Implementierung und Testen eines relationalen Lern-Algorithmus

Bachelor-Thesis von Christian Reuter
Oktober 2009

TECHNISCHE UNIVERSITÄT DARMSTADT

Fachbereich Informatik
Fachgebiet Knowledge Engineering

Implementierung und Testen eines relationalen Lern-Algorithmus

vorgelegte Bachelor-Thesis von Christian Reuter

1. Gutachten: Prof. Dr. Johannes Fürnkranz
2. Gutachten: -

Tag der Einreichung:

Erklärung zur Bachelor-Thesis

Hiermit versichere ich die vorliegende Bachelor-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 21.10.2009

(Christian Reuter)

Zusammenfassung

Im Rahmen dieser Bachelorarbeit wurde eine vorhandene Implementierung des relationalen Lern-Algorithmus FOIL so angepasst, dass sie zusätzlich zu dem Weighted Information Gain zwei weitere Heuristiken zur Bewertung von Regeln verwenden kann. Hierbei handelte es sich um das m-estimate und Laplace. Außerdem wurde es ermöglicht, den Parameter des m-estimates anzupassen.

Anschließend wurde die erweiterte Implementierung mit verschiedenen Beispieldatensätzen getestet. Anhand dieser Tests wurden die Heuristiken abschließend miteinander verglichen und untersucht, welchen Einfluss die Wahl des Parameters auf das Ergebnis hat.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

FOIL ist ein relationaler Lern-Algorithmus, das heißt er nimmt als Eingabe positive und negative Beispiele für Fakten und leitet aus diesen allgemeingültige Regeln ab. Die Beispiele werden dabei in Form von Tupeln[1] aus Konstanten angegeben, die, im Falle der positiven Beispiele, Teil der zu lernenden Relation[2] sind. Seit seiner Entwicklung 1990 wurde er für eine Reihe von Problemstellungen erfolgreich eingesetzt [QC95].

Dabei geht er so vor, dass dem *Separate-and-Conquer*-Prinzip folgend solange neue Regeln erzeugt werden, bis alle positiven Beispiele erklärt sind. Jede einzelne Regel wird dabei mit einer *Hill-Climbing-Suche* so lange verfeinert, also mit Literalen ergänzt, bis sie nicht mehr verbessert werden kann. Dies ist in der Regel dann der Fall, wenn sie keine negativen Beispiele mehr abdeckt. Sind die Eingabedaten allerdings mit einem Rauschen behaftet, weisen also Ungenauigkeiten auf, kann der Algorithmus eine Regel auch schon als vollständig ansehen, obwohl diese noch negative Beispiele abdeckt.

Bei dieser Suche werden alle möglichen Ergänzungen einer Regel zunächst mit der Heuristik *Weighted Information Gain* bewertet und das Literal, das die höchste Bewertung erhalten hat, an die Regel angehängt. Neuere Forschungsarbeiten schlagen jedoch die Verwendung von anderen Heuristiken vor, die beispielsweise besser mit dem bereits erwähnten Rauschen zurechtkommen.

Ziel dieser Bachelorarbeit ist es daher, eine vorliegende Referenzimplementierung von FOIL [FOIL] so anzupassen, dass diese zwei der vorgeschlagenen Heuristiken, das *m-estimate* und *Laplace*, verwenden kann. Dem Benutzer soll es dabei über Parameter ermöglicht werden, für jeden Aufruf des Programms eine dieser Heuristiken auszuwählen. Auch der für das m-estimate nötige Parameter soll auf diese Art gesetzt werden können.

Anschließend werden diese Heuristiken im Rahmen von Testläufen miteinander verglichen. Dazu werden aus Testdatensätzen unter Verwendung der unterschiedlichen Heuristiken Regeln gelernt sowie deren Präzision auf den Trainings- und Validierungsdatensätzen miteinander verglichen. Die Größe der gelernten Theorien und die Laufzeit der Heuristiken werden ebenfalls betrachtet. Zudem wird untersucht, welche Auswirkung die Wahl des Parameters für das m-estimate hat.

1.2 Übersicht

Im Folgenden wird zunächst der aktuelle Stand der Forschung betrachtet. Das Hauptaugenmerk liegt dabei auf einer Analyse des aktuellen Standes von FOIL. Dieser wird dabei als Erstes klassifiziert und sein Vorgehen zu dem von anderen Lern-Algorithmen abgegrenzt. Anschließend wird sein Aufbau beschrieben und einige erfolgreiche Anwendungen aufgelistet. Danach werden das in FOIL verwendete Weighted Information Gain sowie zwei dazu vorgeschlagene Alternativen, das m-estimate und Laplace, vorgestellt. *mFOIL* [DB92], eine Variante von FOIL die diese Heuristiken verwendet, wird ebenfalls betrachtet und mit dem originalen Algorithmus verglichen.

Als nächstes wird die Anpassung der Implementierung beschrieben. Dazu wird zuerst die vorhandene Referenzimplementierung von FOIL analysiert. Dies geschieht analog zur Abarbeitungsreihenfolge in drei Phasen: Eingabe und Initialisierung, Konstruktion einer Theorie und Nachbereitung. Darauf folgend werden die theoretischen Überlegungen vorgestellt, die nötig waren, um die neuen Heuristiken implementieren zu können.

Es folgt eine Beschreibung der Testläufe, in denen die unterschiedlichen Heuristiken verglichen wurden. Dabei werden zunächst die Testmethode sowie die verwendeten Datensätze beschrieben. Anschließend werden die Testergebnisse dargestellt und anhand dieser die einzelnen Heuristiken bewertet. Außerdem wird aufgeschlüsselt, welche Auswirkungen die Wahl des Parameters für das m-estimate hat.

[1] Ein Tupel ist eine geordnete Gruppe von Werten.

[2] Als Relation bezeichnet man eine Menge von Tupeln, deren Elemente in einer bestimmten Beziehung zueinander stehen.

Zuletzt werden die erzielten Ergebnisse zusammengefasst und ein Ausblick auf mögliche Vertiefungen des Themas gegeben.

Zusätzlich befinden sich Informationen bezüglich der Verwendung der erweiterten Implementierung im Anhang. Dort werden ebenfalls alle veränderten Dateien aufgeführt, um die Anpassungen nachvollziehbar zu machen.

2 Stand der Forschung

2.1 FOIL

FOIL wurde ursprünglich 1990 von J. R. Quinlan entwickelt [Q90] und ist seitdem mehrfach weiterentwickelt worden [FOIL].

2.1.1 Klassifikation

Das Programm ist ein *induktiver Lern-Algorithmus*. Algorithmen dieser Klasse, auch *First-Order Learning Systems* genannt, bekommen eine Reihe von Beispieltupeln für die sogenannte Zielrelation und eventuell dazugehöriges Hintergrundwissen, ebenfalls in Form von Tupeln, als Eingabe. Positive Beispiele, also Tupel die Teil der Zielrelation sind, sind essentiell. Negative Beispiele können entweder explizit angegeben oder aus der *Closed-World-Assumption* gewonnen werden. Diese besagt, dass alles falsch ist, was nicht als wahr angegeben wurde. Dementsprechend sind alle Tupel, die aus den gegebenen Konstanten erzeugt werden können und nicht als positives Beispiel deklariert wurden, negative Beispiele. Die Beispiele und das Hintergrundwissen sind extensional definiert, das heißt es werden die in der Relation enthaltenen Tupel explizit angegeben.

First-Order Learning Systems entwickeln daraus eine Theorie, welche die hinter den Beispielen stehende Zielrelation erklärt [QC95]. Diese Theorie sollte möglichst komplett sein, also alle positiven (Vollständigkeit) und keine negativen Beispiele (Konsistenz) abdecken[1]. Die Theorie sollte auch bei einer endlichen Menge von Beispielen über diese hinaus gültig sein. Beispielsweise sollte eine aus den natürlichen Zahlen von 0 bis 10 gelernte Theorie für „even“ auch die Zahl Zwölf als gerade erkennen. Sie definiert die Relation intensional, das heißt in Form einer Konzeptbeschreibung durch Regeln, und wird durch ein logisches Programm ausgedrückt.

Im Gegensatz dazu verarbeiten *Zeroth-Order Learning Systems* wie *C4.5*, das Quinlan vor FOIL entwickelt hat, vorklassifizierte Fälle in Form von Wertetupeln und stellen die entwickelte Theorie als Entscheidungsbaum dar [QC93].

2.1.2 Kurzbeschreibung

Nachdem FOIL die Beispiele eingelesen hat, konstruiert es eine auf ihnen basierende Theorie. Dabei nutzt es eine *Separate-and-Conquer-Strategie*, das heißt der Algorithmus iteriert über die zu entwickelnden Regeln. Dabei werden von Anfang an alle Beispieltupel berücksichtigt und der Algorithmus entwickelt in jedem Schritt eine Regel, die eine Teilmenge der positiven Beispiele abdeckt. Diese werden dann entfernt, und das Programm terminiert, sobald alle positiven Beispiele durch mindestens eine Regel erklärt werden. Dem gegenüber steht die *Successive-Revision-Methode*, welche über die Tupel iteriert. Es wird zunächst eine komplette Theorie entwickelt, die ein positives Tupel erklärt. In jeder Iteration wird diese so angepasst, dass sie ein weiteres positives Beispiel abdeckt. Diese Verfeinerung ist in der Praxis aufwändig zu berechnen, wurde aber dennoch in früheren Systemen wie *CIGOL* verwendet. Ferner hängt die Qualität der entwickelten Theorie bei dieser Methode sehr von den verwendeten Beispieltupeln ab [QC95].

Bei der Entwicklung einer Regel nutzt FOIL einen Top-Down-Ansatz. Er geht zunächst von einer allgemeinen Regel aus und verfeinert diese durch das Hinzufügen von Literalen[2] solange, bis sie keine negativen Beispiele mehr abdeckt. Bottom-Up-Systeme dagegen gehen von einer sehr spezifischen Regel aus, die eine sehr kleine Teilmenge der positiven Beispiele abdeckt. Diese wird anschließend durch das Entfernen von Literalen solange generalisiert, wie sie keine negativen Beispiele abdeckt. Zu dieser Gruppe von Systemen gehört beispielsweise *GOLEM*. Beide Varianten der Separate-and-Conquer-Methode wurden bereits erfolgreich für große Problemstellungen eingesetzt und waren dort deutlich schneller als Algorithmen mit der Successive-Revision-Methode [QC95].

Bei der Verfeinerung nutzt FOIL eine Hill-Climbing-Suche. Es wird immer genau das Literal ausgewählt, das die beste Bewertung durch eine Heuristik erhält. Diese Form einer Suche sorgt dafür, dass Minima schnell gefunden werden (ein

[1] Eine Theorie in Prolog deckt ein Tupel genau dann ab, wenn die entsprechende Grundanfrage durch das Prolog-Programm herleitbar ist.

[2] Ein Literal besteht aus einem atomaren Term oder seiner Negation.

Minima ist im Kontext von FOIL eine Regel, die nicht weiter verbessert werden kann). Dabei steigt allerdings das Risiko, lokale Minima zu finden. Diese können zwar nicht mehr weiter verfeinert werden, aber es gibt in der Gesamtmenge der möglichen Regeln bessere Alternativen (die globalen Minima).

Die Bewertung aller potenziellen Ergänzungen einer Regel erfolgt mit der Heuristik Weighted Information Gain. Dabei werden anstatt der abgedeckten Beispieltupel die korrespondierenden Variablenbelegungen gezählt. Dadurch wird berücksichtigt, wie oft sich ein Beispiel mit dieser Regel beweisen lässt. Da der Körper einer Regel neue Variablen einführen kann, ist es möglich, dass die Anzahl der möglichen Belegungen für ein Tupel beim Verfeinern zunimmt.

Während der Suche werden verschiedene Forwad-Pruning-Methoden verwendet. Diese stellen sicher, dass niedrig bewertete Literale möglichst früh erkannt und nicht weiter berücksichtigt werden.

FOIL besitzt außerdem einen Mechanismus, der abschließend alle gelernten Regeln nachbereitet. Dabei werden beispielsweise unnötige Regeln und Literale erkannt und entfernt. Durch dieses *Pruning* werden die Regeln kürzer und sind leichter verständlich.

Quinlan hat den FOIL-Algorithmus in C implementiert und bereits mehrfach weiterentwickelt. Die aktuelle Version trägt die Nummer 6.4 und stammt aus dem Jahre 1996 [FOIL].

Eine genauere Beschreibung des Algorithmus findet sich in Abschnitt 3.1.

2.1.3 Anwendungen

FOIL wurde von Quinlan mit einer Reihe von Beispielanwendungen erfolgreich getestet: rekursive Listenfunktionen, arithmetische Funktionen, Klassifizierungsprobleme, Analyse von Proteinstrukturen, Identifikation von Komponenten eines Dokumentes und Analyse der Züge in einem Schachendspiel [QC95].

Im Falle der Listenfunktionen konnte ein Großteil der 18 getesteten Funktionen gelernt werden, wobei einige aber nicht allgemeingültig und nur in der Welt der Trainingsbeispiele korrekt waren. Dennoch konnte gezeigt werden, dass FOIL rekursive Konzepte wie den Quicksort-Algorithmus lernen kann.

Bei den arithmetischen Funktionen ist hervorzuheben, dass FOIL die Ackermann-Funktion lernen kann. Diese ist besonders interessant, da eine der dafür nötigen Regeln eine doppelte Rekursion beinhaltet, die jeweils unterschiedliche Argumente verkleinert. Laut Quinlan ist FOIL dabei der erste Algorithmus, der diese Definition lernen konnte [QC95]. Erwähnenswert ist ebenfalls, dass die gelernten Definitionen allgemeingültig waren, obwohl die Trainingsdaten nur Zahlen von 0 bis 20 beinhalteten.

Angewendet auf Klassifizierungsprobleme erreicht FOIL etwas präzisere Ergebnisse als C4.5, benötigt aber eine längere Berechnungszeit. Dabei stellte sich heraus, dass FOIL dazu tendiert, komplexere Theorien zu lernen. Es neigt außerdem zu Overfitting, das heißt es passt die Theorie zu genau an die Trainingsdaten an und ist daher nicht allgemeingültig genug.

Dieses Ergebnis bestätigte sich bei der Analyse von Proteinstrukturen. Hier erreichten die von GOLEM gelernten Regeln eine bessere Genauigkeit. GOLEM wurde dabei allerdings mit einem domänenspezifischen Kriterium zur Vermeidung von Overfitting ergänzt.

Eine weitere Anwendung war die Identifikation von Komponenten innerhalb eines Dokumentes, beispielsweise des Absenderfeldes. Die Testdaten bestanden dabei aus einer Reihe von Beziehungen der Komponenten untereinander, sodass dieses Problem für Zeroth-Order Systeme mit Vektoren fester Länge nicht geeignet ist. FOIL erreichte dabei Genauigkeiten von 96,3 % bis 100 %, abhängig von der untersuchten Komponente.

Zuletzt wurde das Schachendspiel König und Turm gegen König untersucht und versucht, die maximal verbleibende Zuganzahl bei optimalem Spiel vorauszusagen. FOIL lernte hierbei fast ausschließlich korrekte Regeln, nur in einem Subproblem (matt in elf Zügen) wurde ein negatives Beispiel abgedeckt. Verglichen mit *GCWS*, einer Modifikation von GOLEM, lernte FOIL kürzere Regeln. Diese deckten aber nicht alle positiven Beispiele ab.

