile, z7s5. Die letzte Ziffer befindet sich im selben Block wie auch ihr Vorgänger, in z9s6, und auch sie ist hier der einzige weitere Kandidat für diese Ziffer. Wir betrachten zwei Fälle: die Ziffer 7 steht in z6s2 oder sie steht dort nicht. Im ersten Fall kann der rot markierte Kandidat in z6s6 gelöscht werden, da er direkt ausgeschlossen wird. Wenn die 7 dort nicht steht, dann muss sie in z7s2 stehen, da das der einzige andere Kandidat in der Zeile ist. Darum kann die 7 dann nicht in z7s5 stehen. Daraus folgt, dass sie in z9s6 stehen muss, was dann im zweiten Fall die Ziffer 7 in z6s6 ausschließt, womit diese dort in jedem Fall nicht stehen kann.

Empty Rectangle

Für die Technik *Empty Rectangle* gilt das selbe wie für alle *Single Digit Patterns*, es wird nur eine Ziffer betrachtet und der Name leitet sich aus der Form der Anordnung der Ziffern ab. Um ein *Empty Rectangle* zu finden, wird ein Block gesucht, in dem ein Kandidat ausschließlich noch in einer Zeile und in einer Spalte vorkommen kann. Dann bilden die verbliebenen Kandidaten eine Ecke eines *Empty Rectangles*.

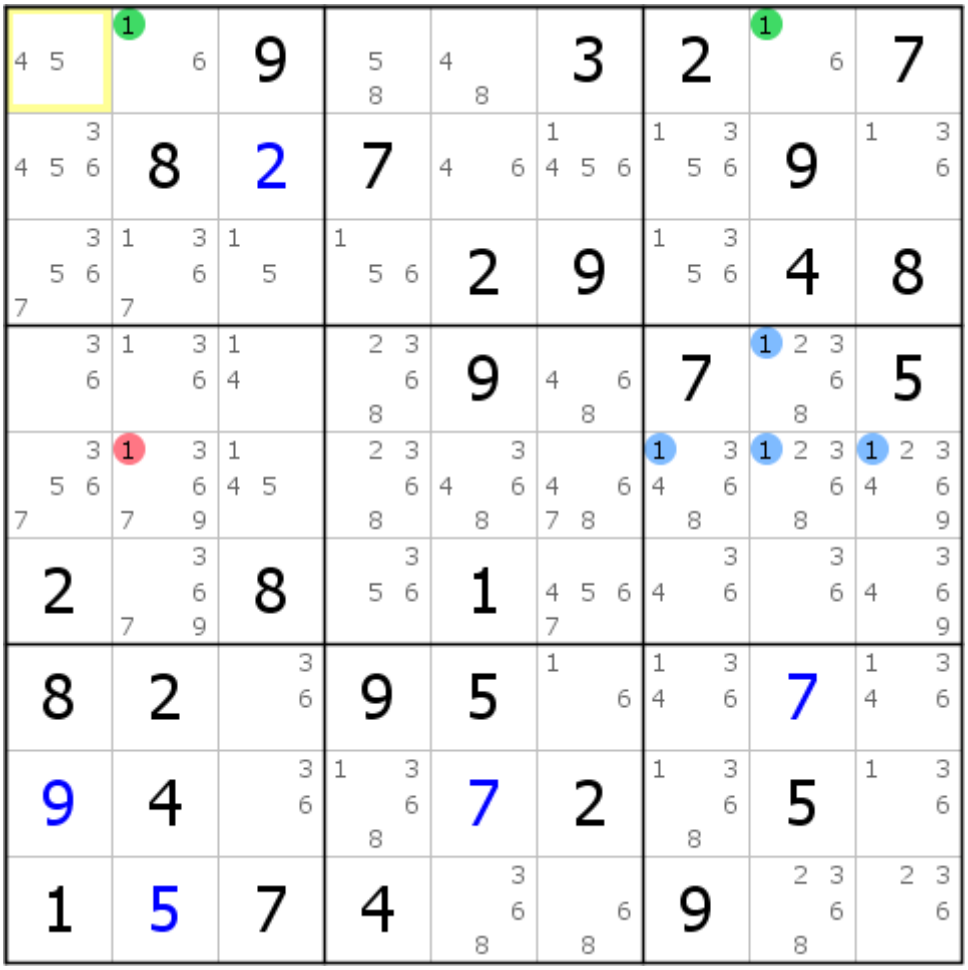

Abbildung 2.14: Empty Rectangle

In **Abbildung 2.14** sehen wir ein *Empty Rectangle*, das von Block 6 ausgeht. Die verbleibenden Kandidaten der Ziffer 1 können hier nur noch in Zeile 5 und Spalte 8 stehen. Wenn wir Zeile 1 betrachten, dann sehen wir, dass die Ziffer 1 hier nur noch an zwei Positionen stehen kann, nämlich an der zweiten und an der achten. Wir betrachten nun diese beiden Fälle getrennt. Wenn die Ziffer 1 in z1s2 steht, dann ist die Ziffer 1 in z5s2 direkt ausgeschlossen. Wenn die Ziffer 1 in z1s8 steht, dann kann sie in Block 6 nur noch in einer Zeile stehen, nämlich Zeile 5. Auch in diesem Fall ist die Ziffer 1 dann im Feld z5s2 ausgeschlossen. Daher kann sie dort als Kandidat gelöscht werden.

2.4.11 Wings

XY-Wing

Die Technik *XY-Wing* wird manchmal auch nur *Y-Wing* genannt, da sie aussieht wie ein *X-Wing* (siehe Kapitel 2.4.9) nur mit drei Ecken. Zuerst sucht man eine Zelle, in der nur noch 2 Kandidaten verblieben sind. Diese Kandidaten nennt man dann x und y. Daher kommt der im Allgemeinen bekanntere Name der Strategie. Im nächsten Schritt sucht man nun eine Zelle, die in einer gemeinsamen Figur mit der ersten Zelle liegt und auch nur noch 2 Kanididaten der Form hat, dass einer der Kandidaten dem x aus der ersten Zelle entspricht und der zweite Kandidat ungleich dem y ist. dieser wird nun z genannt. Anschließend sucht man eine dritte Zelle mit nur noch zwei verbliebenen Kandidaten, die ebenfalls in einer gemeinsamen Figur mit der ersten Zelle liegt, aber nicht in der selben wie die zweite gefundene Zelle. Wenn diese Zelle nun die Kandidaten y und z enthält, dann hat man einen *XY-Wing* gefunden. Gelöscht werden kann nun der Kandidat z aus der Zelle, die von der zweiten und dritten Zelle ausgeschlossen wird. Das funktioniert, da in der ersten Zelle entweder x oder y steht. Wenn in der ersten Zelle x steht, dann steht in der zweiten Zelle z, da dort nur x und z stehen kann, x aber durch die erste Zelle ausgeschlossen wird. Wenn in der ersten Zelle aber y steht, dann muss in der dritten Zelle z stehen, da dort nur y und z stehen können und y ausgeschlossen wird. Daher steht in einer der beiden Zellen z und alle Kandidaten von z, die durch beide Felder ausgeschlossen werden, können gelöscht werden.

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

Abbildung 2.15: XY-Wing

In **Abbildung 2.15** stehen in z8s9 die Kandidaten 6 für x und 9 für y. In Zelle z5s9 stehen 6 für x und 7 für z. in z8s6 stehen 9 für y und 7 für z. Wieder werden zwei Fälle getrennt betrachtet. Im ersten Fall nehmen wir an, die Ziffer 7 steht in z5s9. Dann wäre die rot markierte 7 in z5s6 direkt ausgeschlossen, da die beiden Zellen in der selben Zeile liegen. Im zweiten Fall nehmen wir an, die Ziffer 7 steht nicht in z5s9. Dann muss dort die Ziffer 6 stehen. Dadurch muss in z8s9 die Ziffer 9 eingetragen werden, was dazu führt, dass in z8s6 die Ziffer 7 steht und ebenfalls die rot markierte 7 in z5s6 ausschließt. In beiden möglichen Fällen kann die Ziffer 7 also nicht in z5s6 stehen und kann damit gelöscht werden.

XYZ-Wing

Bei der Technik *XYZ-Wing* handelt ies sich um eine erweiterte Version der Technik *XY-Wing*. Zusätzlich zu den Kandidaten x und y steht hier in der ersten Zelle noch der Kandidat z, der Rest ändert sich nicht. Statt zwei Fällen werden hier drei Fälle betrachtet. Wenn der Kandidat x in der ersten Zelle steht, dannn bleibt die Argumentation wie beim *XY-Wing*. Das selbe gilt für den Fall, dass der Kandidat y in der ersten Zelle steht. Im dritten Fall steht der Kandidat z in der ersten Zelle. In diesem Fall würde er alle Kandidaten ausschließen, die von dort aus direkt auschgeschlossen werden. Daher können alle Kandidaten der Ziffer z gelöscht werden, die von allen drei Zellen ausgeschlossen werden.