Basierend auf diesen Ergebnissen und der bisherigen Entwicklung beurteilt Quinlan FOIL als „gereiften“ Algorithmus, der aber noch nicht für große Probleme aus der realen Welt geeignet ist. Dies wollte er jedoch durch zukünftige Forschungen erreichen [QC95].

2.2 Heuristiken

Es gibt unterschiedliche Heuristiken, die verwendet werden können um die „Güte“ einer Regel zu bewerten. Darunter versteht man ihre Fähigkeit, möglichst viele positive und wenige negative Trainingsbeispiele abzudecken.

2.2.1 Weighted Information Gain

Die von FOIL genutzt Heuristik ist das Weighted Information Gain. Es ist entropiebasiert, bewertet also den Informationsgehalt von Regeln. Dabei berechnet es zunächst die Anzahl der Bits, die benötigt werden um zu spezifizieren, dass eine abgedeckte Belegung zu einem positiven Beispiel gehört (n^+ und n^- beschreiben die Anzahl von Belegungen einer Regel, die zu positiven beziehungsweise negativen Beispielen führen) [LD94]:

$$I(n^+, n^-) = -log_2 \frac{n^+}{n^+ + n^-}$$

Wird die Regel verfeinert, sodass sie n'^+ beziehungsweise n'^- Belegungen abdeckt, werden weniger Bits benötigt, um oben genannte Information darzustellen. Somit beträgt der Informationsgewinn:

$$IG(n^+, n^-, n'^+, n'^-) = I(n^+, n^-) - I(n'^+, n'^-) = log_2 \frac{n'^+}{n'^+ + n'^-} - log_2 \frac{n^+}{n^+ + n^-}$$

Dieser Informationsgewinn wird abschließend mit dem Faktor k gewichtet, wobei k Anzahl von Belegungen aus n^+ ist, die auch durch die Verfeinerung abgedeckt werden. Dadurch erhalten Regeln, die eine größere Menge von positiven Beispielen abdecken, eine höhere Bewertung:

$$WIG(k, n^+, n^-, n'^+, n'^-) = k \cdot \left(log_2 \frac{n'^+}{n'^+ + n'^-} - log_2 \frac{n^+}{n^+ + n^-} \right)$$

2.2.2 Alternative Heuristiken

In [LD94] werden alternative Heuristiken zum Bewerten einer Regel vorgestellt. Diese werden insbesondere für Anwendungsbereiche empfohlen, in denen die Beispiele Ungenauigkeiten (auch „Noise“ genannt) aufweisen. Dort kann es passieren, dass sich die Regeln zu genau an die ungenauen Trainingsdaten anpassen und das Programm ein zu spezielles und daher insgesamt schlechteres Ergebnis produziert.

Precision

Anstatt wie im Weighted Information Gain den Informationsgehalt einer Regel zu bewerten, kann deren Genauigkeit[3] bestimmt werden. Diese stellt die Wahrscheinlichkeit dar, mit der ein von der Regel abgedecktes Beispiel positiv ist:

$$P(n^+, n^-) = \frac{n^+}{n^+ + n^-}$$

Mit diesem Konzept kann, analog zum Weighted Information Gain, der *Weighted Precision Gain* bestimmt werden:

$$WPG(k, n^+, n^-, n'^+, n'^-) = k \cdot \left(\frac{n'^+}{n'^+ + n'^-} - \frac{n^+}{n^+ + n^-} \right)$$

m-estimate

In der vorherigen Berechnung wurde die relative Frequenz $\frac{n'^+}{n'^+ + n'^-}$ als Abschätzung für die Wahrscheinlichkeit, dass ein abgedecktes Beispiel positiv ist, verwendet. Diese Schätzung wird allerdings ungenau, falls die Trainingsmenge nur aus wenigen Beispielen besteht. Daher wird vorgeschlagen, sie durch das *m-estimate* zu ersetzen.

[3] Von Lavrac und Dzeroski [LD94] Accuracy genannt.

Das m-estimate berechnet die Wahrscheinlichkeit, nach n^+ positiven Beispielen ein weiteres positives zu finden:

$$\frac{n^+ + m \cdot p(+)}{n^+ + n^- + m} = \frac{n^+ + m \cdot \frac{N^+}{N^+ + N^-}}{n^+ + n^- + m}$$

N^+ und N^- stellen hierbei die Anzahl der positiven und negativen Beispiele in der Trainingsmenge dar. Dementsprechend steht $p(+)$ für die geschätzte Wahrscheinlichkeit dafür, dass ein zufällig gewähltes Beispiel aus der Trainingsmenge positiv ist. m ist ein frei wählbarer Parameter, der die Zuverlässigkeit der Trainingsbeispiele widerspiegelt. Ein großer Wert für m sorgt dafür, dass diese Wahrscheinlichkeit einen starken Einfluss auf die Bewertung hat und ist daher für ungenaue Trainingsbeispiele geeignet.

Das m-estimate eignet sich dadurch im Gegensatz zu der relativen Frequenz auch für kleine Trainingssets. Des Weiteren konnte es in einem Vergleich verschiedener Heuristiken [JF08] bei optimal gewählten Parametern die höchste Genauigkeit erzielen.

Für die im Rahmen dieser Arbeit erstellten Implementierung wurde zusätzlich zwischen zwei Varianten des m-estimates, der Absolut- und der Gain-Variante, unterschieden. Bei erstgenannter stammen N^+ und N^- aus dem Grundproblem. Bei der Gain-Variante entsprechen diese Werte den Tupeln, die nach Anwendung der aktuellen (partiellen) Theorie vor Anhängen des neuen Literals noch nicht abgedeckt sind.

Laplace

Laplace ist ein Spezialfall des m-estimates, bei dem eine Gleichverteilung von positiven und negativen Beispielen ($N^+ = N^-$) angenommen wird. Außerdem wird $m = 2$ gesetzt, so dass sich folgende Abschätzung ergibt:

$$\frac{n^+ + 2 \cdot \frac{N^+}{N^+ + N^+}}{n^+ + n^- + 2} = \frac{n^+ + 2 \cdot \frac{N^+}{2 \cdot N^+}}{n^+ + n^- + 2} = \frac{n^+ + 1}{n^+ + n^- + 2}$$

2.3 mFOIL

mFOIL [DB92], 1991 von Dzeroski entwickelt, ist eine Variante von FOIL, die sich durch besseres Noise-Handling auszeichnet. Zu diesem Zweck wurde das Weighted Information Gain durch das bereits erwähnte m-estimate beziehungsweise Laplace ersetzt, zwischen denen über einen Parameter gewechselt werden kann. Diese Heuristiken werden dabei sowohl für die Suche nach neuen Literalen als auch als Abbruchkriterium verwendet. mFOIL zählt zudem Beispiele anstatt Belegungen. Dadurch soll verhindert werden, dass aus einem fehlerhaften Beispiel mehrere Belegungen entstehen und sich der Fehler so verstärkt.

Zusätzlich wird an Stelle der Hill-Climbing-Suche eine Beam-Suche verwendet. Dies bedeutet, dass nicht nur die aktuell beste Regel, sondern eine festgelegte Anzahl von signifikanten Regeln verfeinert wird. Dadurch entstehen pro Iteration mehrere Verfeinerungen, von denen die mit der größten Verbesserung der Genauigkeit weiter verfeinert werden. Durch dieses Vorgehen sollen das Erreichen von lokalen Minima vermieden werden.

Ob eine Regel signifikant ist, wird dabei durch Verwendung eines χ^2-ähnlichen Signifikanztestes entschieden, der auf vergleichbare Art und Weise in *CN2* [CN89], einem weiteren induktiven Lern-Algorithmus, verwendet wird. Dabei wird die Wahrscheinlichkeit berechnet, mit der ein von der Regel abgedecktes Beispiel positiv ist. Unterschreitet diese einen bestimmten Grenzwert, wird die dazugehörige Regel nicht weiter untersucht. mFOIL bricht die Suche nach Verfeinerungen ab, sobald der Beam leer ist, also keine Verbesserung der Regeln mehr möglich ist. Anschließend wird geprüft, ob die Genauigkeit der erzeugten Regel größer als der Anteil von positiven Beispielen in der Trainingsmenge ist. Ist dies nicht der Fall, besitzt die Regel eine gleiche Genauigkeit wie die Default-Regel[4] und wird daher verworfen.

Im Gegensatz zu FOIL muss das Hintergrundwissen nicht ausschließlich aus Grundfakten, also Literalen ohne Variablen, bestehen. Die Trainingsbeispiele und das Hintergrundwissen dürfen ebenfalls Funktionen beinhalten, diese werden aber zu einer einzigen Variablen generalisiert. Das bedeutet, dass beispielsweise `f(a, b, c)` nicht zu `f(X, Y, Z)` generalisiert werden kann.

[4] Eine Regel mit gleichem Kopf und leerem Körper, die für alle (positiven und negativen) Beispiele wahr ist.

Ein weiterer Unterschied zu FOIL besteht bei den Möglichkeiten, den Suchraum zu reduzieren. FOIL erlaubt es, Negate auszuschließen und die Anzahl beziehungsweise Tiefe von neuen Variablen zu begrenzen. mFOIL nutzt im Gegensatz dazu Symmetrien innerhalb der Relationen aus[5]. Literale können außerdem als „rectified“ definiert werden, sodass eine gleiche Belegung der Variablen ausgeschlossen ist. Bei beiden Algorithmen ist es zudem möglich, an bestimmten Stellen einer Relation keine ungebundenen Variablen[6] zu erlauben. Alle diese Constraints werden vom Benutzer vor Programmstart definiert und zusammen mit den Eingabedaten übergeben.

mFOIL kann, wie auch FOIL, rekursive Regeln lernen. Dies bedeutet, dass bei ihnen eine Relation sowohl im Kopf als auch im Körper einer Regel verwendet wird [7]. Es verwendet jedoch ein schwächeres Kriterium, um unendliche Rekursionen zu vermeiden. Während FOIL dazu eine partielle Ordnung von Literalen erstellt, verhindert mFOIL nur, dass der Kopf einer Regel in deren Körper erneut verwendet wird. Eine unendliche Rekursion wie `same(A, B) :- same(B, A).` kann dadurch nicht verhindert werden.

Des Weiteren behandelt mFOIL deterministische Literale nicht gesondert. Diese führen neue Variablen mit nur einer möglichen Belegung ein.

Experimente von Lavrac und Dzeroski [LD94] zeigen, dass mFOIL bei einem korrekt gewählten Parameter m weniger anfällig für Noise ist als FOIL. Auch wenn die Einführung von Noise bei mFOIL die Genauigkeit der gelernten Regeln verringert, erreichen diese eine insgesamt höhere Genauigkeit als bei FOIL. Dies gilt sowohl für gestörte Variablenbelegungen als auch für falsch klassifizierte Beispiele.

Während FOIL in C implementiert wurde, ist mFOIL in Quintus Prolog geschrieben. Daraus resultiert eine deutlich höhere Laufzeit, beispielsweise bei dem „mesh design“-Problem zwei Stunden gegenüber mehreren Minuten [LD94].

Diese geringere Performance stellt auch einen der beiden Gründe dar, warum für diese Arbeit die C-Referenzimplementierung von FOIL um die Heuristiken m-estimate und Laplace ergänzt wurde, obwohl mFOIL diese bereits beinhaltet. Ein weiterer Vorteil ist, dass für diese nur die Heuristiken verändert werden, während mFOIL beispielsweise auch eine andere Suchstrategie verwendet. So kann besser geprüft werden, welche Auswirkungen die alleinige Wahl der Heuristik hat.

[5] Da `same(A, B)` und `same(B, A)` das gleiche aussagen, wird mFOIL nur eines dieser beiden Literale untersuchen.

[6] Variablen werden als ungebunden bezeichnet, wenn sie bisher weder im Kopf der Regel noch im Körper einer Regel verwendet werden.

[7] Ein Beispiel hierfür ist folgende Regel aus der Member-Relation für Listen: `member(A, B) :- components(B, C, D), member(A, D).`

3 Anpassung der Implementierung

3.1 Vorhandene Implementierung von FOIL

Um den Algorithmus modifizieren zu können, wurde zuerst die aktuelle Referenzimplementierung näher untersucht. Dabei wurde diese analog zur Abarbeitungsreihenfolge in drei Phasen zerlegt.

3.1.1 Eingabe und Initialisierung

Das zu bearbeitende Problem wird dem Programm in Form einer Datei übergeben. Zudem können beim Programmstart unterschiedliche Parameter gesetzt werden, die das Vorgehen des Algorithmus beeinflussen.

Eingabedatei

Zu jedem Problem erhält der Algorithmus genau eine Eingabedatei. Diese beinhaltet zunächst alle Konstanten, die in der entsprechenden Domäne auftreten und in den Beispieltupeln verwendet werden. Sie können zusätzlich als sogenannte „Theoriekonstanten“ markiert werden, sodass FOIL sie auch in den gelernten Regeln verwenden kann.

Die Konstanten sind nach Typen sortiert und können entweder explizit angegeben oder über ein Schlüsselwort einem kontinuierlichen Wertebereich zugeordnet werden. Dieser umfasst alle reellen Zahlen. Um in den gelernten Regeln mit Vergleichsoperatoren wie „<“ arbeiten zu können und um unendliche Rekursionen zu vermeiden, benötigt FOIL zusätzlich Wissen darüber, ob ein Typ geordnet ist. Eine vorhandene Ordnung kann in der Datei angegeben werden. Es besteht aber auch die Möglichkeit, dass der Algorithmus auf Basis der folgenden Beispieltupel eine solche findet, sofern diese existiert. Dabei werden alle angegebenen Relationen daraufhin untersucht, ob sie eine partielle Ordnung $\prec$ auf einem Argumentenpaar darstellen. Ist dies der Fall und existiert keine Konstante k mit $k \prec k$, betrachtet FOIL den dazugehörigen Typ als geordnet [QC95].

Darauf folgend müssen die gesondert markierte Zielrelation und eventuell vorhandene Hintergrundrelationen aufgeführt werden. Für die Zielrelation soll der Algorithmus eine intensionale Definition lernen. Die Hintergrundrelationen stellen Hintergrundwissen aus der Domäne dar und dürfen in den entstehenden Regeln verwendet werden. Alle Relationen müssen extensional, als Tupel aus Konstanten, definiert werden. Die Argumente der Relationen werden außerdem an einen Konstantentyp gebunden und FOIL stellt beim Einlesen sicher, dass alle Tupel diese Bindung erfüllen. Ferner ist es an dieser Stelle möglich, den Suchraum der späteren Hill-Climbing-Suche zu beschränken. Für jedes Argument einer Relation kann festgelegt werden, dass an dieser Stelle später nur bereits gebundene Variablen eingesetzt werden dürfen.

Negative Beispieltupel, also Tupel die der entsprechenden Relation nicht angehören, können auf die gleiche Weise angegeben werden. Ist dies nicht der Fall, werden diese aus der Closed-World-Assumption generiert.

Zusätzlich zu den manuell definierten Konstanten können die Tupel noch zwei vordefinierte beinhalten. Die erste davon symbolisiert einen unbekannten Wert und wird für Domänen benötigt, in denen unvollständige Informationen vorkommen können. FOIL lernt anschließend keine Regeln, die diese Konstante explizit beinhalten und ignoriert diese beim Finden einer partiellen Ordnung.