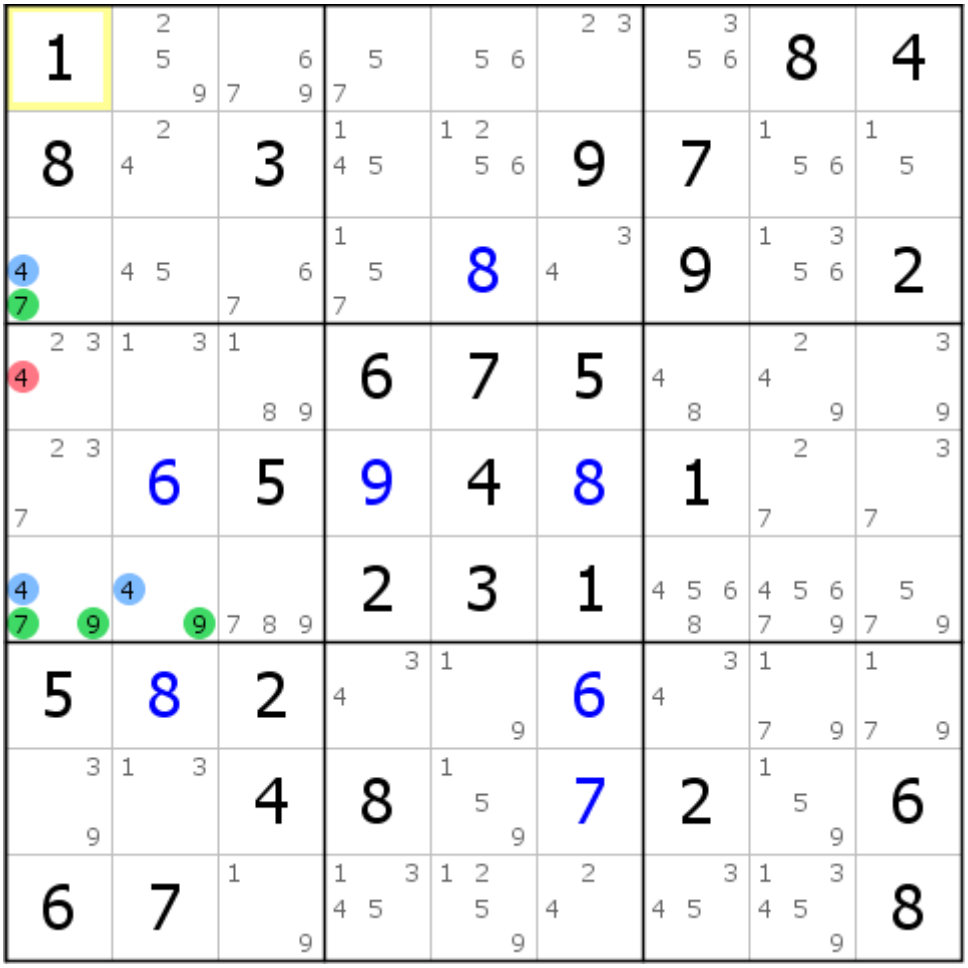

Abbildung 2.16: XYZ-Wing

In **Abbildung 2.16** sieht man einen *XYZ-Wing*, dessen erste Zelle z6s1 ist. Hier stehen die Kandidaten 7 für x, 9 für y und 4 für z. Bei z3s1 findet man die zweite Zelle mit den Kandidaten 7 und 4, also x und z. Die dritte Zelle ist z6s2, sie enthält die Kandidaten 9 und 4 und damit y und z. Wir betrachten nun die drei möglichen Fälle für z6s1. Wenn dort die Ziffer 4 steht, dann ist die rot markierte Ziffer 4 in z4s1 direkt ausgeschlossen. Wenn in z6s1 die Ziffer 7 steht, dann kann diese nicht in z3s1 vorkommen, daher muss dort die Ziffer 4 stehen, was diese wiederum in z4s1 ausschließt. Im dritten Fall steht in z6s1 die Ziffer 9, was dazu führt, dass sie nicht in z6s2 stehen kann und daher dort die Ziffer 4 stehen muss. Das schließt auch im letzten möglichen Fall die Ziffer 4 in z4s1 aus, damit kann sie gelöscht werden.

W-Wing

Die Technik *W-Wing* ist die letzte und schwierigste Technik der Wings. Hierbei werden immer zwei Ziffern betrachtet. Zuerst sucht man eine Zelle, in der nur noch zwei Kandidaten x und y möglich sind. Nun wird eine Figur gesucht, in der der Kandidat x nur noch zwei mal vorkommen kann und eines der möglichen Vorkommen von der ersten Zelle ausgeschlossen würde. Im letzten Schritt sucht man eine Zelle, die wieder ausschließlich die Kandidaten x und y enthält und die das andere mögliche Vorkommen der Ziffer x in der zuvor gefundenen Figur ausschließen würde. Findet man eine solche Konstelation, dann handelt es sich um einen *W-Wing*. Gelöscht werden können die Kandidaten y, die von beiden Zellen gleichzeitig ausgeschlossen werden, da der Kandidat y entweder in der ersten oder in der zweiten gefundenen Zelle stehen muss.

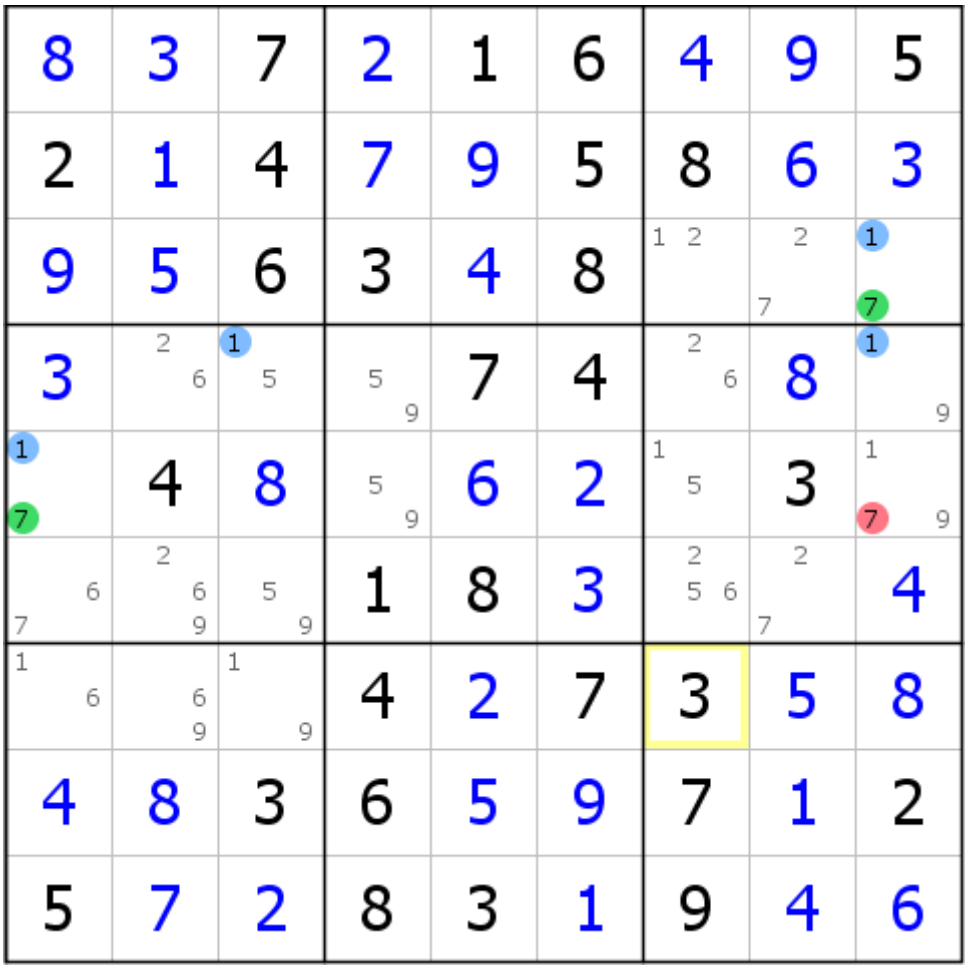

Abbildung 2.17: W-Wing

In **Abbildung 2.17** betrachten wir zuerst z3s9. Hier sind die einzigen beiden Möglichkeiten 1 für x und 9 für y. Betrachten wir also zwei Fälle. Im Fall 1 nehmen wir an, dass die Ziffer 7 in z3s9 steht. Dadruch ist die rot markierte Ziffer 7 in z5s9 ausgeschlossen. Wenn die Ziffer 7 nicht in z3s9 stht, dann muss dort die Ziffer 1 stehen, das ist Fall 2. Dadurch wird die blau markierte 1 in z4s9 auschgeschlossen, was dazu führt, dass die Ziffer 1 in z4s3 stehen muss. Da dann die einzige verbleibende Möglichkeit für z4s1 die Ziffer 7 ist und diese die rot markierte Ziffer 7 in z5s9 ausschließt, kann die Ziffer 7 in z5s9 in keinem Fall dirt stehen und wird gelöscht.

2.4.12 Sue de Coq

Die Technik *Sue de Coq* wurde ursprünglich unter dem Namen *Two-Sector Disjoint Subsets* in einem Forum[3] vorgestellt. Anderen Benutzern des Forums war dieser Name zu umständlich und sie begannen die Strategie als *Sue de Coq* zu bezeichnen, dem Benutzernamen des Vorstellers.
Um die Technik anzuwenden sucht man zuerst zwei oder drei Zellen in einem Block, die in der selben Reihe oder Spalte liegen. Die Vereinigung der Kandidatenlisten muss dabei bei zwei Zellen vier Kandidaten enthalten und bei drei Zellen müssen fünf Kandidaten vorhanden sein. Nun sucht man zusätzlich eine Zelle im selben Block, die ausschließlich zwei Kandidaten der Vereinigung enthält. Eine solche Zelle sucht man auch in der Reihe oder Spalte, in der die ersten Zellen liegen. Die Schnittmenge der Kandidatenlisten der beiden zusätzlichen Zellen muss leer sein.
Die Kandidaten aus der zusätzlichen Zelle in der Reihe können nun aus allen unbeteiligten Zellen der Reihe gelöscht werden. Das gilt ebenfalls für die Kandidaten der zusätzlichen Zelle des Blocks, diese können aus allen unbeteiligten Zellen des Blocks entfernt werden.

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

Abbildung 2.18: Sue de Coq

In Abbildung **3.16**[4] sieht man einen *Sue de Coq* mit den zwei Basiszellen z7s1 und z7s3. Diese liegen im Block 7 in der selben Zeile. Die Vereinigung der Kandidaten enthält die Ziffern 3, 4, 5 und 9. Die zusätzliche Zelle in der Zeile ist z7s7. Sie enthält die Kandidaten 4 und 5 die Teilmenge der vorherigen

[3] `http://forum.enjoysudoku.com/two-sector-disjoint-subsets-t2033.html`
[4] Beispiel aus [3]

Vereinigung ist. Die zusätzliche Zelle im Block 7 ist z8s3. Diese enthält die Kandidaten 3 und 9, die ebenfalls Teilmenge der Vereinigung ist. der Schnitt der Kandidatenlisten der zusätzlichen Zellen ist leer. Damit sind alle Bedingungen für den *Sue de Coq* erfüllt und die rot markierten Kandidaten können gelöscht werden.

2.4.13 Simple Coloring

Beim *Simple Coloring* wird für die betrachtete Ziffer nach Figuren gesucht, in denen die Ziffer nur noch in zwei Kandidatenlisten vorkommt. Den Zellen dieser Kandidatenlisten werden dann zwei unterschiedliche Farben zugeordnet. Dadurch, dass sich die Kandidaten in den Figuren gegenseitig ausschließen, steht die betrachtete Ziffer immer in jedem Feld der entweder einen oder anderen Farbe.
Erster Widerspruch (Color Trap): In Zellen, die nicht gefärbt sind, die von beiden Farben gesehen werden und die die betrachtete Ziffer enthalten, kann diese gelöscht werden.
Zweiter Widerspruch (Color Wrap): Wenn eine Farbe in der gleichen Figur zwei mal vorkommt, dann können alle Vorkommen der Ziffer in Zellen mit dieser Farbe gelöscht werden, da die Ziffer entweder in allen Zellen steht oder in keiner. In allen Zellen kann die Ziffer nicht stehen, da eine Ziffer nicht zwei mal in einer Figur vorkommen kann.

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

Abbildung 2.19: Simple Coloring - Color Trap

In **Abbildung 3.17**[5] wird die Ziffer 3 betrachtet. Sie kann entweder in allen dunkelgelb oder allen hellgelb gefärbten Zellen stehen. Wir betrachten nun zwei Fälle. Wenn die Ziffer 3 in z8s9 steht, dann kann sind die Kandidaten in z8s1, z1s9 und z3s9 ausgeschlossen. Wenn nicht, dann muss sie in allen dunkelgelb gefärbten Zellen stehen, insbesondere in z1s4 und z3s3. Diese schließen ebenfalls die rot markierten Kandidaten aus. Damit können diese in keinem Fall in z1s9 und z3s9 stehen und können gelöscht werden.

[5] Beispiel aus [2]

2.4.14 Almost Locked Set

Ein *Locked Set* bezeichnet eine Gruppe von n ungelösten Zellen, deren Vereinigung der Kandidatenlisten nur noch n Kandidaten enthält. Das erlaubt das Entfernen aller Kandidaten aus den anderen Zellen der Figur, die in der Vereinugung der Kandidatenlisten vorkommen.
Ein *Almost Locked Set* hat $n + 1$ Kandidaten und ist daher einzeln nicht zu gebrauchen. Man kann allerdings Informationen erhalten, indem man zwei *Almost Locked Sets* kombiniert. Dazu müssen sie mindestens einen gemeinsamen Kandidaten haben für den zusätzlich gilt, dass alle Instanzen des Kandidaten in ALS 1 alle Instanzen des Kandidaten in ALS 2 sehen können. Dieser Kandidat kann dann nur einmal in beiden ALS gesetzt werden, da sich alle Instanzen gegenseitig ausschließen. Man nennt diese Ziffer *Resitricted Common Candidate* oder *RCC*.

ALS XZ

Die Technik *ALS XZ* ist die einfachste Unterart der *Almost Lockes Sets*. Man sucht zwei *ALS* mit einem *RCC*. Dieser wird X genannt. Wenn beide *ALS* noch einen gemeinsamen Kandidaten Z besitzen, der kein *RCC* ist, dann kann Z aus allen Zellen gelöscht werden, die nicht zum *ALS* gehören und die alle Instanzen von Z in beiden *ALS* sehen. Das funktioniert, da durch X ein *ALS* zum *Locked Set* wird. Da in beiden *ALS* die Ziffer Z vorkommt wird diese auf ein *ALS* beschränkt. Das bedeutet, dass Z in genau einem *ALS* vorkommt und daher können alle Kandidaten gelöscht werden, die von allen Instanzen gesehen werden.

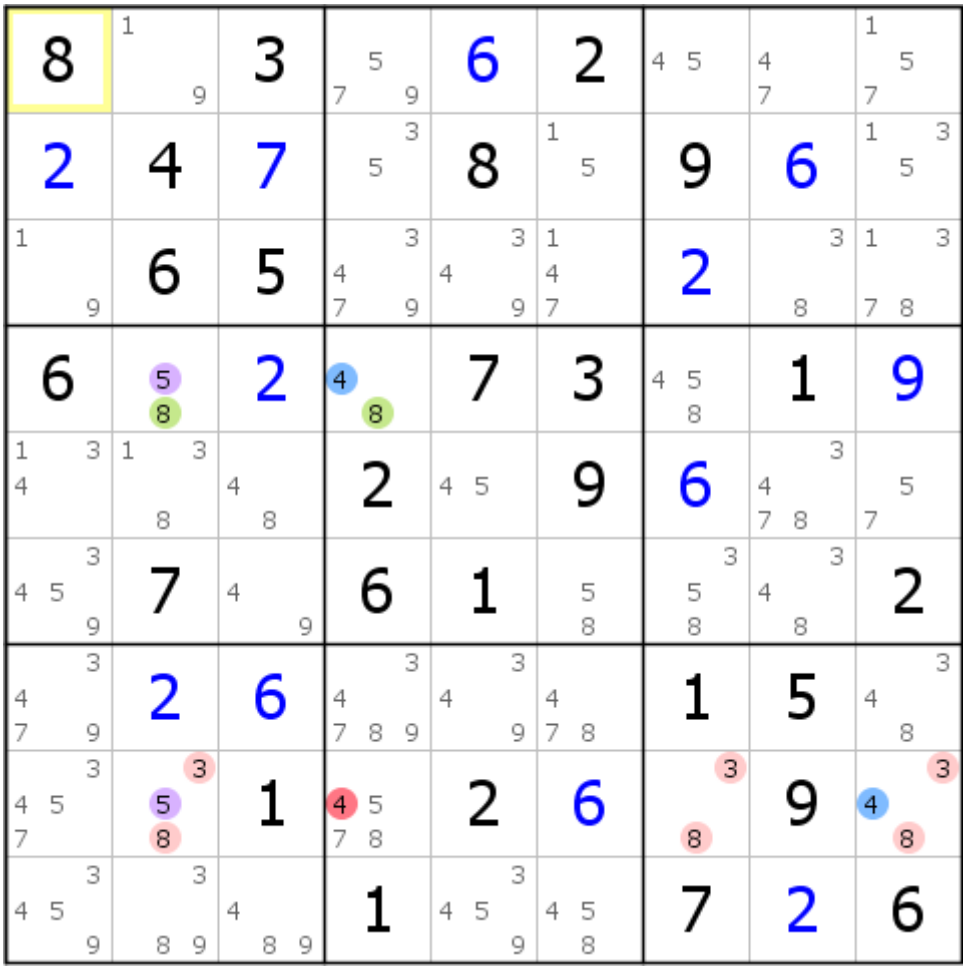

Abbildung 2.20: ALS XZ

In **Abbildung 2.20** sehen wir die beiden *ALS* einmal in Zeile 4 Spalte 2 und 4 mit den Kandidaten 4, 5 und 8 und das zweite in Zeile 6 Spalte 2, 7 und 9 mit den Kandidaten 3, 4, 5 und 6. Der *RCC* ist hier die Ziffer 5, da alle Instanzen in beiden *ALS* in Spalte 2 liegen. Ausserdem kommt in beiden *ALS* die Ziffer 4 vor, die aber kein *RCC* ist, da sich zum Beispiel z4s4 und z8s9 nicht sehen können. Nun können alle Zellen die Ziffer 7 von ihren Kandidatenlisten, die alle Instanzen der Ziffer 7 in beiden *ALS* sehen, was in z8s4 der Fall ist.

ALS XY Wing

Bei der Technik *ALS XY Wing* benötigt man 3 *ALS* A, B und C. A und C haben den gemeinsamen *RCC* X, B und C haben den gemeinsamen *RCC* Y. X und Y sind unterschiedlich. *ALS* A und B haben die gemeinsame Ziffer Z, die kein *RCC* ist.
Nun werden zwei Fälle betrachtet. Wenn Z nicht in A steht, dann muss X in A stehen, da bei einem *ALS* nur eine Ziffer fehlen darf. Das bedeutet, C muss Y enthalten, da Z nicht in C steht. Daraus kann man folgern, dass B den Kandidaten Z enthalten muss. Im Fall zwei nehmen wir an, dass Z nicht in B steht. Dann muss Y in B stehen, woraus folgt, dass X in B steht. Daher muss dann Z in A stehen. In jedem möglichen Fall steht Z also entweder in A oder in B. Daher können alle Kandidaten von Z ausgeschlossen werden, die von allen Instanzen in A und B gesehen werden.

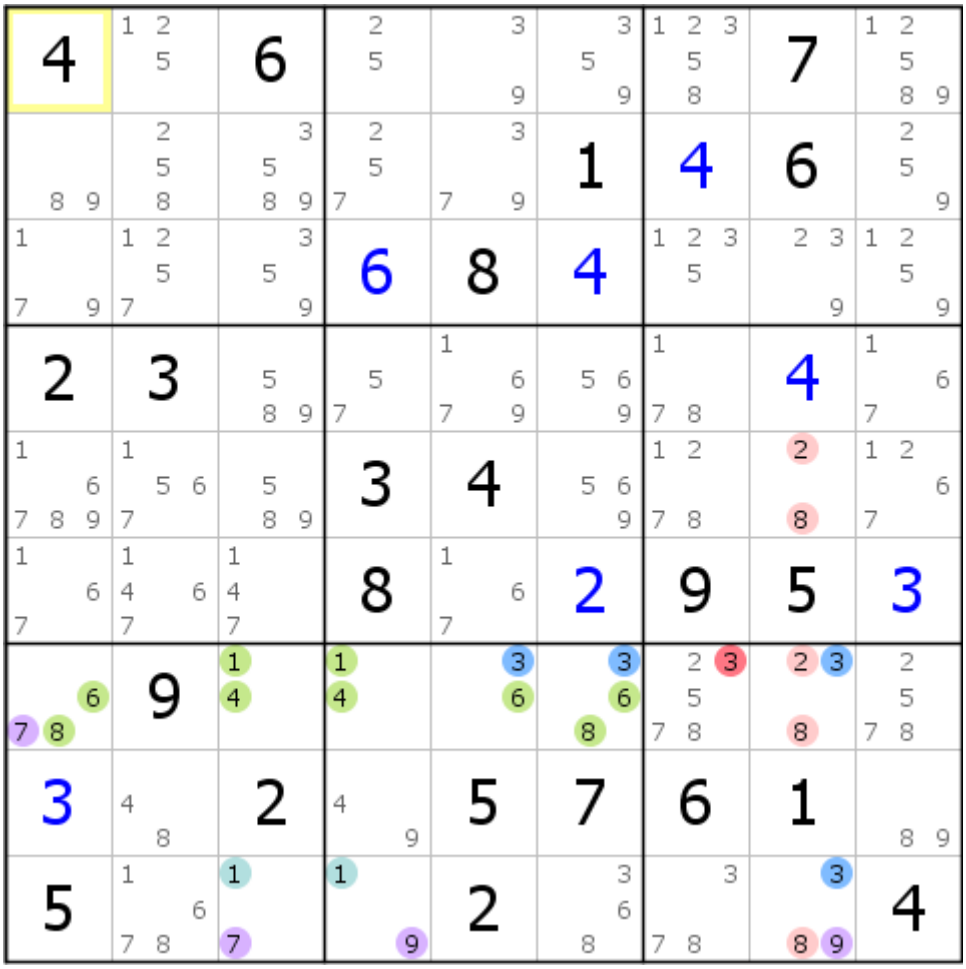

Abbildung 2.21: ALS XY Wing

Im oben stehenden Beispiel **Abbildung 4.19**[6] sieht man einen *ALS XY Wing*, *ALS* A ist hier Zeile 7 Spalten 1, 5 und 6 mit den Kandidaten 3, 6, 7 und 8, B ist Spalte 8 Zeilen 5, 7 und 9 mit den Kandidaten 2, 3, 8 und 9. *ALS* C befindet sich in Zeile 9 Spalten 3 und 4 mit den Kandidaten 1, 7 und 9. X ist also 7, Y ist 9 und Z ist 3. Da Z wegen der oben genannten Begründung nur in *ALS* A und B vorkommen kann, können jetzt alle Kandidaten von Z gelöscht werden, die von allen Instanzen von Z in A und B gesehen werden. Das ist die Ziffer 3 in z7s7.