Mit Version 6.1 wurde außerdem eine sogenannte „out-of-wold“-Konstante eingefügt, die besagt, dass ein Wert außerhalb der Closed World liegt. Dies kann beispielsweise beim Lernen von Listenfunktionen auftreten. Da es beliebig lange Listen gibt, ist die dazugehörige Relation prinzipiell unendlich und es ist nicht möglich, sie extensional zu definieren. Daher wird bei der Definition nur eine begrenzte Welt, zum Beispiel mit Listen aus maximal drei Elementen, berücksichtigt. In diesem Fall liegt eine vierelementige Liste außerhalb dieser Welt. Dabei wird die Annahme getroffen, dass innerhalb der eingeschränkten Welt gelernte Regeln auch für Listen beliebiger Länge gültig sind (*open domain assumption* [BW03]). Um dies zu gewährleisten, werden Literale, die diese Konstante beinhalten, beim Lernen der Regeln ignoriert [QC95].

Abschließend können optional Testfälle spezifiziert werden, mit denen die gelernten Regeln überprüft werden. Da FOIL

die Regeln nicht mit Hilfe eines Prolog-Interpreters interpretiert, sondern für Rekursionen die angegebenen Beispieltupel verwendet [FOIL], werden diese allerdings nicht immer korrekt klassifiziert. Ausgehend von der Definition `greater(A, B) :- greater(A, C), greater(C, B).` kann ein Testfall `greater(7, 5): +` nur korrekt als erfüllt erkannt werden, wenn sowohl `greater(7, 6)` als auch `greater(6, 5)` als positive Beispiele für greater angegeben wurden.

Parameter beim Programmstart

Beim Start des Algorithmus kann über verschiedene Parameter Einfluss auf die Ausführung genommen werden. So ist es möglich, die Generierung von negativen Beispieltupeln aus der Closed-World-Assumption zu beschränken. FOIL generiert dann nur noch eine zufällige Auswahl aller möglichen negativen Tupel, wenn diese nicht explizit angegeben wurden. In Domänen mit Relationen von hoher Arität, also mit vielen Argumenten, ist dies nötig, da sonst sehr viele Tupel generiert werden würden.

Der Suchraum der Hill-Climbing-Suche kann ebenfalls eingeschränkt werden. Es ist möglich, negative Literale auszuschließen und die Anzahl von neu eingeführten Variablen zu begrenzen. Außerdem kann eine maximale Variablentiefe festgelegt werden. Diese ist so definiert, dass alle im Kopf einer Regel vorkommenden Variablen die Tiefe 0 haben. Wird im Körper eine neue Variable eingeführt, hat diese die maximale Tiefe aller schon vorhandenen Variablen + 1. Die Anzahl der „schwachen“ Literale in einer Regel, das sind Literale ohne ausreichende Bewertung, kann ebenfalls begrenzt werden. Dabei wird eine Folge von deterministischen Literalen nur als ein schwaches Literal betrachtet.

Es können auch andere Eigenschaften der Suche über Parameter geändert werden. Diese betreffen unter anderem die Anzahl an alternativen Ergänzungen einer Regel, die zwischengespeichert werden, und die Abweichung, die deren Bewertung von der des besten Literals aufweisen darf. FOIL hängt zudem automatisch alle deterministischen Literale an eine Regel an, sobald nicht ein bestimmter Prozentsatz der maximal möglichen Bewertung von mindestens einem Literal erreicht wird. Dieser Grenzwert lässt sich ebenfalls steuern.

Nach dem Lernen einer Theorie prüft FOIL deren Genauigkeit. Liegt diese unterhalb eines festgelegten Wertes, wird sie verworfen. Auch dieser Grenzwert lässt sich beim Programmstart festlegen.

3.1.2 Konstruktion einer Theorie

Wie bereits erwähnt, ist es das Ziel von FOIL, eine Theorie zu finden, welche die Zielrelation intensional definiert. Ausgedrückt wird diese in einen Dialekt von *Prolog*. Der Dialekt verwendet kein „cut“, „fail“, „disjunctive goals“ und Funktionen (ausgenommen Konstanten). Das Fehlen von Funktionen bedeutet allerdings insofern keine Einschränkung, da es möglich ist, n-äre Funktionen in (n+1)-äre Relationen umzuwandeln[1]. „Not“ ist vorhanden und wird im Sinne von Prolog als *negation by failure* interpretiert, das heißt es ist alles falsch, was nicht bewiesen werden kann. Eine limitierte Quantifikation, beispielsweise für Existenzaussagen[2], wird ebenso unterstützt wie Vergleiche zwischen Variablen oder mit einer Konstanten. FOIL verwendet in den Regeln außerdem Tupel aus dem Hintergrundwissen und kann rekursive Konzepte lernen. Dabei lässt der Algorithmus nur solche Rekursionen zu, bei denen die Terminierung der Regeln sichergestellt ist.

Separate & Conquer

Zu der Konstruktion der Theorie verwendet der Algorithmus eine Separate & Conquer-Strategie. Das bedeutet, dass er in der äußeren Schleife über die Regeln der Theorie iteriert. Dabei werden zunächst alle positiven und negativen Beispieltupel berücksichtigt. In jedem Iterationsschritt wird von FOIL mit Hilfe einer Hill-Climbing-Suche jeweils eine Regel konstruiert. Ist diese komplett, werden alle positiven Beispiele, die sie abdeckt, aus der Beispielmenge entfernt. Dies ist zulässig, da sie ja bereits durch die zuletzt gelernte Regel abgedeckt werden und daher nicht mehr berücksichtigt werden müssen. Die Theorie ist dann komplett, wenn auf diese Weise alle positiven Beispiele entfernt wurden. Ist dies der Fall oder kann die Suche keine weitere Verbesserung der Regeln finden, wird die Iteration beendet.

[1] Beispielsweise lässt sich die Funktion `f(a, b) = c` als Relation `f(a, b, c)` ausdrücken.

[2] „Es existiert eine Liste mit x als erstem Element“ ist ein Beispiel für eine Existenzaussage.

Bei der Konstruktion einer einzelnen Regel startet FOIL eine Hill-Climbing-Suche, ausgehend von der Default-Regel. Zu dieser werden so lange Literale hinzugefügt, bis sie keine negativen Beispiele mehr abdeckt.

Der Suchraum umfasst dabei folgende Arten von Literalen [QC95]:
$R(V_1, ..., V_n)$ und $not(R(V_1, ..., V_n))$, wobei R eine Relation aus dem Hintergrundwissen oder die Zielrelation ist. V_1 bis V_n sind Variablen, die zu den Typconstraints der entsprechenden Relation passen. Mindestens eine Variable muss dabei gebunden sein. Außerdem dürfen zwei Variablen V_i und V_j nicht gleich sein, wenn nicht mindestens ein Tupel in der Beispielmenge diese Gleichheit widerspiegelt. Bei Rekursionen gibt es noch weitere Einschränkungen, die in einem späteren Absatz näher erläutert werden.
$V_i = V_j$, $V_i \neq V_j$, $V_i = C$ und $V_i \neq C$. C ist dabei eine Theoriekonstante und V_i, V_j und C sind vom gleichen Typ.
$V_i \leq V_j$, $V_i > V_j$, $V_i \leq T$ und $V_i > T$. T ist hier ein vom Algorithmus gewählter Grenzwert und sowohl V_i als auch V_j haben einen Typ, dem ein kontinuierlicher Wertebereich zugeordnet ist.
Dieser Suchraum kann beim Programmstart so eingegrenzt werden, dass $not(R(V_1, ..., V_n))$, $V_i \neq V_j$ und $V_i \neq C$ ignoriert werden. Damit wird erreicht, dass die gelernten Regeln Horn-Klauseln entsprechen[3].

Bei der Auswahl des nächsten Literals unterscheidet FOIL zwischen zwei Arten, gewinnbringend („gainful“) und deterministisch („determinante“). Gewinnbringende Literale sind solche, die bei Anwendung der ausgewählten Heuristik (siehe Abschnitt 2.2) eine positive Bewertung erhalten und daher die Abdeckung einer Regel verbessern. Die erweiterte Regel verfügt über weniger Belegungen, die zu negativen Beispieltupeln gehören und ist daher näher an ihrer Komplettierung. FOIL verhält sich dabei gierig („greedy“), das heißt es wählt immer das am besten bewertete Literal aus.

Deterministische Literale hingegen besitzen immer genau eine Belegung für jedes positive und maximal eine für jedes negative Beispieltupel. Sie werden verwendet, um neue Variablen einzuführen, die für später folgende Literale verwendet werden können. Sie sind insofern nützlich, dass sie die Abdeckung von positiven Belegungen nicht reduzieren und die Anzahl an möglichen Variablenbindungen nicht erweitern [QC95]. Diese Art von Literalen wird ohne weitere Prüfung an die Regel angehängt, sobald kein gewinnbringendes Literal eine ausreichend hohe Bewertung erhält.

Bei Rekursionen stellt der Algorithmus sicher, dass die entstehenden Regeln terminieren, also keine unendliche Rekursion beinhalten. Dabei nutzt er eine eventuell vorhandene partielle Ordnung auf den Konstanten der in den Relationen verwendeten Datentypen und versucht mit ihnen, eine Ordnung auf den Argumenten einer Relation zu finden. Gilt in allen Beispieltupeln für zwei Elemente $E_i \prec E_j$ (wobei $\prec$ ein beliebiger Vergleichsoperator sein kann), dann besteht zwischen den korrespondierenden Variablen der Relation eine partielle Ordnung $V_i \prec V_j$. Aus dieser leitet er eine partielle Ordnung auf den Literalen her. Bei zwei Literalen V und V' hat diese die Gestalt $(V_a \prec V'_a) \vee (V_a = V'_a \wedge V_b \prec V'_b) \vee ...$ für geeignete Positionen a, b, et cetera.

Rekursionen werden nur dann erlaubt, wenn das Literal im Körper der Regel im Sinne dieser Ordnung kleiner ist als das im Kopf. Dieses Vorgehen ist konservativ, es lehnt alle Rekursionen ab, bei denen es die Terminierung nicht garantieren kann. Es erlaubt zudem nur rekursive Definitionen auf einer Relation. Wechselseitige Rekursionen von mehreren Relationen sind nicht möglich. Dennoch bezeichnet Quinlan das Verfahren als unbedingt nötig und effizient in der Praxis [QC95].

Während der Suche speichert FOIL eine feste Anzahl Checkpoints, sobald das beste gewinnbringende Literal nur minimal besser ist als eine Alternative. Wird im weiteren Verlauf klar, dass mit der konstruierten Regel nicht alle negativen Belegungen ausgeschlossen werden können, kehrt der Algorithmus zu dem besten gespeicherten Checkpoint zurück (nicht-chronologisches Backtracking). Dies ist allerdings nur selten nötig, da die greedy-Strategie in den meisten Fällen gute Ergebnisse liefert [QC95].

Literale, die eine Regel komplettieren würden, aber wegen ihrer Bewertung nicht ausgewählt werden, speichert FOIL ebenfalls zwischen. Nach regulärer Vollendung der Regel wird diese dann mit der gespeicherten Alternative verglichen und die im Bezug auf Abdeckung und Kompaktheit bessere Variante für die Theorie verwendet.

Das Vorgehen wird ebenfalls modifiziert, sobald FOIL ein Literal anhängt, das nur Variablen aus dem Kopf der Regel verwendet. Dieses wird an den Anfang gestellt und alle nichtdeterministischen Literale, die Variablen eingeführt haben, werden entfernt. Dies rechtfertigt Quinlan damit, dass das neue Literal auch zuerst hätte eingeführt werden können,

[3] Horn-Klauseln sind Formeln von der Gestalt $\neg a \vee ... \vee \neg y \vee z$. Dies lässt sich unter Verwendung des De Morgan'schen Gesetzes umformen zu $\neg(a \wedge ... \wedge y) \vee z$, was äquivalent zu $(a \wedge ... \wedge y) \rightarrow z$ ist. In Prolog ausgedrückt entspricht dies $z :- a, ..., y.$.

während die entfernten eventuell die Abdeckung der Regel verringert haben [QC95]. Anschließend wird die Suche normal fortgeführt.

FOIL setzt Forward-Pruning ein, das heißt es bricht die Evaluation von Literalen möglichst früh ab, sobald eindeutig ist, dass sie nicht deterministisch sind und zu viele positive Belegungen ausschließen. Zu diesem Zwecke wird geprüft, ob die Bewertung des Literals unter der Annahme, dass es keine negativen Tupel abdecken wird, einen bestimmten Prozentsatz der maximal möglichen Bewertung nicht unterschreitet.

Eine zweite Pruning-Methode basiert auf dem *Minimum Description Length Principle* [QR89]. Dieses besagt, dass die Kodierungskosten einer Regel die Kosten der expliziten Darstellung aller abgedeckten Tupel nicht übersteigen darf. Die Kosten einer Regel setzen sich dabei aus den Bits für alle Literale im Körper der Regel zusammen. Diese teilen sich wiederum in die Kosten für die jeweilige Relation und deren Argumente auf. Deterministische Literale werden hierbei ausgenommen, da sie ohne Bewertung eingeführt wurden und daher nicht unbedingt für die Regel notwendig sein müssen. Die Kosten für eine explizite Darstellung der Tupel werden aus dem Zweierlogarithmus der Möglichkeiten, die Anzahl der abgedeckten Tupel aus der Gesamtmenge aller Tupel auszuwählen, berechnet. Berücksichtigt werden dabei sowohl positive als auch negative Beispieltupel. Sind die Kodierungskosten einer expliziten Darstellung geringer als die der Regel, ist diese zu komplex im Bezug auf die Trainingsdaten und wird daher verworfen.

3.1.3 Nachbereitung

Nachdem eine Regel komplettiert wurde, wird diese nachbereitet. Dabei wird zunächst versucht, implizite Gleichheiten explizit zu machen. Eine solche ist beispielsweise im Literal `member(A, [1])` vorhanden, es muss `A = 1` gelten. FOIL untersucht nach Abschluss einer Regel die von ihr abgedeckten Belegungen auf solche Gleichheiten und fügt diese als neues Literal hinter der Stelle, an der die dazugehörige Variable einführt wird, ein. Da die folgenden Pruning-Maßnahmen die Regel von hinten abarbeiten, wird das neue Literal dadurch möglichst lange behalten. Durch diese explizite Darstellung der Gleichheiten vereinfacht das folgende Pruning die Regel stärker [QC95].

Danach wird die Regel auf unnötige Literale untersucht. Dies sind Literale, die weder negative Belegungen ausschließen, noch für andere Literale nötige Variablen einführen oder eine Rekursion absichern. Sie entstehen hauptsächlich durch das ungeprüfte Anhängen von deterministischen Literalen. Entfernt man sie, wird die Regel kürzer und kann eventuell mehr positive Belegungen abdecken. Dabei entfernt FOIL aus Gründen der Effizienz zunächst gleichzeitig alle deterministischen Literale, die keine neuen Variablen einführen. Deckt die Regel nach dieser Modifikation negative Belegungen ab, wird die Änderung zurückgenommen und die Literale einzeln betrachtet. Dabei durchläuft der Algorithmus die Regel von hinten nach vorne und versucht, das jeweilige Literal auf die gleiche Art zu entfernen. Dieses Vorgehen ist bei langen Regeln aufwändig, bringt allerdings den Vorteil mit sich, dass verkürzte Regeln unter Umständen mehr positive Belegungen abdecken und daher für die vollständige Theorie weniger Regeln gelernt werden müssen [QC95].