[6] Beispiel aus [1]

ALS Chain

ALS Chains sind die Verallgemeinerung von *ALS XZ* und *ALS XY Wing*. Ein *ALS XZ* ist eine *ALS Chain* der Länge 2 und ein *ALS XY Wing* ist eine *ALS Chain* der Länge 3. Eine *ALS Chain* ist eine Kette von ALS verbunden durch *RCCs*, für die gilt, dass keine zwei aufeinanderfolgenden *RCCs* gleich sein dürfen. Die *ALS* am Anfang und Ende der Kette enthalten eine gemeinsame Ziffer Z. Diese Ziffer wird wie gewohnt dann aus allen Kandidatenlisten gelöscht, die von allen Instanzen der Ziffer Z im *ALS* am Anfang und am Ende der Kette gesehen werden.
Das geht, da durch die Verknüpfung durch *RCCs* die Ziffer Z entweder im *ALS* am Anfang der Kette stehen muss oder in dem am Ende.

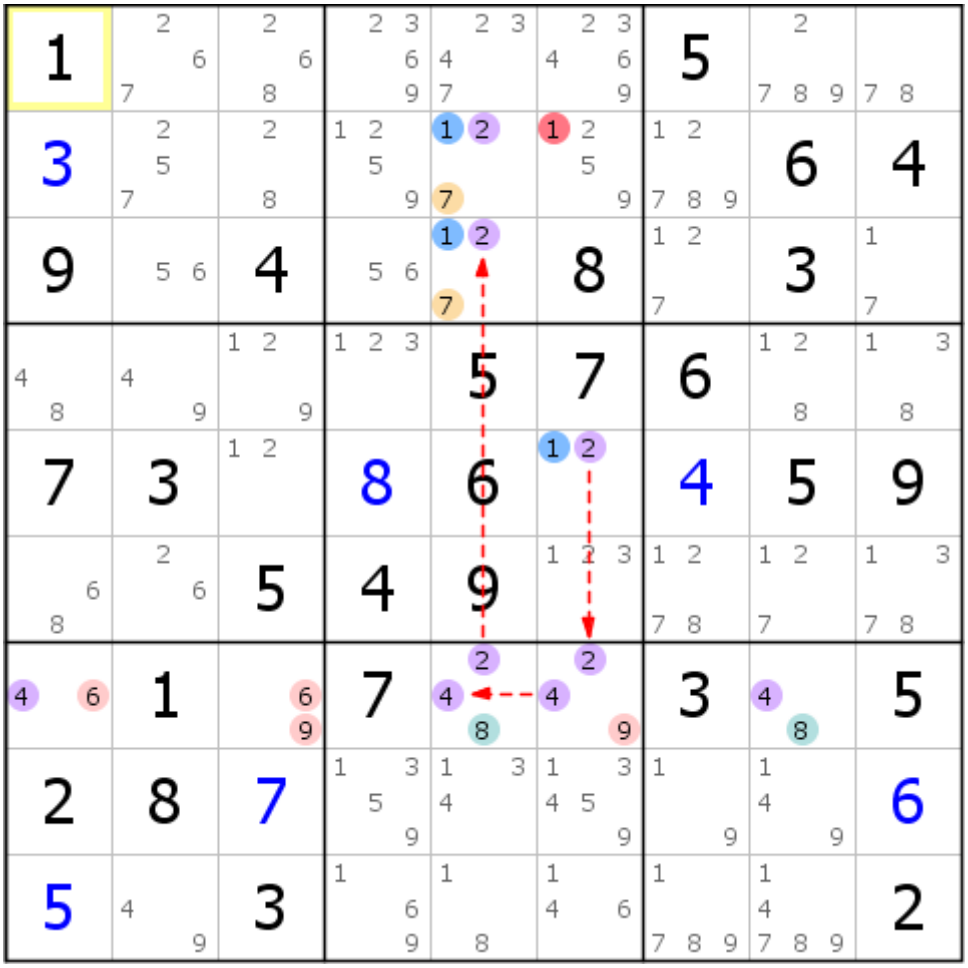

Abbildung 2.22: ALS Chain

In **Abbildung 2.22** sieht man eine *ALS Chain* der Länge 4. Das erste und einzellige *ALS* ist in z5s6, es enthält die Kandidaten 1 und 2. Durch den *RCC* 2 ist es verbunden mit dem*ALS* in Zeile 7 Spalten 1, 3 und 6, das die Kandidaten 2, 4, 6 und 9 enthält. Dieses *ALS* ist durch den *RCC* 4 verbunden mit dem *ALS* in Zeile 7, Spalten 5 und 8 mit den Kandidaten 2, 4 und 8. Die letzte Verbindung besteht dann durch den *RCC* 2 zum *ALS* in Spalte 5 Reihen 2 und 3, es enthält die Kandidaten 1, 2 und 7. Das erste und das letzte *ALS* enthalten den gemeinsamen Kandidaten 1. Dieser kann nun als Kandidat aus allen Zellen gelöscht werden, die alle Instanzen der Ziffer 1 in beiden *ALS* sehen. Das ist der Fall in z2s6.

2.4.15 Backtracking

Backtracking arbeitet nach dem *trial and error* Prinzip. Es wird eine zufällige Zahl aus der Kandidatenliste eines Feldes in das Feld eingesetzt. Danach werden die Kandidatenlisten wieder aktualisiert und Backtracking beginnt von vorne. Wenn bemerkt wird, dass durch das Einsetzen einer Zahl eine Situation entsteht, die die Sudoku Regel verletzt, dann wird der letzte Schritt zurück genommen. *Backtracking* ist also ein rekursiver Algorithmus und führt eine Tiefensuche über den Lösungsraum des Sudokus durch, der als Baum dargestellt werden kann. Sobald ein Ast des Baumes komplett durchsucht wurde, ohne eine Lösung zu finden, dann wird im Baum so lange wieder nach oben gegangen, bis eine andere Abzweigung verfügbar ist. Wenn eine Lösung gefunden wurde, dann terminiert der Algorithmus. Die Terminierung ist also immer gegeben, da jedes Sudoku eine eindeutige Lösung hat, wie in den *Regeln* 2.1 definiert.
Backtracking unterscheidet sich von den anderen Lösungstechniken dadurch, dass es das Sudoku vollständig löst, unabhängig davon, wie viele Zahlen dafür eingetragen werden müssen.

3 Maschinelles Lernen zur Charakterisierung des Schwierigkeitsgrades

3.1 Klassifikation

Unter einer Klassifikation versteht man in der Informatik das Einteilen von Objekten in vorher festgelegte Klassen. Diese Einteilung wird von einem Algorithmus durchgeführt, der anhand von festgelegten Merkmalen jedem Objekt eine Klasse zuordnet. Einen solchen Algorithmus nennt man Klassifikator. Um die Qualität eines Klassifikators zu analysieren gibt es verschiedene Metriken.

- Accuracy - Die Anzahl der richtig zugeordneten Klassen
- Recall - Der Anteil der positiven Beispiele, die auch positiv klassifiziert wurden
- Precision - Der Anteil der positiv klassifizierten Beispiele, die auch positiv sind

Ein Klassifikator benötigt vor der Phase der Klassifikation zunächst einmal eine Trainigsphase, in der er anhand von Beispielen lernt. Mit dem erlenten Wissen wird anschließend die Einteilung in die Klassen vorgenommen.
Es gibt viele verschiedene Ansätze für Klassifikatoren, von denen die wichtigsten in einem open source Framework implementiert sind. Dieses Framework heisst Weka[1] und wurde im praktischen Teil dieser Bachelorthesis verwendet.
Weka arbeitet unter anderem mit dem .arff[2] Format. In einer .arff Datei befindet sich neben den Metadaten hauptsächlich eine Sammlung von Featurevektoren. Jeder Featurevektor beschreibt ein zu klassifizierendes Objekt. Ein Eintrag in einem Featurevektor beschreibt eine Eigenschaft dieses Objekts. Das könnte bei Fahrzeugen zum Beispiel die Anzahl der Reifen sein. Bezogen auf Sudokus bedeutet das, dass jedes Sudoku durch einen Featurevektor beschrieben wird. Jede zur Klassifikation verwendete Eigenschaft eines Sudokus ist dann ein Wert im entsprechenden Featurevektor. Die Schwierigkeitsgrade der Sudokus sind die vorgegebenen Klassen.
Jeder Klassifikator in Weka hat als Eingabe eine Liste von Featurevektoren. Möchte man also das Zuordnen von Sudokus zu Schwierigkeitsgraden mit Weka realisieren, dann muss eine Methode entwickelt werden, die aus einem gegebenen Sudoku einen Featurevektor extrahiert.
Bei der Klassifikation wird ein Verfahren angewendet, das als *cross validation* bekannt ist und auch von Weka zur Verfügung gestellt wird. Dabei werden die Daten in eine vorgegebene Anzahl gleich großer Folds eingeteilt. Ein Fold ist eine Sammlung von Featurevektoren. Wenn die Einteilung in k Folds erfolgt ist, dann wird der Klassifikator k mal ausgewertet, einmal mit jedem Fold.
Der, in dieser Arbeit verwendete, Klassifikator ist J48, dabei handelt es sich um eine Java Implementierung von C4.5. Dieser Algorithmus baut während der Lernphase einen Entscheidungsbaum auf. Dazu wird ein Feature ausgewählt. Alle Featurevectoren, deren Wert dieses Feature größer ist, werden zusammen in einen Subtree eingeordnet, alle anderen Featurevectoren in den anderen Subtree. Für Features mit nominalen Werten wird für jeden Wert im Wertebereich ein eigener Unterbaum erzeugt und die Featurevectoren entsprechend zugeordnet. So entsteht der Entscheidungsbaum.
In der Phase der Klassifikation wird dann für jedes zu klassifizierende Objekt der Baum von der Wurzel an traversiert. Entsprechend der Einträge im Featurevector entscheidet der Algorithmus dann, in welchem Unterbaum die Suche fortgesetzt wird. Zum Aufbau des Entscheidungsbaums werden zwei Parameter benötigt, C und M. C ist eine Schwelle, die angibt, aber welcher Gewissheit der Entscheidungsbaum abgeschnitten wird. Je höher C ist, deste eher werden Unterbäume abgeschnitten und desto kleiner ist der Baum. M gibt an, wie viele Objekte in einem Knoten mindestens vorhanden sein müssen. Wenn nach

[1] http://www.cs.waikato.ac.nz/ml/weka/index.html
[2] http://weka.wikispaces.com/ARFF

einer weiteren Aufteilung weniger Objekte in den entstehenden Knoten wären, dann wird keine Aufteilung vorgenommen.[10, chapter 6.1]
In Kapitel 5.1 wird erklärt, wie diese Parameter gewählt wurden.
Bei J48 handelt es sich um einen genauen Klassifikator, der nicht nur Einblicke in die Ergebnisse gewährt, sondern zusätzlich noch den Entscheidungsbaum ausgibt, an dem abgelesen werden kann, welche Features für die Klassifikation besonders relevant waren und welche Features weniger wichtig waren.
Die Qualität der resultierenden Klassifikation ist neben der Wahl der Parameter sehr stark von der Wahl der Einträge des Featurevektors abhängig. Die Ermittlung der Features wird daher in den nächsten Kapiteln sehr detailliert beschrieben.

3.2 Aufbau des Featurevectors

Die Sudokus sollen nach ihrem Schwierigkeitsgrad für menschliche Spieler klassifiziert werden. Für die Featurevectoren werden daher Features gesucht, die sich durch einen Computer aus dem Sudoku extrahieren lassen und etwas über die Schwierigkeit des Sudokus aussagen. Ein solches Feature sind zum Beispiel die von Anfang an vorgegebenen Zahlen. Ein Sudoku, das mehr Zahlen vorgegeben hat, wird im Allgemeinen als leichter empfunden.
Wenn von einer Zahl kein Kandidat vorgegeben ist und von einer anderen Zahl bereits viele Kandidaten von Anfang an ausgefüllt sind, dann wird das Sudoku generell als schwerer empfunden, als Sudokus mit der gleichen Gesamtanzahl an vorgegebenen Kandidaten, die aber gleich verteilt sind. Daher ist es sinnvoll, die Anzahl der vorgegebenen Kandidaten für jede Zahl einzeln zu speichern. Die neun entstehenden Features werden als erstes in den Featurevector eingetragen. Eintrag zwei sagt dann zum Beispiel aus, wie viele Zweien vor dem Lösen im Sudoku stehen.
Als nächstes werden Kandidatenlisten, wie in 2.4.1 angelegt. Diesen Listen kann man entnehmen, an wie vielen Stellen eine bestimmte Ziffer noch stehen kann. Ein Sudoku, dass sehr viele Kandidaten offen lässt, ist im Allgemeinen für menschliche Spieler schwerer. Das ergibt, für jede Ziffer einzeln, wieder neun Features, die in den Featurevector eingetragen werden.
Die Lösungsmethoden können generell in zwei Kategorien aufgeteilt werden. Die erste und einfachere Kategorie füllt Ziffern im Sudoku aus. Die zweite Kategorie Lösungsmethoden schließt Ziffern für bestimmte Felder aus, das bedeutet, dass sie aus den Kandidatenlisten gelöscht werden. Wenn eine Lösungsmethode auf das Sudoku angewendet wurde, dann wird das im Featurevector eingetragen. Hier existieren für jede Lösungsmethode neun Einträge - ein Eintrag für jede Ziffer. Jedes mal, wenn eine Lösungsmethode eine Ziffer ausfüllt, dann wird der entsprechende Eintrag um eins erhöht. Immer, wenn eine Lösungsmethode eine Ziffer aus einer Kandidatenliste entfernt, dann wird auch hier der zugehörige Eintrag im Featurevector um eins erhöht. Um das zu verdeutlichen sehen wir uns folgendes Beispiel an. Angenommen die Lösungsmethode Skyscarper (2.4.10) wird auf ein Sudoku angewendet und entfernt von den Kandidatenlisten von zwei Zellen die Ziffer 4. Dann wird im Featurevector der Abschnitt mit den neun Einträgen für die Lösungsmethode *Skyscarper* gesucht. Aus den neun Einträgen wird der Eintrag für die Ziffer 4 gesucht und sein Wert um zwei erhöht.
Nun stellt sich die Frage, welche der Lösungsmethoden zuerst angewendet wird. Um ein Sudoku mit den in dieser Arbeit beschriebenen Lösungsmethoden zu Lösen, gibt es im Allgemeinen mehrere Wege. Diese entstehen durch eine unterschiedliche Anwendungsreihenfolge der Lösungsmethoden auf das Sudoku. Die Anwendungsreihenfolge ist aber entscheidend für den Aufbau des Featurevectors. Wenn zum Beispiel ein Sudoku nur mit den Methoden *Full House* (2.4.2) und *Naked Single* (2.4.3) gelöst werden kann, dann ist das für den Spieler sehr einfach. Allerdings könnte im Sudoku dennoch eine Stelle vorkommen, an der man *Coloring* (2.4.13) verwenden kann. Wenn bei der Klassifikation im Featurevector vermerkt ist, dass die Methode Coloring verwendet wurde, die großen Aufwand erfordert und die meißt nur von Computern angewendet wird, dann scheint das Sudoku schwerer zu sein, als es tatsächlich war. Das würde das Ergebniss der Klassifikation verfälschen.

Um dieses Problem zu umgehen, wird eine Methode gesucht, um den einfachsten Lösungsweg zu finden. Der einfachste Lösungsweg ist die Folge von Lösungsschritten, die die einfachsten Lösungsschritte enthält, die im jeweiligen Zustand des Sudokus anwendbar waren. Dazu ist es nötig, den Lösungsmethoden einen Schwierigkeitsgrad zuzuweisen.
Die in dieser Arbeit vorgestellten Lösungsmethoden sind bereits von leicht nach schwer sortiert, die zuerst vorgestellten sind die Einfachsten, die zuletzt vorgestellten sind die Schwersten. Bei der Recherche nach dem Schwierigkeitsgrad der Lösungsmethoden, gab es eine Übereinstimmung in verschiedenen Quellen, wenn auch teilweise weniger oder mehr Lösungsmethoden vorgestellt wurden.[4][6][8][9]
Nachdem also der Schwierigkeitsgrad der einzelnen Lösungsmthoden bekannt ist, muss jetzt zu jedem Sudoku der einfachste Lösungsweg gefunden werden. Dazu wird der folgende Algorithmus verwendet.

```
sudoku: the current sudoku;
solvingMethod[]: Array of Solving methods sorted by difficulty;
int solvingMethodCounter = 0;
FeatureVector fv = new FeatureVector();
while sudoku not solved do
    apply solvingMethod[solvingMethodCounter] to sudoku;
    if sudoku changed then
        note changes in fv;
        solvingMethodCounter = 0;
    else
        solvingMethodCounter++;
    end
end
```

Algorithm 1: Build Featurevector

Der Algorithmus versucht also, immer die einfachste Lösungsmethode auf ein Sudoku anzuwenden so lange diese anwendbar ist. Wenn die einfachste Lösungsmethode nicht anwendbar ist, dann wird die nächst schwerere Methode versucht, so lange bis eine Methode anwendbar war und das Sudoku verändert hat. Dann fällt der Algorithmus zurück auf die leichteste Methode, da diese durch die Veränderung im Sudoku anwendbar geworden sein könnte. So wird sicher gestellt, dass immer zuerst die leichtest mögliche Methode angewendet wird und somit wird der leichteste Lösungsweg gewählt.

3.3 Entkopplung von konkreten Zahlen

Aus einem Sudoku kann leicht ein neues Sudoku erzeugt werden, indem man Ziffern tauscht. Das bedeutet zum Beispiel, dass man alle Vorkommen der Ziffer 7 mit der Ziffer 8 ersetzt und umgekehrt. Die Lösung des neuen Sudokus unterscheidet sich von der Lösung des alten Sudokus nur darin, dass dort ebenfalls die Ziffer 8 mit der Ziffer 7 vertauscht ist. An der Schwierigkeit des Sudokus ändert sich nichts. Das spiegelt sich allerdings nicht im Featurevector wieder, da dort die Einträge für die Ziffer 7 und die Ziffer 8 in jeder Lösungsmethode vertauscht sind. Bei einem Sudoku exakt gleicher Schwierigkeit ergibt sich also ein anderer Featurevector.
Das kann verhindert werden, indem die Einträge im Featurevector von den konkreten Zahlen entkoppelt werden. Dazu werden nach der vollständigen Ermittlung des Featurevectors die neun Einträge für jede Lösungsmethode, für die am Anfang vorgegebenen Ziffern und für die am Anfang möglichen Positionen nach ihrer Häufigkeit absteigend geordnet. Das wird am folgenden Beispiel deutlich.
Hier betrachten wir die Lösungsmethode *Coloring* 2.4.13. Vor der Sortierung hat der Vector die diese Darstellung
$(\mathbf{1},\mathbf{0},\mathbf{4},\mathbf{15},\mathbf{3},\mathbf{0},\mathbf{9},\mathbf{2},\mathbf{0})^{\mathbf{T}}$
Das bedeutet zum Beispiel, die Ziffer 3 wurde vier mal mit der Methode *Coloring* ausgeschlossen. Würde man, wie im oben genannten Problemfall, die Ziifern 7 und 8 zu Anfang im Sudoku vertauschen, dann hätte der Vector die Form
$(\mathbf{1},\mathbf{0},\mathbf{4},\mathbf{15},\mathbf{3},\mathbf{0},\mathbf{2},\mathbf{9},\mathbf{0})^{\mathbf{T}}$
Wenn man aber nach der Lösung des Sudokus diesen Vector nach Häufigkeit absteigend sortiert, dann erhält man in beiden Fällen
$(\mathbf{15},\mathbf{9},\mathbf{4},\mathbf{3},\mathbf{2},\mathbf{1},\mathbf{0},\mathbf{0},\mathbf{0})^{\mathbf{T}}$
Der Vector sagt nun aus, dass die am dritthäufigsten von der Methode *Coloring* ausgeschlossene Zahl vier mal ausgeschlossen wurde. Für die Sudokus mit gleicher Schwierigkeit ist der Featurevector nun gleich, hat aber an für die Klassifizierung relevanter Aussagekraft nicht verloren.