Ist die Theorie vollständig, kann es positive Beispiele geben, die von mehreren Regeln abgedeckt werden. Auch wenn die korrespondierenden Tupel, nachdem sie abgedeckt sind, aus der Trainingsmenge entfernt werden, ist nicht ausgeschlossen, dass später gelernte Regeln ebenfalls auf sie zutreffen. Daher überprüft FOIL abschließend, ob alle gelernten Regeln mindestens ein positives Tupel alleine abdecken und entfernt solche, bei denen dies nicht zutrifft. Dadurch verringert sich die Größe der Theorie und sie wird für den Menschen verständlicher.

Zuletzt werden die Regeln umgeformt, um die Verständlichkeit noch weiter zu erhöhen. Dabei wird die Transitivität der Gleichheit ausgenutzt um Literale der Form $V_i = V_j$ und $V_i = C$ zu eliminieren. Aus der Regel `relation(A, B, C) :- A = x, B = C.` würde durch Substitution von `A` und `C` die Regel `relation(x, B, B).` entstehen.

3.2 Vorgenommene Anpassungen

Um die nötigen Änderungen am Programm vornehmen zu können, mussten außer der Bewertungsfunktion auch andere Berechnungen verändert werden, die mit dieser zusammenhängen.

3.2.1 Neue Bewertungsfunktion

Die Bewertungsfunktion selbst wurde so angepasst, dass sie, abhängig von einem beim Programmstart gesetzten Parameter, eine Regel wahlweise mit dem Weighted Information Gain, dem m-estimate oder Laplace bewertet. Im Falle des

m-estimates kann auf diese Weise außerdem der Wert des Parameters m bestimmt und zwischen der Absolut- und der Gain-Variante ausgewählt werden.

FOIL sorgt dafür, dass „Literale ohne Gain“ immer eine geringe Bewertung bekommen, wenn sie deterministisch sind. Dieses Verhalten musste für die neuen Heuristiken angepasst werden. Während beim Weighted Information Gain die Bewertung relativ zur vorherigen Regel erfolgt, verwenden die neuen Heuristiken eine davon unabhängige Bewertung. Daher bekommen Literale ohne Gain keine Bewertung von 0 und müssen entsprechend anders erkannt werden.

Bei der Absolut-Variante und Laplace ist ein „Literal ohne Gain“ eines, das die gleiche Bewertung wie die Default-Regel erhält. Diese deckt alle positiven und negativen Beispiele ab, daher gilt für sie ($n^+ = N^+$ und $n^- = N^-$). Wird die Gain-Variante verwendet, wird ein „Literal ohne Gain“ gleich der Vorgängerregel bewertet. In beiden Fällen ist der Vergleichswert die Basis, auf die der geringe Bonus addiert wird, falls das Literal deterministisch ist.

Zusätzlich muss zu jeder Bewertungsfunktion deren Umkehrfunktion implementiert werden. Diese berechnet die maximale Anzahl von abgedeckten negativen Tupeln n^{-*}, mit denen eine Regel bei einer festen Anzahl positiver Tupel n^{+*} mindestens eine bestimmte Bewertung *MinUsefulGain*[4] erhält. Für das m-estimate ergibt sich dabei folgende Formel:

$$\begin{aligned}
\frac{n^{+*} + m \cdot \frac{N^+}{N^+ + N^-}}{n^{+*} + n^{-*} + m} &\geq MinUsefulGain \\
\frac{n^{+*} + m \cdot \frac{N^+}{N^+ + N^-}}{MinUsefulGain} &\geq n^{+*} + n^{-*} + m \\
\frac{n^{+*} + m \cdot \frac{N^+}{N^+ + N^-}}{MinUsefulGain} - (n^{+*} + m) &\geq n^{-*}
\end{aligned}$$

Im Spezialfall Laplace vereinfacht sich diese Formel auf:

$$\begin{aligned}
\frac{n^{+*} + 2 \cdot \frac{N^+}{N^+ + N^+}}{MinUsefulGain} - (n^{+*} + 2) &\geq n^{-*} \\
\frac{n^{+*} + 2 \cdot \frac{N^+}{2 \cdot N^+}}{MinUsefulGain} - (n^{+*} + 2) &\geq n^{-*} \\
\frac{n^{+*} + 1}{MinUsefulGain} - (n^{+*} + 2) &\geq n^{-*}
\end{aligned}$$

3.2.2 Anpassung der Grenzwerte

Die Heuristik beeinflusst ebenfalls die Berechnung von $MinPos$, welches als Grenzwert für das Forward-Pruning verwendet wird. Dieser Wert steht dabei für die Anzahl positiver Tupel, die mindestens abgedeckt werden müssen, um eine Bewertung von *MinUsefulGain* zu erreichen unter der Annahme, dass gleichzeitig keine negativen Tupel abgedeckt werden. Im Falle des m-estimates gilt daher:

$$\begin{aligned}
\frac{MinPos + m \cdot \frac{N^+}{N^+ + N^-}}{MinPos + 0 + m} &= MinUsefulGain \\
\frac{MinPos + m \cdot \frac{N^+}{N^+ + N^-}}{MinUsefulGain} &= MinPos + m \\
\frac{m \cdot \frac{N^+}{N^+ + N^-}}{MinUsefulGain} - m &= MinPos - \frac{MinPos}{MinUsefulGain} \\
\frac{m \cdot \frac{N^+}{N^+ + N^-} - m \cdot MinUsefulGain}{MinUsefulGain} &= MinPos \cdot \left(1 - \frac{1}{MinUsefulGain}\right) \\
\frac{m \cdot \left(\frac{N^+}{N^+ + N^-} - MinUsefulGain\right)}{MinUsefulGain \cdot \left(1 - \frac{1}{MinUsefulGain}\right)} &= MinPos
\end{aligned}$$

[4] Diese Bezeichnung stammt aus der Referenzimplementierung und wurde beibehalten, obwohl bei Laplace und der Absolut-Variante kein „Gain“ berechnet wird.

$$\frac{m \cdot \left(\frac{N^+}{N^+ + N^-} - MinUsefulGain\right)}{MinUsefulGain - 1} = MinPos$$

Entsprechend gilt für Laplace:

$$\begin{aligned}
\frac{2 \cdot \left(\frac{N^+}{N^+ + N^+} - MinUsefulGain\right)}{MinUsefulGain - 1} &= MinPos \\
\frac{2 \cdot \left(\frac{1}{2} - MinUsefulGain\right)}{MinUsefulGain - 1} &= MinPos \\
\frac{1 - 2 \cdot MinUsefulGain}{MinUsefulGain - 1} &= MinPos
\end{aligned}$$

Da der Wertebereich des m-estimates und Laplace zwischen 0 und 1 liegt, muss auch *MinUsefulGain* in diesem Bereich liegen. Deshalb ergeben die Nenner der Brüche jeweils negative Werte. Ein negatives Ergebnis für $MinPos$ bedeutet jedoch, dass an dieser Stelle niemals geprunt würde, denn es können nicht weniger als 0 Tupel abgedeckt werden. Daher sollte der Wert für *MinUsefulGain* bei beiden Heuristiken so gewählt werden, dass auch der Zähler immer negativ ist und sich dadurch insgesamt ein positiver Wert ergibt. Daraus ergibt sich folgende Bedingung für das m-estimate:

$$\frac{N^+}{N^+ + N^-} < MinUsefulGain < 1$$

Ähnliches gilt für Laplace:

$$\frac{1}{2} < MinUsefulGain < 1$$

Auch die Berechnung von $MaxPossibleGain$ musste angepasst werden. Dieser Werte entspricht der Bewertung einer neuen Regel, die alle verbliebenen positiven Tupel n^{+*} und keine negativen abdecken würde. Hier gilt für das m-estimate:

$$MaxPossibleGain = \frac{n^+ + m \cdot \frac{N^+}{N^+ + N^-}}{n^+ + n^{-*} + m} = \frac{n^{+*} + m \cdot \frac{N^+}{N^+ + N^-}}{n^{+*} + m}$$

Und für Laplace:

$$MaxPossibleGain = \frac{n^{+*} + m \cdot \frac{N^+}{N^+ + N^-}}{n^{+*} + m} = \frac{n^{+*} + 2 \cdot \frac{N^+}{N^+ + N^+}}{n^{+*} + 2} = \frac{n^{+*} + 2 \cdot \frac{N^+}{2 \cdot N^+}}{n^{+*} + 2} = \frac{n^{+*} + 1}{n^{+*} + 2}$$

3.2.3 Ergänzung des Prunings

Sowohl das m-estimate als auch Laplace berücksichtigen die bestehende partielle Regel nicht, sondern bewerten diese nur zusammen mit dem zu ergänzenden Literal. Dadurch kann es in seltenen Fällen passieren, dass eine Regel mehrfach durch das gleiche Literal ergänzt wird. Dies kann dann passieren, wenn keine mögliche Ergänzung eine höhere Bewertung als die partielle Regel bekommt. Bei Verwendung des Weighted Information Gain tritt dieses Problem nicht auf, da eine solche Ergänzung dort kein Gain erhält und daher ignoriert wird.

Um diesen Effekt zu verhindern, wurde daher das Pruning erweitert. Der Algorithmus ignoriert durch die Erweiterung der Suche alle Literale, die bereits in der partiellen Regel vorhanden sind. Gleiches gilt für deren Negat, da in diesem Fall kein Tupel die entstehende Regel erfüllen könnte. Diese Prüfung erfordert, dass das neue Literal mit allen bereits in der Regel vorhandenen verglichen wird. Daher wird diese Art des Prunings nur in Verbindung mit den beiden neuen Heuristiken verwendet, um den entsprechenden Aufwand bei Verwendung des Weighted Information Gains einzusparen.

4 Ergebnisse

4.1 Testmethode

Um die verschiedenen Heuristiken vergleichen zu können, wurde das modifizierte Programm mit unterschiedlichen Testdatensätzen ausgeführt und die Ergebnisse aufgezeichnet. Dabei handelte es sich zunächst um die sechs Datensätze, die bei der FOIL-Implementierung mitgeliefert wurden: *ackermann*, *crx*, *member*, *ncm*, *qs44* und *sort* [FOIL]. Zusätzlich wurden acht propositionale Datensätze aus dem UCI-Repository [AN], *hill-valley*, *hill-valley-noise*, *house-votes-84*, *krkopt*, *monks-2*, *monks-3*, *page-blocks* und *sonar* sowie sieben relationale Datensätze *mesh-a*, *mesh-b*, *mesh-c*, *mesh-d*, *mesh-e*, *pyrimidines* [ILPAD] und *uwcse* [ACE] verwendet. Relationale Datensätze verfügen im Gegensatz zu propositionalen über angefügtes Hintergrundwissen, wodurch sich der Suchraum für mögliche Literale vergrößert.

Da in dieser Arbeit die Unterschiede zwischen den Heuristiken untersucht werden sollen, wurden keine Datensätze verwendet, bei denen in ersten Tests mit allen Heuristiken eine perfekte Abdeckung erzielt wurde oder keine Regeln gelernt werden konnten.

Bei den Testläufen wurde neben den Heuristiken auch der Parameter des m-estimates variiert. Dabei wurden für m die Werte 0, 0.1, 0.5, 1, 2, 3, 4, 8 und 16 [LD94] sowie 22.466 [JF08] verwendet. Die Gain-Variante dieser Heuristik wurde mit den gleichen Werten für m getestet. Um eine bessere Vergleichbarkeit zwischen den Datensätzen zu gewährleisten, wurde außer dem „Verbosity Level“ der Ausgabe kein weiterer Parameter für FOIL gesetzt.

Zusätzlich wurden drei weitere relationale Datensätze, *bongard4* [ACE], *proteins* und *satellite* [ILPAD], gesondert behandelt. Da deren mittlere Laufzeit, im Gegensatz zu den vorher erwähnten Datensätzen, nicht im Bereich von Sekunden, sondern zwischen 30 und 160 Minuten lag, waren keine derart ausführlichen Tests möglich. Es wurde daher mit Hilfe der bereits genannten Datensätze zunächst der beste Parameter m = 16 für beide Varianten des m-estimates bestimmt (Abschnitt 4.2.2). Die Tests mit den drei zusätzlichen Datensätzen wurden dann ausschließlich mit diesem Wert für m durchgeführt.

Jeder Datensatz wurde in zehn *Folds* aufgeteilt. Dazu wurden jeweils 10 % der Tupel aus den Trainingsdaten entfernt und zur Validierung der gelernten Regeln verwendet. Diese Validierungsdatensätze sind disjunkt, es wird also jedes Tupel in genau einem Fold zur Validierung verwendet. Bei der Auswertung wird dann der Mittelwert der Messdaten über allen Folds eines Datensatzes gebildet. Durch dieses Vorgehen, die *Cross Validation*, werden zu genaue Anpassungen der Regeln an die Trainingsdaten erkannt.

Dies setzt voraus, dass negative Beispiele explizit angegeben werden. Ist dies nicht der Fall, generiert FOIL diese aus der Closed-World-Assumption, wodurch die für den jeweiligen Fold entfernten positiven Beispiele von dem Programm als negative Beispiele verwendet werden. Bei der anschließenden Validierung werden diese dann entsprechend falsch klassifiziert. Um diesen Nebeneffekt zu verhindern wurden die Datensätze *ackermann*, *qs44* und *sort* um alle in der jeweiligen Welt möglichen negativen Beispiele ergänzt.

Am Ende jedes Laufes wird zunächst die Genauigkeit (Accuracy) der gelernten Regeln bewertet. Diese basiert auf den Trainingsdaten und berechnet sich wie folgt (n^+ und n^- entspricht den abgedeckten positiven und negativen Beispielen, N^+ und N^- der jeweiligen Gesamtzahl):

$$Accuracy = \frac{(n^+ + (N^- - n^-))}{(N^+ + N^-)}$$

Der Wert der Genauigkeit ist am höchsten, wenn alle positiven und keine negativen Beispiele abgedeckt werden.

Da sich dieser Wert nur auf die Trainingsdaten bezieht und daher keine Aussage über Overfitting trifft, wird außerdem die Genauigkeit auf den Validierungsdatensätzen nach der gleichen Formel berechnet.

Als weiteres Merkmal wird untersucht, wie sich die Wahl der Heuristik auf die Größe der gelernten Theorie auswirkt. Dazu wird sowohl die Anzahl der gelernten Regeln als auch die Anzahl der Literale, aus denen diese bestehen, gezählt. Dabei werden die Literale in Kopf und Körper der Regeln berücksichtigt. Treten unterschiedlich große, aber im Bezug auf

die Genauigkeiten gleichwertige Theorien auf, ist die weniger komplexe zu bevorzugen.

Außerdem wird die Laufzeit des Programms auf einem Rechner mit zwei Opteron 250-Prozessoren von AMD gemessen, um einen Hinweis auf die Effizienz der unterschiedlichen Heuristiken zu erhalten. Die Prozessoren sind mit 2,33 GHz getaktet. FOIL nutzt allerdings während seiner Berechnungen immer nur einen der zwei Prozessoren. Da die Messung der Laufzeit kleinere Ungenauigkeiten, beispielsweise durch parallel laufende Systemprozesse, aufweisen kann, wird hier der Mittelwert aus drei Läufen gebildet.

4.2 Auswertung

Die Auswertung gliedert sich in mehrere Teile. Zunächst werden die Heuristiken untereinander verglichen, wobei für das m-estimate jeweils nur die Parameterwahl betrachtet wird, die auf den jeweiligen Validierungsdatensätzen die höchste Genauigkeit aufgewiesen hat. Es kann daher innerhalb eines Datensatzes passieren, dass für die verschiedenen Varianten des m-estimates unterschiedliche Werte für m angenommen werden. Danach werden die drei zusätzlichen relationalen Datensätze einzeln betrachtet, da hier nur ein Wert des Parameters getestet wurde und es daher nicht möglich ist, auf die gleiche Art und Weise das beste Ergebnis auszuwählen. Somit sind die Ergebnisse auch nur eingeschränkt vergleichbar.

Anschließend werden die Auswirkungen des Parameters genauer aufgeschlüsselt, indem die verschiedenen Werte für m noch einmal einzeln analysiert werden. Abschließend folgt eine Bewertung der Heuristiken auf Basis dieser Ergebnisse.

4.2.1 Vergleich der Heuristiken

Zuerst wird die Genauigkeit der gelernten Regeln auf den Trainings- und Validierungsdatensätzen betrachtet. Danach werden die Größe der gelernten Theorie und bei ausgewählten Datensätzen der Zeitaufwand eines Programmlaufs mit den unterschiedlichen Heuristiken verglichen. Abschließend findet eine gesonderte Betrachtung von drei Datensätze mit besonders langen Laufzeiten statt, ebenfalls aufgeteilt in Untersuchungen zu Genauigkeit, Theoriegröße und Laufzeit. Eine zusätzliche Darstellung der Ergebnisse in Tabellenform findet sich im Anhang A.3.

Genauigkeit

Für den folgenden Vergleich wurden die Datensätze entsprechend Abschnitt 4.1 in drei Gruppen eingeteilt. Um die Übersichtlichkeit zu gewährleisten wird nur die, in dieser Arbeit als wichtigstes Unterscheidungsmerkmal betrachtete, Genauigkeit auf den Testdaten grafisch dargestellt.

Bei den der Referenzimplementierung beiliegenden Datensätzen (Abbildung 4.1) *ackermann*, *member*, *qs44* und *sort* erreichen alle verwendeten Heuristiken nahezu 100 % Genauigkeit auf den Trainings- und Testdaten. Mit *ncm* beträgt dieser Wert auf den Trainingsdaten ebenfalls fast 100 %, allerdings tritt hier eine geringere Genauigkeit auf den Testdaten auf: bei Laplace und der Absolut-Variante des m-estimates 99 %, bei der Gain-Variante und dem Weighted Information Gain 98 %. Dies entspricht aufgrund der geringen Größe des Datensatzes allerdings nur maximal einem falsch klassifizierten Tupel pro Fold.

Ein schlechteres Ergebnis wird von allen Heuristiken beim Datensatz *crx* erzielt, wo erstmals Overfitting auftritt. Laplace kommt hier bei den Trainingsdaten auf 98 % Genauigkeit, die Absolut-Variante auf 96 % und die Gain-Variante auf 99 %, bei einer Genauigkeit auf den Validierungsdatensätzen von jeweils 82 %. Das Weighted Information Gain weist eine niedrigere Genauigkeit von 81 % mit den Testdaten auf, bei einer Genauigkeit von 99 % auf den Trainingsdaten.

Wenn man die Genauigkeit auf den Validierungsdatensätzen als primäres Unterscheidungsmerkmal nimmt, zeigt sich also bei diese Datensätzen kein großer Unterschied zwischen den einzelnen Heuristiken (maximal 1 %). Auch auf den Trainingsdaten ist sie meist identisch und unterscheidet sich um maximal 3 %.

Betrachtet man die propositionalen Datensätze (Abbildung 4.2), zeigt sich zunächst ein ähnliches Bild. Alle Heuristiken erreichen mit *hill-valley* eine Genauigkeit von nahezu 100 % auf den Trainingsdaten. Laplace hat dabei eine Genauigkeit von 99 % auf den Testdaten, die anderen Heuristiken circa 100 %. Bei *krkopt* zeigen sich keine Unterschiede, hier wird von allen Heuristiken etwa 100 % Genauigkeit auf den Trainings- und 99 % auf den Validierungsdatensätzen erreicht.

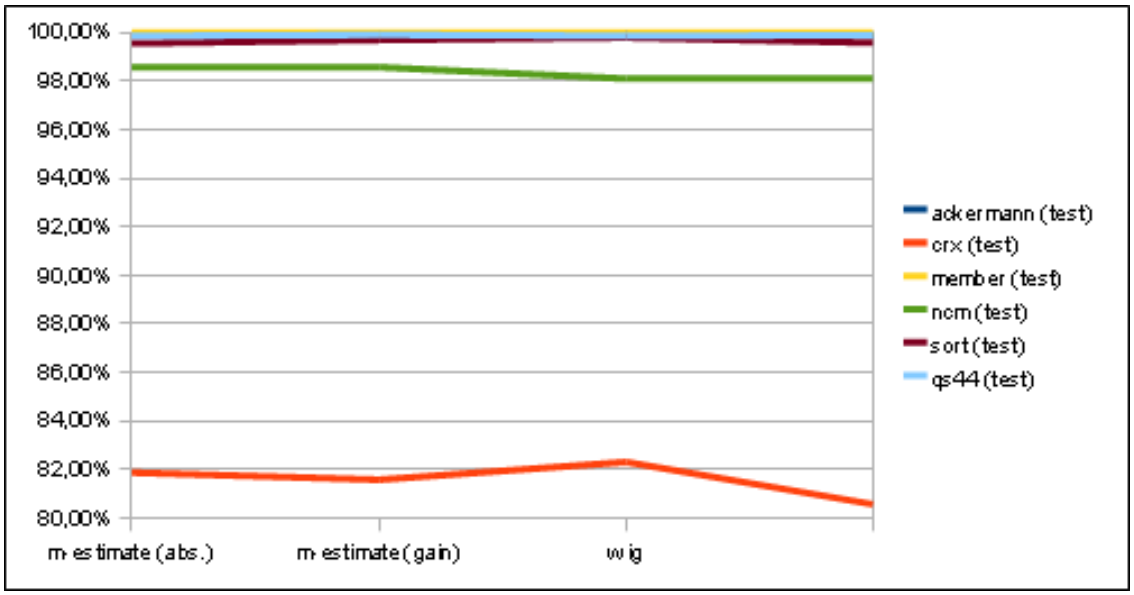

Abbildung 4.1.: Genauigkeit auf den Validierungsdaten (mitgelieferte Datensätze)

Etwas schlechtere, aber untereinander ähnliche Ergebnisse werden bei dem Datensatz *monks-3* erzielt. Laplace erreicht hier auf den Trainingsdaten 98 % und auf den Testdaten 97 % Genauigkeit, die übrigen Heuristiken 99 % und 98 %. Gleiches gilt für *house-votes-84* (Laplace 99 % / 93 %, übrige 99 % / 94 %) und *page-blocks* (Laplace und Absolut-Variante 97 % / 95 %, Gain-Variante 99 % / 96 %, Weighted Information Gain 99 % / 97 %).

Große Unterschiede zwischen den Heuristiken treten zum ersten Mal bei *monks-2* zutage. Hier erreichen zwar alle Heuristiken eine Genauigkeit von ungefähr 100 % auf den Trainingsdaten, bei Laplace und der Absolut-Variante des m-estimates beträgt diese allerdings nur 89 % auf den Testdaten. Dies deutet darauf hin, dass an dieser Stelle Overfitting auftritt. Bei der Gain-Variante und dem Weighted Information Gain ist dieser Effekt nicht zu beobachten, hier beträgt der Wert 98 % beziehungsweise 99 %.

Auch bei *sonar* unterscheiden sich die Ergebnisse stark, obwohl bei allen Heuristiken Overfitting zu beobachten ist. Laplace kommt dabei auf 96 % Genauigkeit auf den Trainingsdaten und 61 % auf den Testdaten. Die beiden Varianten des m-estimates weisen annähernd gleiche Werte von 98 % / 72 % und 100 % / 70 % auf, wobei auch hier die Gain-Variante vorne liegt. Ein noch besseres Ergebnis erreicht das Weighted Information Gain mit 100 % / 76 %.

Der deutlichste Unterschied tritt bei *hill-valley-noise* auf. Laplace und die Absolut-Variante kommen hier auf Werte von 74 % / 54 % und 77 % / 59 %. Merklich besser sind die Gain-Variante und das Weighted Information Gain, die beide auf 75 % Genauigkeit auf den Validierungsdatensätzen bei 97 % und 100 % auf den Trainingsdaten kommen. Hierbei ist zu bemerken, dass sich nicht nur das Overfitting verringert, was man an der höheren Genauigkeit auf den Testdaten sehen kann. Zusätzlich ist auch die Qualität der gelernten Regeln höher, ausgedrückt durch eine höhere Genauigkeit auf den Trainingsdaten. Im Vergleich mit dem Datensatz *hill-valley* ist zudem der negative Einfluss von Noise auf die Heuristiken sichtbar.

Bei den propositionalen Datensätzen zeigt sich also insgesamt ein gemischtes Bild. Während bei fünf Datensätzen ebenfalls kaum Unterschiede zwischen den Heuristiken deutlich werden, treten bei den übrigen drei deutliche Differenzen bei den Genauigkeiten auf. Dabei zeigt sich, dass das Weighted Information Gain und die Gain-Variante des m-estimates den beiden anderen Heuristiken bei diesen Datensätzen überlegen sind. Dieser Unterschied fällt mit bis zu 21 % auf den Testdaten und 26 % auf den Trainingsdaten teilweise recht deutlich aus.

Auch bei den relationalen Datensätzen (Abbildung 4.3) gibt es einen Datensatz, bei dem sich die Heuristiken ähnlich verhalten. Mit *uwcse* erreicht die Gain-Variante 97 % Genauigkeit auf den Trainings- und 96 % auf den Testdaten, bei den anderen Heuristiken betragen beide Werte 96 %.

Ein etwas größerer Unterschied tritt bei *pyrimidines* auf, bei Laplace und der Absolut-Variante beträgt die Genauig-

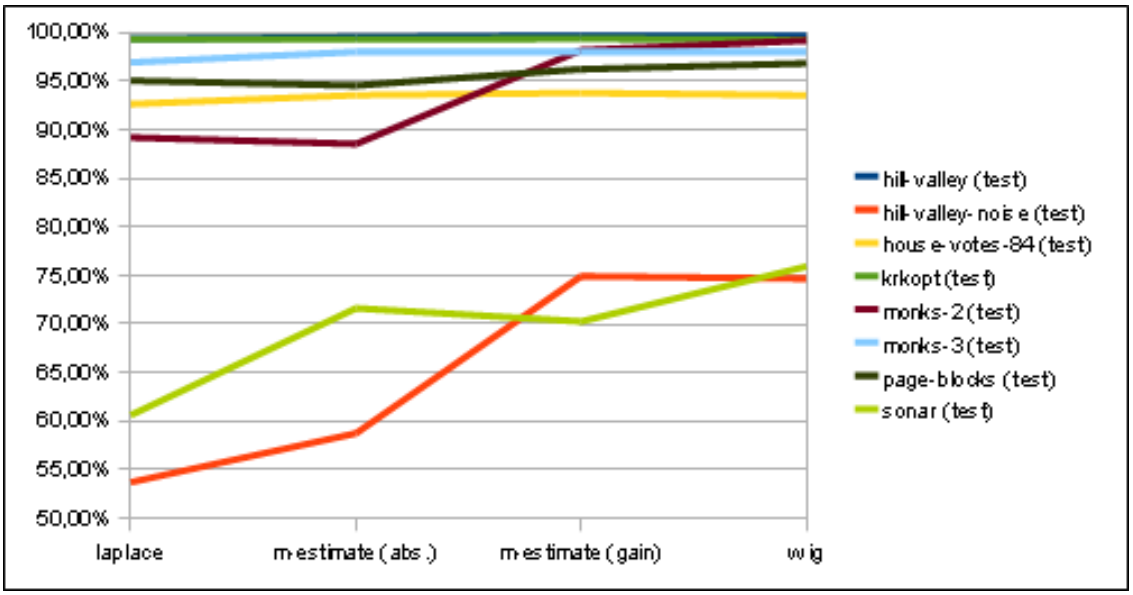

Abbildung 4.2.: Genauigkeit auf den Validierungsdaten (propositionale Datensätze)

keit auf den Validierungsdatensätzen 88 %, bei der Gain-Variante 90 % und beim Weighted Information Gain 92 %. Die Genauigkeit auf den Trainingsdaten beträgt bei allen Heuristiken ungefähr 100 %.

Die mit den Datensätzen *mesh-a, mesh-b, mesh-c, mesh-d* und *mesh-e* erzielten Ergebnisse unterscheiden sich zwischen den Datensätzen kaum. Laplace, die Absolut-Variante und das Weighted Information Gain erzielen hier eine Genauigkeit von fast 100 % auf den Trainingsdaten bei Werten von 83 % - 84 %, 84 % - 85 % und 92 % - 93 % auf den Testdaten. Die Gain-Variante hat zwar einen etwas geringeren Wert auf den Trainingsdaten von 97 % - 98 %, liefert mit 90 % - 91 % aber das zweitbeste Ergebnis auf den Validierungsdatensätzen.

Zusammenfassend lässt sich daher sagen, dass die relationalen Datensätze die bisher erzielten Ergebnisse bestätigen. Auch hier sind die Genauigkeit der Gain-Variante und des Weighted Information Gains etwas besser als die der beiden anderen Heuristiken. Der Unterschied fällt mit maximal 7 % bei den Test- und 2 % bei den Trainingsdaten aber nicht so stark wie bei einem Teil der propositionalen Datensätze aus.

Für die Genauigkeit auf den Validierungsdatensätzen, die in dieser Arbeit als primäres Unterscheidungsmerkmal dient, wurde zudem der durchschnittliche Rang der Heuristiken über alle Datensätze berechnet (Abbildung 4.4). Hier liegen das Weighted Information Gain und die Gain-Variante gleichauf bei 1.86. Es folgen die Absolut-Variante mit 2.48 und Laplace mit 3.24.