3.4 Mapping verschiedener Skalen

Die Schwierigkeitsstufen von Sudokus aus verschiedenen Quellen sind, nach dem Empfinden von menschlichen Spielern, nicht immer identisch. Verschiedene Quellen haben verschiedene Skalen zur Bewertung des Schwierigkeitsgrades, deren Einteilungen die Sudokus nach unterschiedlichen Maßen zugeordnet werden. Daher ist es nützlich, verschiedene Skalen miteinander vergleichen zu können und ähnlich schwere Kategorien einander zuzuordnen. Im Folgenden wird eine Methode vorgestellt, die auf der Klassifikation des Schwierigkeitsgrades der Sudokus mit Hilfe von maschinellem Lernen aufbaut und es ermöglicht, einen Vergleich zwischen zwei unterschiedlichen Skalen zu erstellen.
Angenommen es gibt zwei Sets von Sudokus, **Set1** und **Set2**. Diese haben zwei unterschiedliche Skalen, **Skala1** und **Skala2**, für die Schwierigkeitsgrade. Nun wird ein Klassifizierer mit **Set1** trainiert. Der Klassifizierer kann nun beliebige Sudokus den Klassen aus **Skala1** zuordnen. Das wird benutzt, um die Sudokus aus **Set2** zu klassifizieren. Zu jedem Sudoku aus **Set2** ist nun die ursprüngliche Klasse aus **Skala2** bekannt und zusätzlich auch die vom Klassifizierer zugeordnete Klasse auf **Skala1**. Nun wird für jede Klasse in **Skala2** ausgewertet, welcher Klasse aus **Skala1** die meißten Sudokus zugeordnet werden. Daraus lässt sich schließen, welchem Schwierigkeitsgrad auf **Skala1** die jeweilige Klasse am ehesten entspricht. Wenn das Ergebniss sehr uneindeutig ist, zum Beispiel werden 40% **Klasse1** zugeordnet und 60% werden **Klasse2** zugeordnet, dann ist der Schluss, dass die Schwierigkeit der entsprechenden Klasse zwischen **Klasse1** und **Klasse2** liegt.
Um diese Vorgehensweise zu verdeutlichen, wird das folgende Beispiel betrachtet. Es gibt Sudokus mit bekanntem Schwierigkeitsgrad aus zwei Quellen. Diese Quellen verwenden zur Bewertung des Schwierigkeitsgrades zwei unterschiedliche Skalen. Die erste Quelle teilt Sudokus in die Schwierigkeitsgrade **Leicht**, **Mittel** und **Schwer** ein, die zweite Quelle in **Easy**, **Medium**, **Hard** und **Very Hard**. Der Klassifizierer wird mit den Sudokus aus der ersten Quelle trainiert und kennt daher nur die Bewertungen für die Schwierigkeitsgrade dieser Quelle. Die Einteilung von Sudokus in die Klassen der zweiten Quelle wurde nicht trainiert. Nun wird der Klassifizierer verwendet, um Sudokus aus der zweiten Quelle zu klassifizieren. Als Ergebniss der Klassifikation erhält man für jedes Sudoku der zweiten Quelle einen Schwierigkeitsgrad aus der Skala der ersten Quelle. Angenommen die Klassifikation mit jeweils 100 Sudokus aus jeder Klasse von Quelle 2 liefert das, in **Tabelle 3.1** dargestellte, Ergebniss.

		Zugeordnete Klasse		
		Leicht	Mittel	Schwer
Ursprüngliche Klasse	Easy	90	8	2
	Medium	50	40	10
	Hard	2	78	20
	Very Hard	0	10	90

Tabelle 3.1: Beispiel Resultat

Da 90% aller Sudokus aus der Klasse **Easy** der Klasse **Leicht** zugeordnet wurden, ist es eindeutig, dass diese beiden Klassen annähernd gleich schwer sind. Es gibt hier keine andere Klasse, die der Klasse **Easy** mehr entspricht. Bei der Klasse **Medium** lässt sich keine klare Aussage treffen, da die Ergebnisse mit 50% für **Leicht** und 40% für **Mittel** sehr nahe aneinander liegen. Die Schwierigkeit der Klasse **Medium** liegt zwischen den Schwierigkeiten dieser beiden Klassen. Die Klasse **Hard** entspricht am ehesten der Klasse **Mittel**, allerdings ist eine leichte Tendenz zur Klasse **Schwer** zu erkennen. Daher ist **Hard** etwas schwerer als **Mittel**. Die Klasse **Very Hard** ist eindeutig der Klasse **Schwer** zuzuordnen, da 90% aller Sudokus dementsprechend klassifiziert wurden.

4 Software

Um die Ergebnisse der Klassifikation ermitteln zu können, wurde im Rahmen dieser Bachelorarbeit eine Software entwickelt, die es erlaubt, Sudokus aus Dateien einzulesen und den Featurevector zu extrahieren. Mit den Featurevectoren wird dann einen Klassifizierer trainiert und evaluiert. Ausserdem erlaubt es die Software, dass ein Mapping zwischen zwei verschiedenen Bewertungsskalen vorgenommen wird. Darauf werde ich im Kapitel *Eigene Software* 4.1 eingehen.
Die ganze Software ist in Java geschrieben, der Quellcode ist bei der Abgabe beigelegt. Ausserdem ist die Software momentan verfügbar auf `https://github.com/mbraeunlein/ExtendedHodoku`. Die Software lässt sich einfach über ein Terminal starten.

Für eine *cross validation* ist der Befehl
java -jar ExtendedHodoku cross trainSudokus.txt

Um einen Klassifizierer mit einem Testset zu evaluieren
java -jar ExtendedHodoku test trainSudokus.txt testSudokus.txt

Um ein Mapping zwischen verschiedenen Skalen herzustellen
java -jar ExtendedHodoku map sudokusSkala1.txt sudokusSkala2.txt

Und um die Featurevectoren in eine .arff Datei zu schreiben
java -jar ExtendedHodoku write sudokus.txt

Die Sudokus in den .txt Dateien müssen in einem definierten Format gespeichert werden. In einer Zeile steht jeweils ein Sudoku, bestehend aus 81 Ziffern. Jede Ziffer ist entweder 1 bis 9 für die jeweilige eingesetzte Zahl oder 0 für ein leeres Feld. Die Klassifikation steht jeweils in einer extra Zeile vor den Sudokus. Zum Beispiel steht in einer Datei zu Anfang die Zeile *Einfach*, dann wird die Klassifikation des Schwierigkeitsgrads der folgenden Sudokus *Einfach* sein, bis eine andere Zeile mit einer Klassifikation folgt. Klassifikationen dürfen nicht mit einer Zahl anfangen.

4.1 Eigene Software

Trotz der verwendeten Software ist im Rahmen dieser Arbeit auch eigene Software entwickelt worden. Die Software ist komplett in Java geschrieben. Hier möchte ich kurz auf die einzelnen Komponenten und deren Funktionen eingehen.
Zuerst wurde eine Klasse entwickelt, die es ermöglicht, Sudokus in einem definierten Format 4 einzulesen und in die von Hodoku verwendeten *Sudoku2* Objekte zu parsen. Die nächste, wichtige und selbst entwickelte Funktionalität ist das extrahieren eines Featurevectors. Das grobe Vorgehen ist in 3.2 beschrieben, die Implementiereung findet sich in der Klasse */src/FeatureVectorExtractor.java*. Diese nimmt ein Objekt von *Sudoku2* entgegen und liefert ein FeatureVector Objekt zurück, indem sie das Sudoku schrittweise und mit Hilfe von Hodoku löst. Eine weitere, selbst entwickelte Klasse schreibt eine Menge von Featurevectoren in eine .arff Datei, die von Weka verarbeitet werden kann. Natürlich wäre es möglich gewesen, die Featurevectoren auch direkt an den Klassifizierer zu übergeben, durch das schreiben in eine .arff Datei bleibt dem Nutzer die Möglichkeit erhalten, die Daten später mit Weka genauer zu analysieren.
In *src/analyze/Analyzer.java* wurden die Modi implementiert, die benötigt werden, um Sudokus zu analysieren. Hier wird die Weka library verwendet.
Eine Besonderheit der Software finder sich in der Klasse */src/FeatureVectorExtractor.java*. Hier würde als letzte mögliche Methode *Backtracking* 2.4.15 angewendet werden. Diese löst jedes Sudoku vollständig durch *trial and error*. Es gibt kein Sudoku, das nicht durch *Backtracking* gelöst werden kann. Da diese Methode allerdings das ganze Sudoku auf einmal löst, wird sie nur angewendet, wenn keine andere Methode funktioniert. Für die Klassifikation ist nur relevant, wie viele Zahlen mit *Backtracking* ermittelt wurden. Das kann man auch schon vor dem Anwenden der Methode errechnen, da sie für alle offenen Felder die entsprechenden Zahlen finden wird. Dies kann man ohne Ausführen der Methode in den Featurevector eintragen, was zu einer deutlichen Verbesserung der Laufzeit führt.

4.2 Fremdsoftware

Es wurden zwei andere Peojekte für diese Arbeit verwendet.
Zum Generieren und schrittweisen Lösen von Sudokus wurde Hodoku[1] benutzt. Dieses Programm steht unter der GPLv3 Lizenz[2].
Zur Klassifikation der Featurevectoren wurde Weka[3] verwendet. Weka steht unter der GNU General Public License[4].

4.3 Trainingsdaten

Um fundierte Aussagen über die Qualität der Klassifikation treffen zu können, wird eine große Menge an Trainingsdaten benötigt. Diese müssen bereits vollständig in Schwierigkeitsstufen eingeteilt worden sein. Das ist nötig, da der Klassifikator eine Schwierigkeitsstufe zuordnet und die Qualität der Zuordnung evaluiert werden soll. Um also festzustellen, ob der Klassifikator die richtige Klasse zugeordnet hat, muss diese bekannt sein.
Kostenlose und frei verfügbare Sudokus in digitaler Form mit definiertem Schwierigkeitsgrad lassen sich nicht leicht finden. Daher habe ich bei einigen großen Zeitungen, aufs deren Websites Sudokus zu finden waren, nachgefragt, ob es möglich ist, ihre Sudokusammlungen zur Verfügung zu stellen. Die Anfragen wurden aber leider abgelehnt. Auf eine Anfrage an die Website `http://sudoku.soeinding.`

[1] `http://hodoku.sourceforge.net/de/index.php`
[2] `http://www.gnu.org/licenses/gpl-3.0.html`
[3] `http://www.cs.waikato.ac.nz/ml/weka/`
[4] `http://www.gnu.org/licenses/gpl.html`

de/ wurden von sieben Schwierigkeitsgraden jeweils 32 Sudokus bereitgestellt. Da diese Trainigsdaten nicht ausreichten wurden mit dem open source Programm Hodoku[5] jeweils 1000 Sudokus von fünf unterschiedlichen Schwierigkeitsgraden generiert.
Wie schon an der Anzahl der Schwierigkeitsstufen zu erkennen ist, unterscheiden sich die Skalen der beiden Quellen. Daher konnten die Sudokus nicht gemeinsam klassifiziert werden. Auch war es nicht möglich, eine Quelle als Trainingsdaten für den Klassifikator zu verwenden um ihn anschließend mit der anderen Menge auszuwerten. Allerdings kann man eine Verbindung zwichen den Skalen suchen, zum Beispiel Klassen mit gleich schweren Sudokus.

[5] http://hodoku.sourceforge.net/de/index.php

5 Ergebnisse

5.1 Optimierung der Parameter

Um die Ergebnisse der Klassifikation zu verbessern, wurden die Parameter für den J48 Algorithmus optimiert. Dabei handelt es sich um die Parameter C und M, deren Einfluss in Kapitel 3.1 beschrieben wurde. Der Suchraum wurde diskretisiert und anschließend wurden 1000 Kombinationen der Parameter getestet. Dabei wurden in die Featurevectoren alle beschriebenen Features eingetragen. Getestet wurde mit 5000 Sudokus aus fünf verschiedenen Klassen mit 10-Fold cross validation. Als Maß für die Güte der Klassifikation wurde der prozentuale Anteil der korrekt klassifizierten Sudokus verwendet.

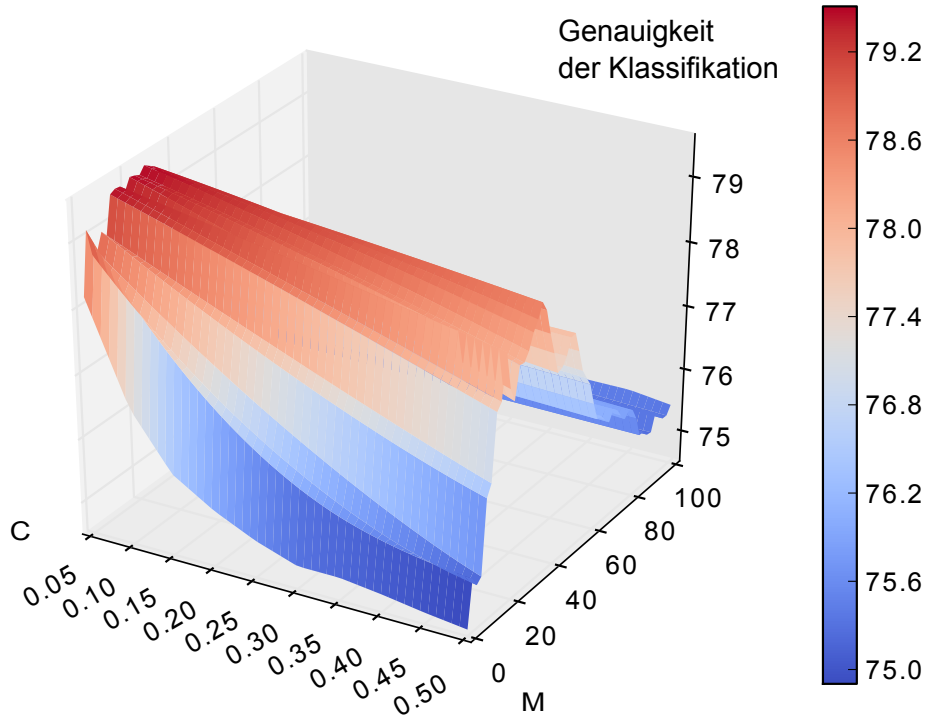

Abbildung 5.1: Anzahl der korrekt klassifizierten Sudokus mit verschiedenen Kombinationen von C und M

Wie die in **Abbildung 5.1** abgebildeten Ergebnisse zeigen, liegen die optimalen Parameter für den Algorithmus für C bei 0.1 und für M bei 30. Dies gilt nur für die in diesem Test verwendeten Sudokus und Featurevectoreinträge, für neue Sudokus können die optimalen Parameter variieren.

5.2 Evaluierung des Featurevectors

Der Featurevector, der in dieser Arbeit verwendet wird, enthält 261 Merkmale. Wie diese ermittelt werden, wurde in Kapitel 3.2 beschrieben. Jedes dieser Features soll dazu beitragen, die richtige Klasse für ein Sudoku zu finden.

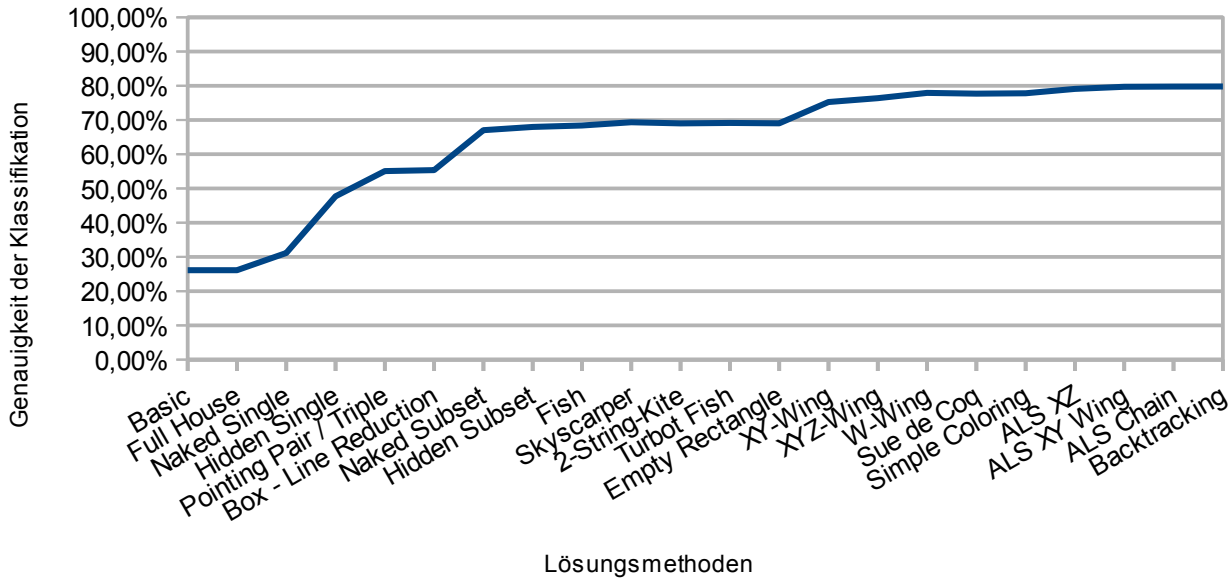

Abbildung 5.2: Genauigkeit des Klassifizierers bei Hinzunahme von Lösungsmethoden

In **Abbildung 5.2** ist auf der X-Achse abgebildet, welche Features in den Featurevector aufgenommen werden. Alle bis dahin aufgenommenen Features bleiben im Featurevector, die neuen Features werden angehängt.
Zur Auswertung wurden 5000 Sudokus aus fünf unterschiedlichen Klassen verwendet. Das bedeutet, bei einer Einteilung in die Klassen mit Hilfe eines Zufallsgenerator läge die Genauigkeit im Schnitt bei 20%. Wenn man im Featurevector nur die Merkmale hat, auf die ohne das Lösen des Sudokus gefunden werden können, dann liegt die Genauigkeit bereits bei über 26%. Durch die Hinzunahme der Lösungsmethoden, die im Sudoku Ziffern ausfüllen, wird eine Geanuigkeit von 47,7% erzielt. Um dieses Ergebniss genauer interpretieren zu können, ist ein Blick auf die Konfusionsmatrizen des Klassifikators notwendig.

		predicted class					
		a	b	c	d	e	
actual value	a	1000	0	0	0	0	a = Easy
	b	0	568	115	93	224	b = Middle
	c	0	288	333	221	158	c = Hard
	d	0	309	257	225	209	d = Unfair
	e	0	415	149	177	259	e = Extreme

Abbildung 5.3: Konfustionsmatrix mit einschließlich *Hidden Single* Methode

Wie man an **Abbildung 5.3** ablesen kann, ist der Klassifikator für Klasse a fehlerfrei und für b deutlich besser als für c, d und e. Der Klassifikator funktioniert also für einfache Sudokus besser als für schwere. Das ist so, weil, in der momentanen Konfiguration, nicht alle Features im Featurevector sind, sondern nur die, die durch einfache Lösungsmethoden errechnet werden können.