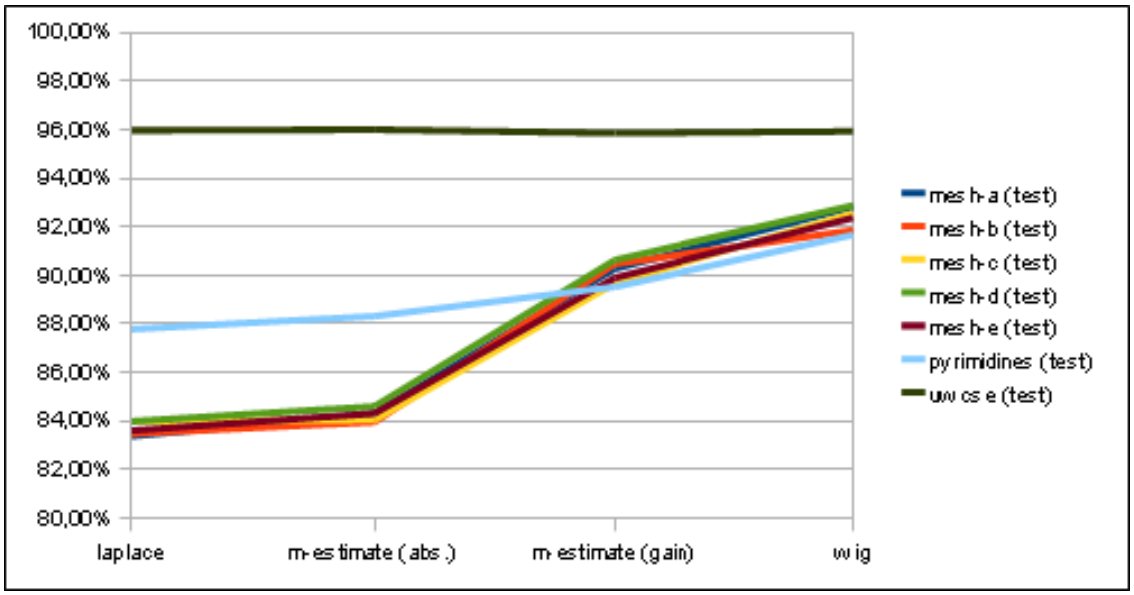

Abbildung 4.3.: Genauigkeit auf den Validierungsdaten (relationale Datensätze)

Abbildung 4.4.: Durchschnittsrang (Validierungsdaten)

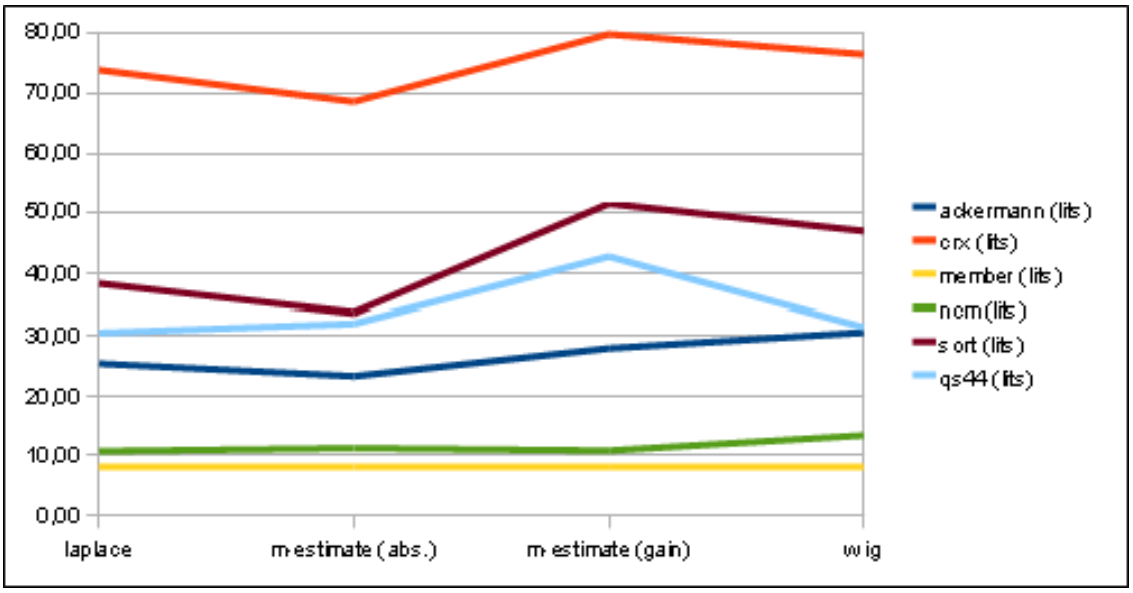

Abbildung 4.5.: Literalanzahl der Theorie (mitgelieferte Datensätze)

Theoriegröße

Die Datensätze werden zur Untersuchung der Theoriegröße ebenfalls in Gruppen eingeteilt. Es werden erneut nur die Durchläufe des m-estimates mit der höchsten Genauigkeit auf den Testdaten betrachtet. Dabei wird nur die Anzahl der erzeugten Literale bebildert.

Mit den der Referenzimplementierung beiliegenden Datensätzen (Abbildung 4.5) produziert der Algorithmus relativ kleine Theorien. Bei *member* besteht diese mit allen Heuristiken aus 3 Regeln und 8 Literalen.

ncm wird dem Weighted Information Gain mit 3 Regeln und 13 Literalen erklärt, die übrigen Heuristiken benötigen 11 Literale bei der gleichen Anzahl von Regeln.

Mit *ackermann* ähnelt sich hingegen die Anzahl der Regeln, die Anzahl der verwendeten Literalen unterscheidet sich etwas stärker. Alle Heuristiken erzeugen ungefähr 6 Regeln mit 23 bis 31 Literalen. Bei beiden Werten erreicht die Absolut-Variante des m-estimates den geringsten Wert, das Weighted Information Gain den höchsten.

Auch bei *sort* wird eine ähnliche Anzahl von Regeln, 5 bis 6, erzeugt. Dabei verwendet die Gain-Variante 43 Literale, alle anderen Heuristiken zwischen 30 und 32.

Eine größere Varianz der Werte zeigt sich bei *qs44*. Auch wenn die Anzahl der Regeln mit 6 bis 8 dicht beisammen liegt, benötigen die Absolut-Variante (34) und Laplace (38) deutlich weniger Literale als das Weighted Information Gain (47) und die Gain-Variante (51).

Bei *crx* tritt der größere Unterschied bei der Anzahl der erzeugten Regeln auf. Mit dem Weighted Information Gain liegt diese bei 25, mit den übrigen Heuristiken zwischen 35 und 40. Die Regeln bestehen dabei aus 69 bis 80 Literalen.

Zusammenfassend lässt sich sagen, dass bei diesen Datensätzen nur verhältnismäßig geringe Unterschiede bei der Anzahl der erzeugten Regeln auftreten. Die Gain-Variante und das Weighted Information Gain verwenden dabei in den meisten Fällen mehr Literale, dieser Unterschied beträgt allerdings bei 4 von 6 Datensätzen weniger als 10. Die einzelnen Regeln fallen mit durchschnittlich 3 bis 6 Literalen relativ kurz aus.

Betrachtet man die propositionalen Datensätze (Abbildung 4.6), zeigen sich zunächst nur kleine Unterschiede in der Theoriegröße. *hill-valley* wird mit 2 bis 4 Regeln und 4 bis 7 Literalen erklärt.

Gleiches gilt für *monks-3* (9 Regeln, 12 bis 14 Literale) und *house-votes-84* (9 bis 11 Regeln, 10 bis 11 Literale). Bei letztgenanntem Datensatz fällt dabei auf, dass die meisten Regeln keinen Körper besitzen.

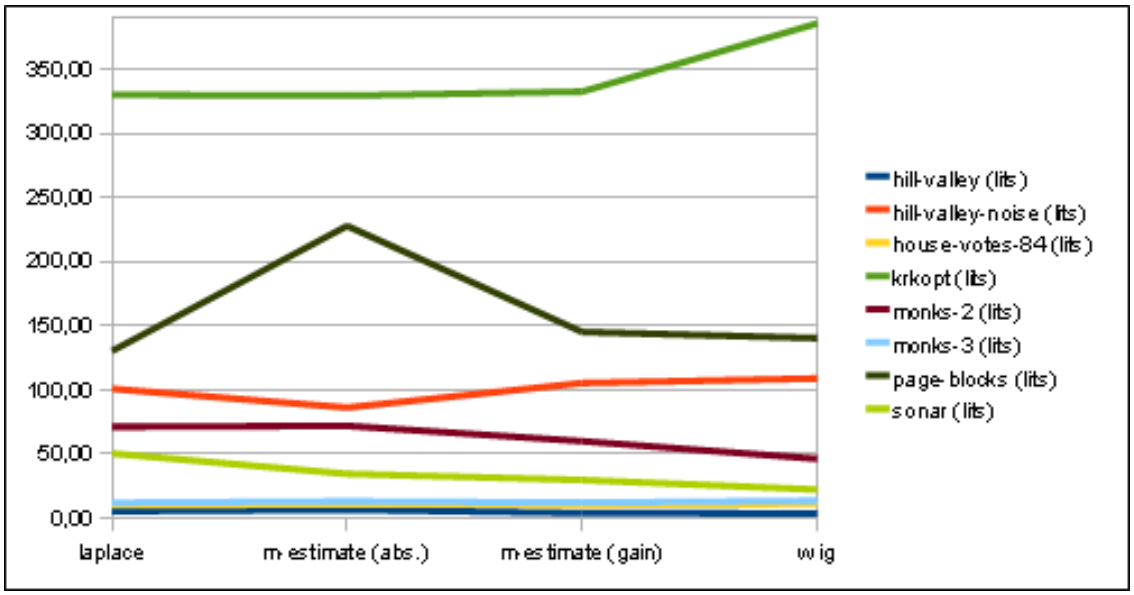

Abbildung 4.6.: Literalanzahl der Theorie (propositionale Datensätze)

Ein deutlicher Unterschied bei der Anzahl der Regeln zeigt sich bei *hill-valley-noise*. Laplace erreicht hier einen Wert von 81, die Absolut-Variante 65, die Gain-Variante 37 und das Weighted Information Gain 20. Im Gegensatz dazu unterscheidet sich die Anzahl der Literale weniger stark, sie liegt mit der Absolut-Variante bei 86 und mit den anderen Heuristiken zwischen 101 und 109. Bemerkenswert ist dabei, dass Laplace über viermal so viele Regeln wie das Weighted Information Gain erzeugt, diese aber deutlich kürzer sind und daher insgesamt weniger Literale benötigt werden.

Bei *monks-2* resultiert dagegen eine geringere Anzahl von Regeln in weniger Literalen. Laplace und die Absolut-Variante erzeugen 41 und 43 Regeln bei 71 und 72 Literalen, die Gain-Variante 29 Regeln bei 60 Literalen und das Weighted Information Gain 21 Regeln bei 46 Literalen.

Das gleiche Verhalten lässt sich auch bei *sonar* feststellen. Die mit Laplace erzeugte Theorie besteht aus 29 Regeln (51 Literale), mit der Absolut-Variante aus 18 (35), mit der Gain-Variante 11 (30) und mit dem Weighted Information Gain 6 (23).

Die Absolut-Variante liefert mit 73 Regeln aus 228 Literalen bei *page-blocks* die größte Theorie. Die übrigen Heuristiken benötigen 28 bis 40 Regeln und 130 bis 145 Literale, wobei Laplace die wenigsten Regeln und das Weighted Information Gain die wenigsten Literale erzeugt.

Für *krkopt* werden von allen Heuristiken sehr komplexe Theorien erzeugt. Die mit dem Weighted Information Gain konstruierte Theorie besitzt mit 242 die wenigsten Regeln und gleichzeitig mit 386 die meisten Literale. Alle anderen Heuristiken liegen mit 290 bis 294 Regeln und 330 bis 333 Literalen nahezu gleichauf.

Der bei den mitgelieferten Datensätzen beobachtete Effekt kehrt sich bei den propositionalen um, hier resultieren aus Laplace und der Absolut-Variante zumeist größere oder annähernd gleich große Theorien. Dies betrifft in den meisten Fällen sowohl die Anzahl der Regeln als auch die der Literale. Im Durchschnitt bestehen die Regeln aus nur 1 - 2 Literalen, alleine für *page-blocks* beträgt dieses Verhältnis 1 zu 4.

Auch bei den relationalen Datensätzen (Abbildung 4.7) zeigt sich kein einheitliches Bild. Für *uwcse* konstruiert die Gain-Variante mit 7 Regeln und 39 Literalen die größte Theorie. Die Absolut-Variante (5 / 26) und das Weighted Information Gain (5 / 29) erzielen untereinander ähnliche Ergebnisse, Laplace benötigt nur 4 Regeln mit 16 Literalen.

Im Gegensatz dazu erzeugt es bei *pyrimidines* die meisten Regeln (408) und Literale (598). Aus der Absolut-Variante resultieren 309 Regeln und 545 Literale, aus der Gain-Variante 241 und 545 sowie aus dem Weighted Information Gain 171 und 459. Es ist festzustellen, dass der Unterschied bei der Anzahl der Regeln mit ungefähr 230 stärker ausfällt als bei den Literalen mit 140.

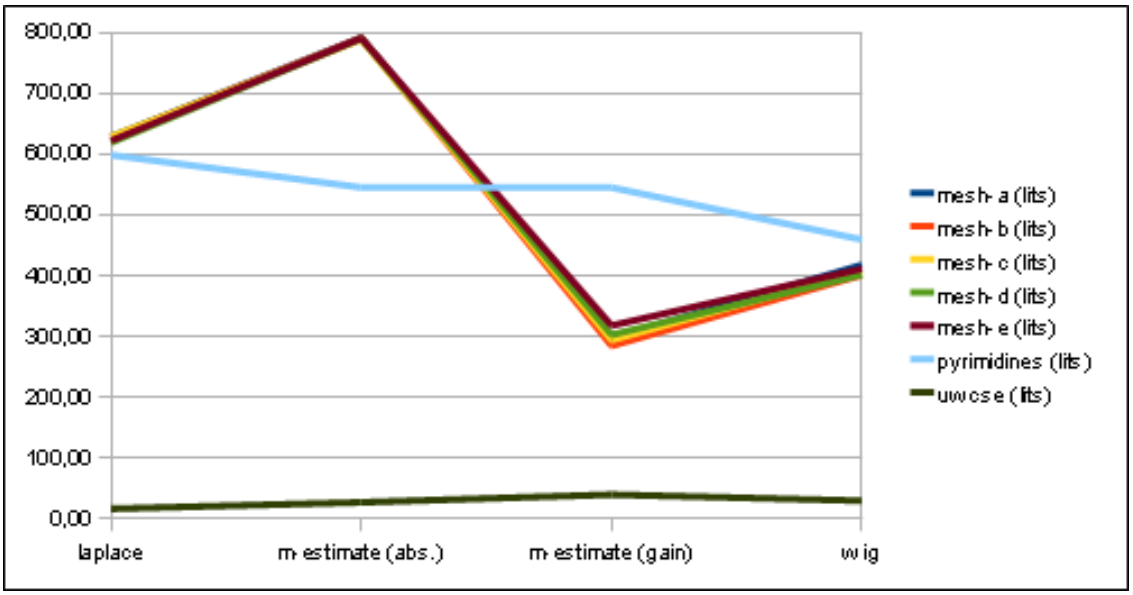

Abbildung 4.7.: Literalanzahl der Theorie (relationale Datensätze)

Die Datensätze *mesh-a*, *mesh-b*, *mesh-c*, *mesh-d* und *mesh-e* verhalten sich auch bei der Theoriegröße fast gleich. Mit der Absolut-Variante wird die größte Theorie konstruiert, sie besteht aus etwa 410 Regeln und 790 Literalen. Es folgen Laplace mit 310 Regeln, zusammengesetzt aus 620 Literalen, und das Weighted Information Gain mit 80 Regeln aus 410 Literalen. Die insgesamt kleinste Theorie liefert die Gain-Variante mit etwa 90 Regeln, aber nur 300 Literalen.

Bei dieser Gruppe bestätigt sich der Eindruck aus den propositionalen Datensätzen, dass die Gain-Variante und das Weighted Information Gain tendenziell eher kleinere Theorien, sowohl im Bezug auf die Regeln als auch auf die Literale, erzeugen. Eine Regel besteht hier im Durchschnitt aus 2 bis 4 Literalen.

Zeitaufwand

Damit der Zeitaufwand der Heuristiken (Abbildung 4.8) sinnvoll ausgewertet werden kann, wurden nur Datensätze verwendet, bei denen die Laufzeit von mindestens einer Heuristik mehr als 5 Sekunden beträgt. Ansonsten ist der Unterschied nicht signifikant genug, um eine Aussage über unterschiedliche Laufzeiten treffen zu können. An dieser Stelle wurden analog zum vorherigen Abschnitt wieder nur die Parameter mit der höchsten Genauigkeit auf den Testdaten berücksichtigt. Zu beachten ist zudem, dass mit fast allen hier betrachteten Datensätzen bei Untersuchung der Genauigkeiten keine großen Unterschiede auftraten, sodass die unterschiedlichen Laufzeiten zu qualitativ gleichwertigen Ergebnissen geführt haben. Die einzige Ausnahme bildet *hill-valley-noise*, wo die Gain-Variante des m-estimates und das Weighted Information Gain bessere Ergebnisse als die anderen Heuristiken geliefert haben.