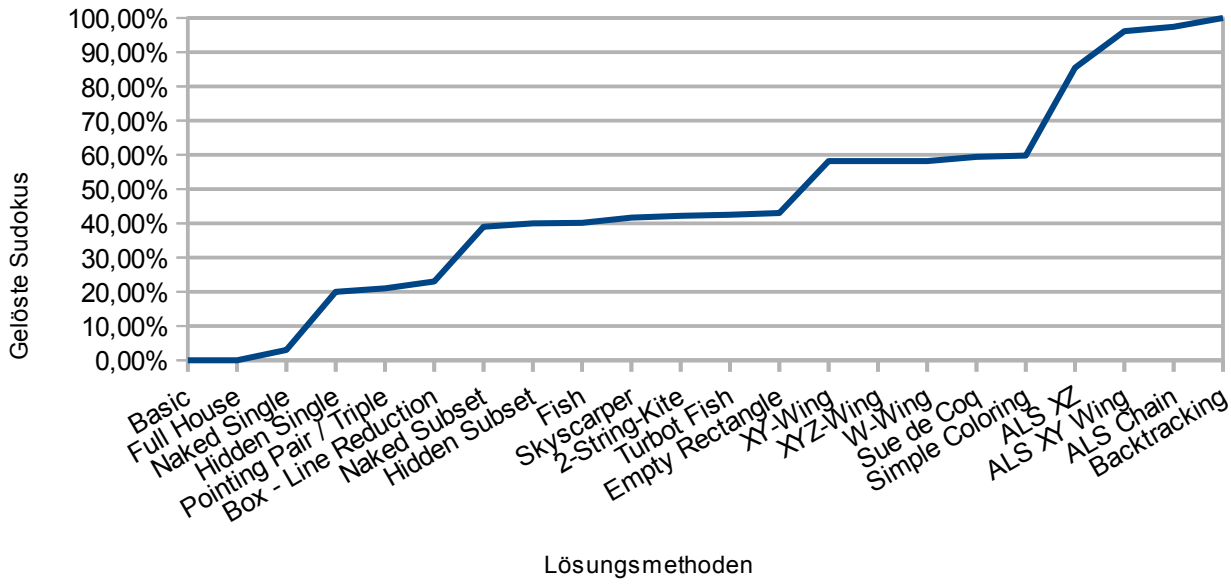

Abbildung 5.4: Anzahl der gelösten Sudokus bei Hinzunahme von Lösungsmethoden

Aus **Abbildung 5.4** kann man entnehmen, dass die Anteil der gelösten Sudokus steigt, je mehr Lösungsmethoden angewendet werden. Da momentan nur Lösungsmethoden bis zur Technik *Hidden Single* (2.4.4) angewendet werden, können nur leichte Sudokus (20% der Gesamtmenge) gelöst werden und damit ist nur deren Featurevector vollständig. Daher funktioniert die Unterschiedung nur zwischen leichten und schwereren Sudokus, eine genaue Unterscheidung innerhalb der schwereren Sudokus ist dem Klassifikator derzeit nicht möglich, da die Featurevectoren zu ähnlich sind. Mit der Hinzunahme von schwereren Lösungsmethoden lässt sich dieses Problem beheben.
Wenn man die Features in den Featurevector aufnimmt, die von etwas schwereren Methoden generiert werden, dann sieht man wieder eine Steigerung der Genauigkeit in **Abbildung 5.3**. Nimmt man die Methoden bis einschließlich *Hidden Subset* (2.4.8) dazu, dann liegt der prozentuale Anteil der richtig klassifizierten Sudokus bei 67,98%. Es können nun auch deutlich mehr Sudokus gelöst werden, genau 2005 Stück, was 40,1% der gesamten Testmenge entspricht.
Hierdurch bedingt wird auch die Unterscheidung zwischen Sudokus in schwereren Klassen deutlicher. Da 40% der Sudokus gelöst werden können, ist auch ihr Featurevector vollständig. Wenn ein Sudoku mit den angewendeten Lösungsmethoden nicht gelöst werden kann, dann wird der bis dorthin erzeugt Featurevector dem Klassifikator übergeben. Da es sich aber hier nur um Informationen handelt, die leichte Lösungsmethoden betreffen, ähneln sich die Featurevectoren der unlösbaren Sudokus stark.

actual value \ predicted class	a	b	c	d	e	
a	1000	0	0	0	0	a = Easy
b	0	1000	0	0	0	b = Middle
c	1	4	518	249	228	c = Hard
d	1	0	312	299	388	d = Unfair
e	0	0	174	244	582	e = Extreme

Abbildung 5.5: Konfustionsmatrix mit einschließlich *Hidden Subset* Methode

In **Abbildung 5.5** sieht man die Konfusionsmatrix, die vom Klassifikator erzeugt wird, wenn man Lösungsmethoden bis einschließlich *Hidden Subset* (2.4.8) zum Lösen der Sudokus verwendet.
Hier sieht man eine deutliche Verbesserung in der Klasse b. Alle Sudokus, die aus Klasse b kommen, werden auch richtig als solche erkannt.
Mit den bisher verwendeten Lösungsmethoden können Sudokus aus den Klassen a und b komplett gelöst werden, aus Klasse c werden von 1000 Sudokus nur fünf gelöst und in den Klassen d und e kein Einziges.
Im Vergleich zu **Abbildung 5.3** ist eine Verbesserung in Klasse c sichtbar, über die Hälfte der Sudokus aus Klasse c wurden korrekt klassifiziert, wobei es davor nur ein drittel waren. Das liegt vor allem daran, dass Sudokus aus Klasse c nicht mehr fälschlicherweise Klasse b zugeordnet werden. Die Sudokus aus Klasse d sind sehr stark in andere Klassen gestreut, jedoch werden die Sudokus fast ausschließlich in benachbarte Klassen eingeordnet.
Man kann hier deutlich erkennen, dass es eine starke Korrelation zwischen vollständig gelösten Sudokus und korrekt klassifizierten Sudokus gibt. Das liegt an der Vollständigkeit der Featurevectoren für die ersten beiden Klassen. Die starke Streuung in Klasse d resultiert daraus, dass es für diese Klasse keine klare Abgrenzung in eine Richtung gibt, was bei den Klasse c und e anders ist.
Der nächste große Sprung im Gewinn der Klassifikationsgenauigkeit ist nach dem Hinzunehmen der *Wing-Techniken* (2.4.11) zu erkennen. Hier steigt der Anteil der korrekt klassifizierten Sudokus auf knapp 78%. Aus der Klasse c können nun 799 von 1000 Sudokus gelöst werden. Bei der Klasse d sind es fast 100 Sudokus und in Klasse e 13. Der Anteil der gelösten Sudokus steigt auf über 58%, wie man auf **Abbildung 5.4** ablesen kann.
In der Zugehörigen Konfusionsmatrix in **Abbildung 5.6** sieht man einen Anstieg der Klassifikationsge-

actual value \ predicted class	a	b	c	d	e	
a	1000	0	0	0	0	a = Easy
b	0	1000	0	0	0	b = Middle
c	24	28	750	144	54	c = Hard
d	4	1	104	507	384	d = Unfair
e	0	0	17	344	639	e = Extreme

Abbildung 5.6: Konfustionsmatrix mit einschließlich *W-Wing* Methode

nauigkeit vor allem in den Klassen c und d, einen leichten Anstieg gibt es auch in Klasse e. Dies begründet sich damit, dass Sudokus der Klasse c nun in den meißten Fällen gelöst werden können. Dadurch ist ihr Featurevector meißtens vollständig und damit besser von den Featurevectoren der anderen Klassen zu unterscheiden. Der Anstieg in der Klasse d kommt zustande, da Sudokus aus der Klasse d nun kaum noch fälschlicherweise in die Klasse c kategorisiert werden, da sich durch das Lösen von Sudokus aus Klasse c die Featurevectoren mehr voneinander unterscheiden.

Wenn man nun alle in dieser Arbeit verwendeten Lösungsmethoden in dem, in Kapitel 3.2 beschriebenen, Algorithmus verwendet, dann werden 100% der Sudokus gelöst. Allerdings müssen 2,6% davon mit *Backtracking* gelöst werden. *Backtracking* ist ein Sonderfall der Lösungsmethoden. Es trägt nicht eine Ziffer ein oder löscht einige Kandidaten aus den Kandidatenlisten der Felder wie andere Lösungsmethoden, *Backtracking* löst das ganze Sudoku auf einmal. Daher lassen sich daraus nicht viele Informationen über den Schwierigkeitsgrad des Sudokus ziehen. Da *Backtracking* erst verwendet wird, wenn keine andere Lösungsmethode mehr funktioniert, kann man sagen, dass das Sudoku für die meißten menschlichen Spieler unlösbar wäre. Dazu ist zu sagen, dass es noch andere Lösungsmethoden gibt, die auch einige schwerere Sudokus lösen können und dass Backtracking auch unter sehr großem Zeitaufwand von menschlichen Spielern durchgeführt werden kann. Betrachtet man den Informationsgewinn nach dem Hinzufügen der *W-Wing* Methode bis zum *Backtracking*, dann sieht man einen Genauigkeitsgewinn von unter 2%. Um diesen Gewinn zu erzielen, mussten 54 Features in den Featurevector aufgenommen werden. Der Gewinn ist also sehr niedrig im Vergleich zum Aufwand, obwohl zusätzliche 38% der Sudokus gelöst werden können, wobei hier *Backtracking* nicht betrachtet wurde.

		predicted class					
		a	b	c	d	e	
actual value	a	1000	0	0	0	0	a = Easy
	b	0	999	1	0	0	b = Middle
	c	1	35	785	145	34	c = Hard
	d	0	10	129	636	225	d = Unfair
	e	0	0	14	367	619	e = Extreme

Abbildung 5.7: Konfustionsmatrix mit allen vorgestellten Lösungsmethoden

Die Konfusionsmatrix, die erzeugt wird, wenn alle vorgestellten Methoden angewendet werden, ist in **Abbildung 5.7** zu sehen. Hier sieht man eine Verbesserung in den Klassen c und d, wobei eine leichte Verschlechterung in der Klasse e stattfindet. 116 Sudokus der Klasse e können mit den verwendeten Lösungsmethoden ohne *Backtracking* nicht gelöst werden. Die schlechteren Ergebnisse in dieser Klasse gegenüber den Klassen mit leichterem Schwierigkeitsgrad können mit der nicht Vollständigkeit des Featurevectors begründet werden.

5.3 Ergebnisse des Mappings

Wie bereits in Kapitel 3.4 beschrieben, wurde ein Mapping zwischen zwei Skalen durchgeführt. Hierzu werden Sudokus aus zwei Quellen benötigt. Das erste Set besteht aus Ingesammt 224 Sudokus, jeweils 32 Sudokus in sieben Schwierigkeitsstufen. Als zweites Set werden wieder die 5000 Sudokus aus fünf Schwierigkeitsgraden verwendet. Zunächst wird der Klassifikator mit dem ersten Set trainiert. Er kennt daher die fünf Klassen **Sehr Einfach**, **Einfach**, **Standard**, **Moderat**, **Anspruchsvoll**, **Sehr Anspruchsvoll** und **Teuflisch**. In diese Klassen werden nun die Sudokus des zweiten Sets durch den Klassifikator eingeteilt.

predicted class

actual value	a	b	c	d	e
a	1000	0	0	0	0
b	0	999	1	0	0
c	16	35	770	145	34
d	5	10	129	631	225
e	2	14	27	367	590

5.4 Relative Güte der Features

6 Zusammenfassung und Ausblick

Literaturverzeichnis

[1] B. Hobiger. Hodoku als xy wing. http://hodoku.sourceforge.net/de/tech_als.php#axy, April 2014.

[2] B. Hobiger. Hodoku simple coloring. http://hodoku.sourceforge.net/de/tech_col.php, April 2014.

[3] B. Hobiger. Hodoku sue de coq. http://hodoku.sourceforge.net/de/tech_misc.php#sdc, April 2014.

[4] A. Johnson and M. Strasser. Simple sudoku schwierigkeitsgrade. http://blechtrottel.net/sudoku.html, April 2014.

[5] Silurian Software Limited. Origin of sudoku. http://www.sudokudragon.com/company.htm, March 2014.

[6] V. Martin and T. Martin. Kristanix games schwierigkeitsgrade. http://www.kristanix.com/sudokuepic/sudoku-solving-techniques.php, April 2014.

[7] Ed Jr. Pegg and Eric W. Weisstein. Sudoku. http://mathworld.wolfram.com/Sudoku.html, March 2014.

[8] A. Rühl. Ahr software schwierigkeitsgrade. http://www.ahr-sudoku.de/solving.php/list/difficulty, April 2014.

[9] G. Vanhegan. Playrcouk schwierigkeitsgrade. http://www.playr.co.uk/sudoku/ratings.php, April 2014.

[10] Ian H. Witten, Eibe Frank, and Mark A. Hall. *Data Mining: Practical Machine Learning Tools and Techniques - Practical Machine Learning Tools and Techniques*. Elsevier, Amsterdam, 3. aufl. edition, 2011.