Bereits bei *hill-valley-noise* zeigt sich ein deutlicher Unterschied zwischen den Laufzeiten. Laplace benötigt hier mit 35 s am längsten, während die Absolut-Variante 25 s, die Gain-Variante 17 s und das Weighted Information Gain 13 s benötigen. Bei *page-blocks* dauert ein Lauf der Absolut-Variante dagegen 10 s, bei den übrigen Heuristiken nur 1 s. Und bei *qs44* ist wiederum die Gain-Variante mit 75 s deutlich unterlegen, Laplace und die Absolut-Variante brauchen nur 13 s und das Weighted Information Gain 21 s.

Im Gegensatz dazu läuft das Weighted Information Gain bei den Datensätzen *mesh-a*, *mesh-b*, *mesh-c*, *mesh-d*, *krkopt*, *pyrimidines* und *uwcse* am längsten. Die mesh-Datensätze verhalten sich dabei annähernd gleich. Das Weighted Information Gain braucht über 5 s für einen Durchlauf, die Absolut-Variante etwas weniger. Mit der Gain-Variante dauert ein Lauf etwas mehr als 3 s und Laplace liegt meist knapp unter diesem Wert. Eine Ausnahme bildet an dieser Stelle *mesh-c*, wo Laplace etwas mehr Zeit als die Gain-Variante in Anspruch nimmt.

Bei *krkopt* benötigt es 48 s, die Absolut-Variante 22 s und die beiden übrigen Heuristiken 21 s. 57 s gegenüber 21 s (Laplace), 9 s (Absolut-Variante) und 10 s (Gain-Variante) sind es beim Datensatz *pyrimidines*. Und auch bei *uwcse* zeigt

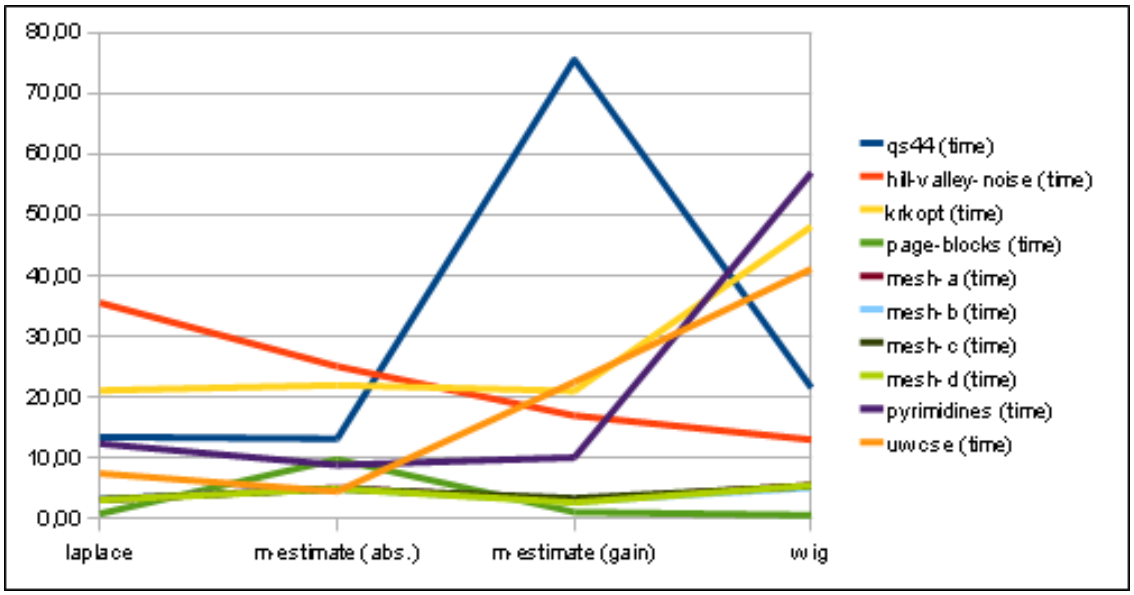

Abbildung 4.8.: Zeitaufwand

sich ein deutlicher Unterschied, der Algorithmus läuft mit dem Weighted Information Gain 41 s, mit Laplace 7 s, mit der Absolut-Variante 4 s und mit der Gain-Variante 22 s.

Aus diesen Laufzeiten lässt sich kein klarer Trend ableiten. Je nach Datensatz benötigt eine andere Heuristik teilweise deutlich mehr Zeit. Und obwohl ein Durchlauf mit dem Weighted Information Gain in 7 von 10 Fällen am längsten dauert, beträgt dieser Unterschied bei 4 Datensätzen weniger als 1 s. Auch in der Reihenfolge der Laufzeiten lassen sich keine Regelmäßigkeiten erkennen. Es ist daher davon auszugehen, dass die Dauer eines Durchlaufs mit den jeweiligen Heuristiken sehr stark problemspezifisch ist.

Ein möglicher Grund für die längere Laufzeit einer Heuristik kann dabei einerseits darin bestehen, dass diese Heuristik komplexere Regeln lernt. Deutlich sichtbar war dies jedoch nur bei den Datensätzen *page-blocks* und *pyrimidines*. Andererseits ist es möglich, dass mit der Heuristik viele Ergänzungen einer Regel untersucht werden, die später verworfen werden. Dies geht allerdings nicht aus dem Endergebnis hervor und wurde daher nicht untersucht.

Zusätzliche Datensätze

Betrachtet man die Genauigkeiten, die von den Heuristiken auf den drei zusätzlichen Datensätzen (Abbildung 4.9) erreicht werden, zeigen sich leichte Unterschiede. Auf *satellite* erzielt Laplace mit 97 % auf den Trainings- beziehungsweise 96 % auf den Testdaten das schlechteste Ergebnis, das m-estimate folgt mit 100 % / 97 % (Absolut-Variante) und 99 % / 99 % (Gain-Variante). Mit dem Weighted Information Gain wird ein Ergebnis von ungefähr 100 % / 100 % erreicht.

Bei *bongard4* liefert dagegen die Gain-Variante mit 92 % / 90 % das schlechteste Ergebnis, gefolgt von Laplace mit 96 % / 93 % und dem Weighted Information Gain mit 97 % / 94 %. Die Absolut-Variante liegt hier mit 98 % auf den Trainings-, bei 95 % auf den Validierungsdatensätzen vorne.

Mit *proteins* produzieren alle Heuristiken eine relativ geringe Genauigkeit auf den Testdaten. Laplace schneidet mit 65 % am besten ab, gefolgt von der Absolut-Variante mit 64 %. Mit der Gain-Variante und dem Weighted Information Gain werden jeweils 62 % erreicht. Stärker unterscheidet sich die Genauigkeit auf den Trainingsdaten, hier ist die Gain-Variante mit 78 % merklich schlechter als die übrigen Heuristiken (Absolut-Variante 90 %, Laplace 95 %, Weighted Information Gain 99 %).

Untersucht man die Theoriegröße (Abbildung 4.10), zeigen sich bei diesen Datensätzen keine Besonderheiten. Mit *bongard4* liefern alle Heuristiken ähnliche Werte von 15 bis 18 Regeln und 156 bis 168 Literalen.

Für *satellite* unterscheiden sich die Theoriegrößen stärker. Aus der Absolut-Variante (7 / 23) und Laplace (6 / 19)

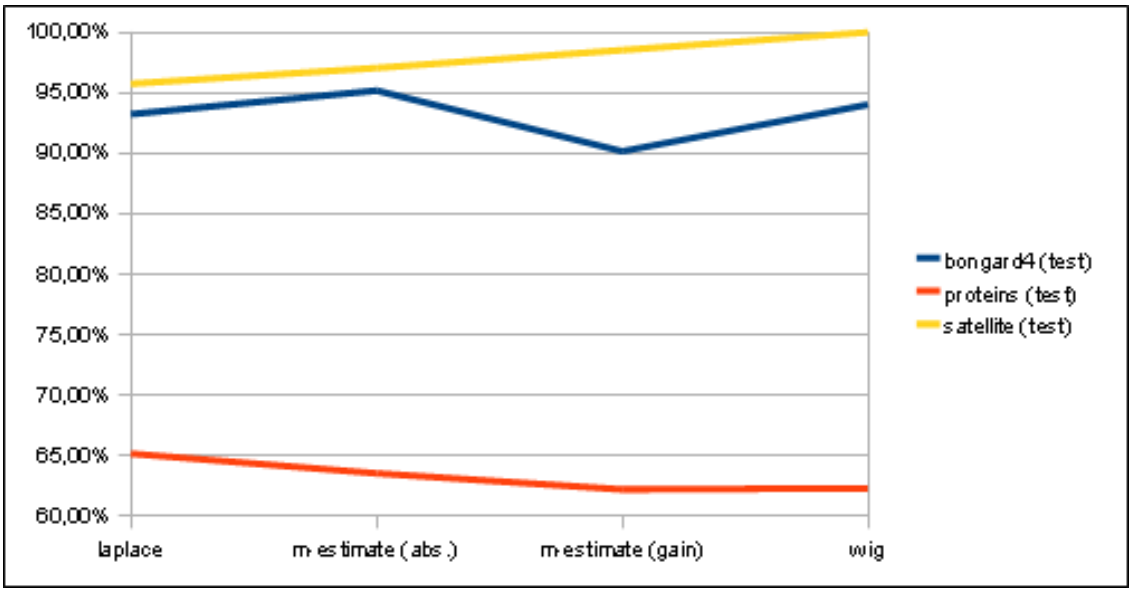

Abbildung 4.9.: Genauigkeit auf den Validierungsdaten (zusätzliche Datensätze)

entstehen die größten Theorien, es folgt das Weighted Information Gain (4 / 16). Das Ergebnis der Gain-Variante besteht nur aus insgesamt 6 Literalen, aufgeteilt auf 3 Regeln.

Mit *proteins* erzeugen alle Heuristiken deutlich größere Theorien. Laplace benötigt 181 Regeln, die Absolut-Variante 97, das Weighted Information Gain 64 und die Gain-Variante 38. Abgesehen von dem Weighted Information Gain (868) verhält sich die Anzahl der Literale proportional zur fallenden Regelzahl (Laplace 847, Absolut-Variante 600, Gain-Variante 272).

Bei diesen Datensätzen fällt auf, dass sich die längere Dauer eines Durchlaufes nicht allgemein durch eine größere Theorie begründen lässt. Auch die einzelnen Regeln sind mit durchschnittlich 3 bis 8 Literalen nur geringfügig länger als bei den zuvor betrachteten Datensätzen.

Beim Zeitaufwand (Abbildung 4.11) zeigen sich noch deutlichere Unterschiede. Für *bongard4* benötigen Laplace und die Gain-Variante etwa 20 Min, die beiden anderen Heuristiken 40 Min. Bei *proteins* braucht die Gain-Variante 75 Min, die Absolut-Variante 120 Min, Laplace 200 Min und das Weighted Information Gain 245 Min. Auf *satellite* ist der Unterschied noch größer, Laplace benötigt hier mit 290 Min die meiste Zeit. Ein Lauf mit der Absolut-Variante dauert im Durchschnitt 210 Min, die Gain-Variante benötigt dagegen 20 Min. Noch weniger braucht bei diesem Datensatz das Weighted Information Gain, ein Lauf dauert hier nur 10 s.

Insgesamt bestätigen sich hier die Ergebnisse mit den vorher überprüften Datensätzen. Die Genauigkeiten der Heuristiken unterscheiden sich, mit Ausnahme der Gain-Variante bei *proteins*, nicht besonders stark. Bei der Theoriegröße liefert die Gain-Variante die kleinsten Ergebnisse, gefolgt von der Absolut-Variante. Die Laufzeiten weisen dagegen noch größere Unterschiede auf, lassen allerdings wie bei den vorherigen Datensätzen keinen Regelmäßigkeiten erkennen.

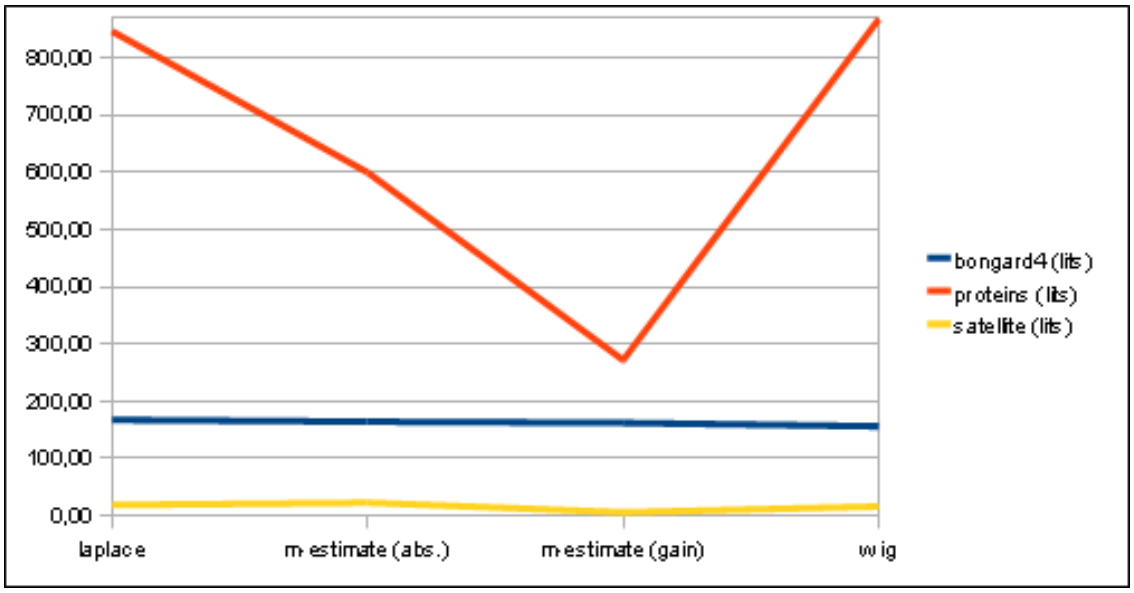

Abbildung 4.10.: Literalanzahl der Theorie (zusätzliche Datensätze)

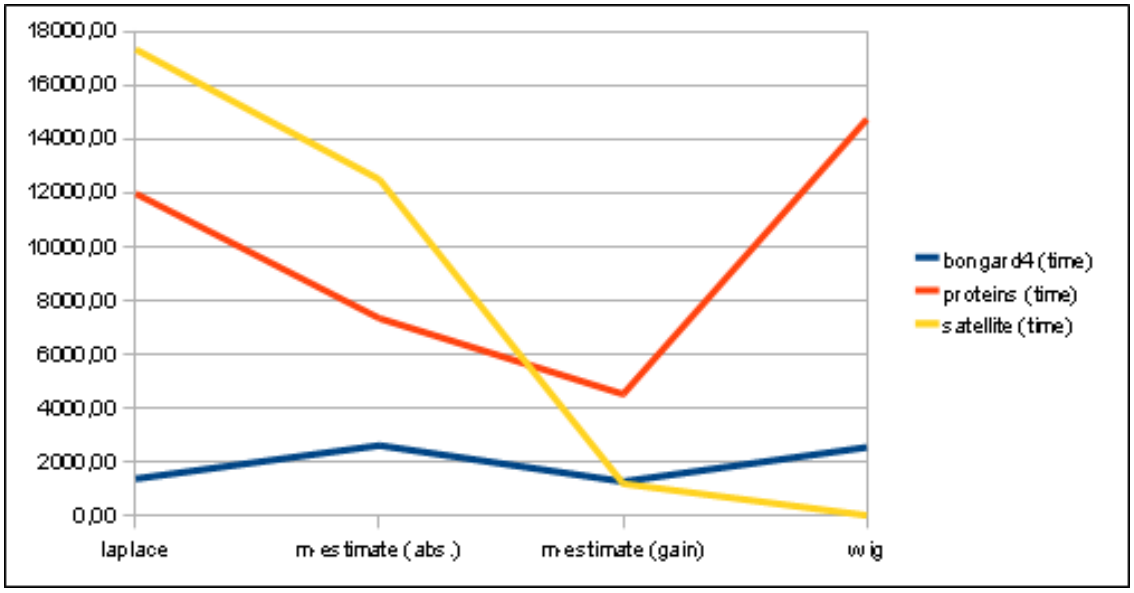

Abbildung 4.11.: Zeitaufwand (zusätzliche Datensätze)

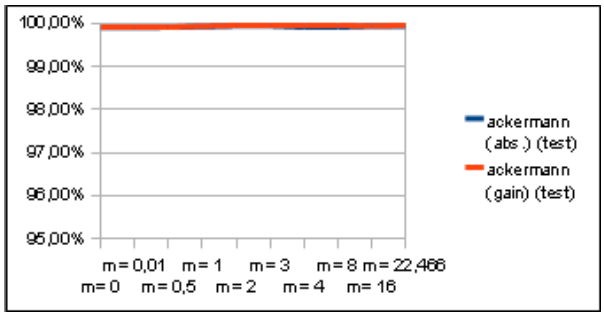

Abbildung 4.12.: Genauigkeit auf den Validierungsdaten (ackermann)

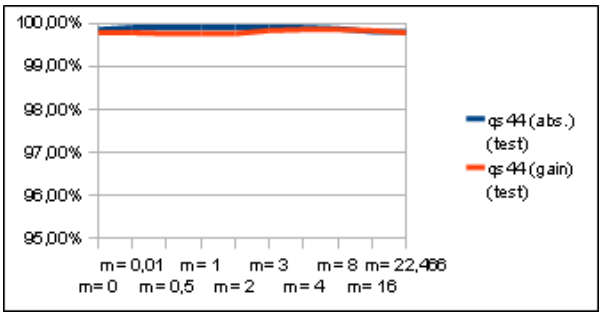

Abbildung 4.13.: Genauigkeit auf den Validierungsdaten (qs44)

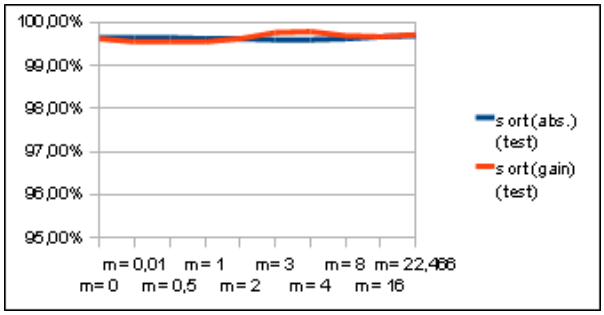

Abbildung 4.14.: Genauigkeit auf den Validierungsdaten (sort)

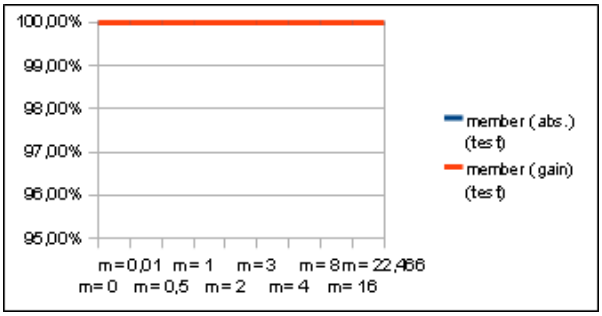

Abbildung 4.15.: Genauigkeit auf den Validierungsdaten (member)

4.2.2 Parameter des m-estimates

Analog zum Vergleich der Heuristiken wird als erstes die Genauigkeit auf Trainings- und Testdaten betrachtet, aufgeteilt in Gruppen von Datensätzen. Anschließend folgt wieder ein Vergleich der Theoriegröße und, bei einem Teil der Datensätze, des Zeitaufwands.

Genauigkeit

Aus Gründen der Übersichtlichkeit wird im Folgenden nur die Genauigkeit auf den Testdaten grafisch dargestellt, die Darstellungen der Genauigkeit auf den Trainingsdaten befinden sich im Anhang A.4.

Bei den Datensätzen, die der Referenzimplementierung beiliegen, zeigt sich zunächst kein Einfluss des Parameters m für das m-estimate. Bei den Datensätzen *ackermann* (Abbildung 4.12), *qs44* (Abbildung 4.13) und *sort* (Abbildung 4.14) erreicht die Heuristik immer eine Genauigkeit von annähernd 100 % auf Trainings- und Validierungsdaten.

Mit *member* (Abbildung 4.15) erzielen beide Varianten auf den Test- und die Absolut-Variante auf den Trainingsdaten 100 %. Mit $m < 0.5$ beträgt die Genauigkeit der Gain-Variante auf den Trainingsdaten ebenfalls 100 %, für größere Parameterwerte 99 %.

Leichte Unterschiede zeigen sich bei *ncm* (Abbildung 4.16). Hier wird bei den meisten Werten von m eine Genauigkeit von fast 100 % auf den Trainingsdaten sowie 99 % (Absolut-Variante) und 98 % (Gain-Variante) auf den Testdaten erzielt. Eine Ausnahme bildet $m = 0$ bei der Gain-Variante, hier ist das Ergebnis mit 93 % / 92 % merklich schlechter. Zudem sinkt mit $m = 22.466$ die Genauigkeit auf den Validierungsdatensätzen bei beiden Varianten leicht auf 98 % beziehungsweise 97 %.

Beide Varianten des m-estimates erreichen bei *crx* (Abbildung 4.17) auf den Trainingsdaten eine mit wachsendem m ansteigende Genauigkeit von 96 % bis 99 %. Auf den Testdaten schwankt diese über alle m unregelmäßig im Bereich zwischen 79 % und 81 %.

Bei dieser Gruppe Datensätze lassen sich also insgesamt nur minimale Unterschiede zwischen den verschiedenen Werten

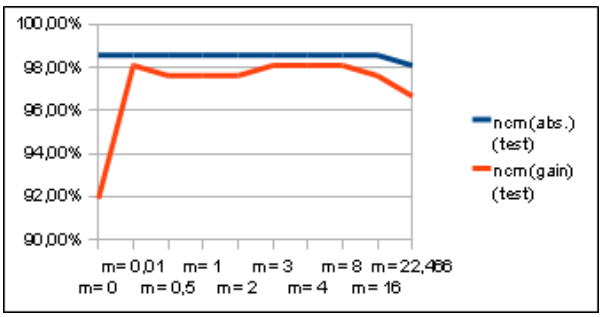

Abbildung 4.16.: Genauigkeit auf den Validierungsdaten (ncm)

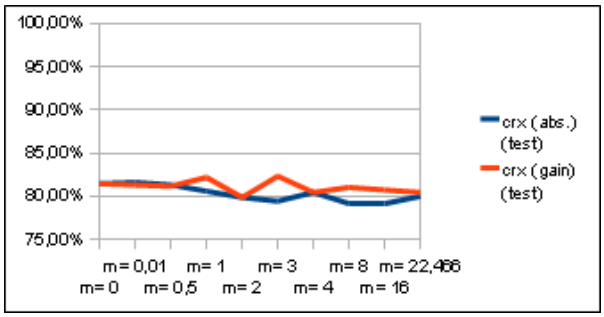

Abbildung 4.17.: Genauigkeit auf den Validierungsdaten (crx)

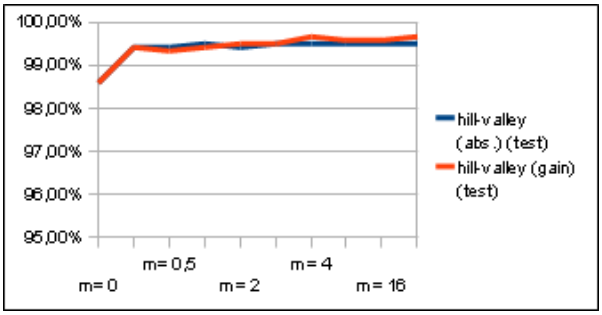

Abbildung 4.18.: Genauigkeit auf den Validierungsdaten (hill-valley)

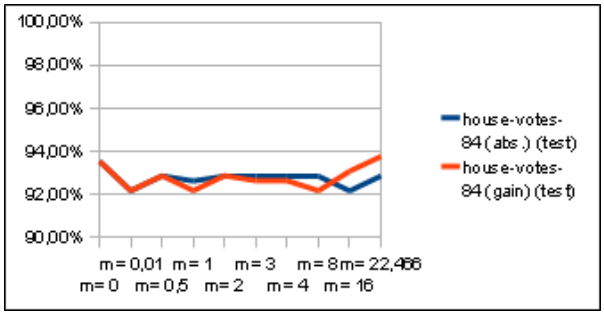

Abbildung 4.19.: Genauigkeit auf den Validierungsdaten (house-votes-84)

für m erkennen.

Auch bei dem propositionalen Datensatz *hill-valley* (Abbildung 4.18) hat die Wahl des Parameters keine großen Auswirkungen. Die Genauigkeit beträgt durchgängig 100 % / 99 %.

Im Gegensatz dazu zeigen sich bei *house-votes-84* (Abbildung 4.19) kleine Unterschiede. Die Genauigkeit auf den Validierungsdaten schwankt bei beiden Varianten ungleichmäßig zwischen 92 % und 94 %, auf den Trainingsdaten liegt sie durchgängig bei 99 %.

Auch bei *monks-3* (Abbildung 4.20) zeigen sich leichte Änderungen. Mit kleinen Werten für m erreicht der Algorithmus 98 % Genauigkeit auf den Trainings- und 97 % auf den Testdaten, für m > 2 (Absolut-Variante) beziehungsweise m > 0.01 (Gain-Variante) verbessert sich diese auf 99 % / 98 %.

Bei *krkopt* (Abbildung 4.21) kehrt sich dieser Effekt um, ab m > 8 verschlechtert sich das Ergebnis bei der Absolut-Variante geringfügig von 100 % / 99 % auf 98 % / 97 %. Die Gain-Variante erreicht von m unabhängig immer 100 % / 99 %.

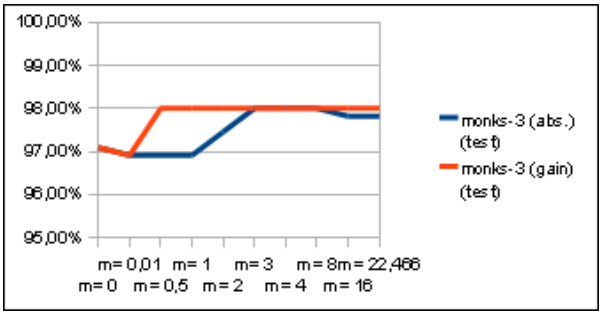

Abbildung 4.20.: Genauigkeit auf den Validierungsdaten (monks-3)

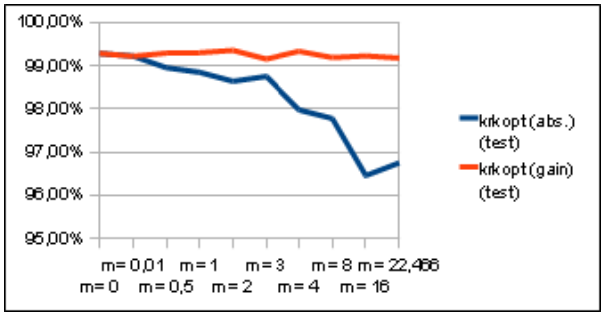

Abbildung 4.21.: Genauigkeit auf den Validierungsdaten (krkopt)

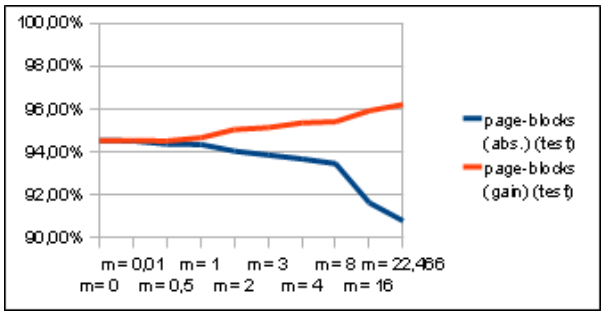

Abbildung 4.22.: Genauigkeit auf den Validierungsdaten (page-blocks)

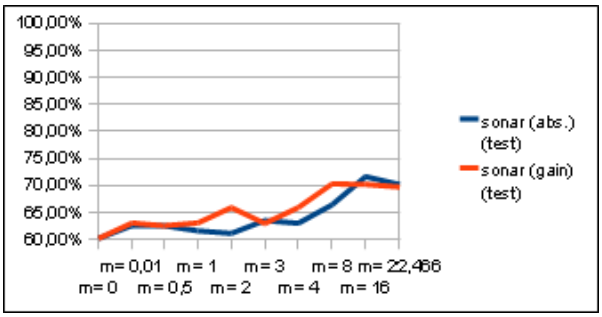

Abbildung 4.23.: Genauigkeit auf den Validierungsdaten (sonar)

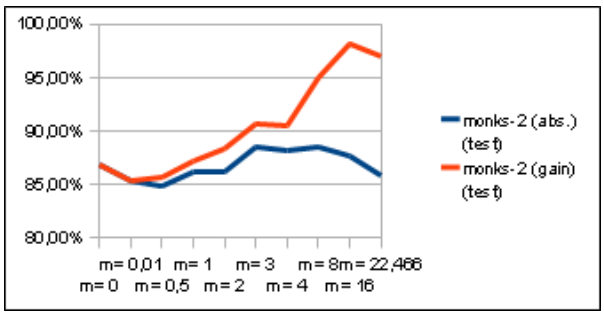

Abbildung 4.24.: Genauigkeit auf den Validierungsdaten (monks-2)

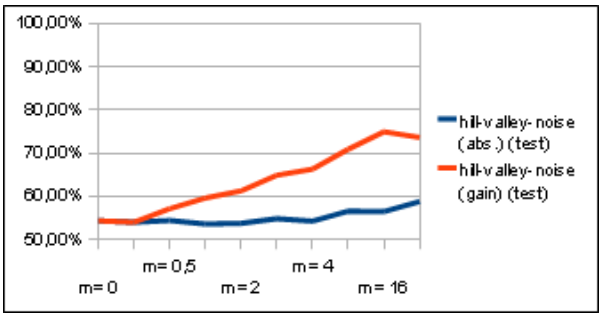

Abbildung 4.25.: Genauigkeit auf den Validierungsdaten (hill-valley-noise)

Ähnliches gilt für *page-blocks* (Abbildung 4.22). Hier verschlechtert sich das Ergebnis der Absolut-Variante ab m > 4 von 97 % / 95 % zu 92 % / 91 %. Die Gain-Variante verbessert sich an dieser Stelle jedoch von den gleichen Ausgangswerten auf 99 % / 96 %.

Bei *sonar* (Abbildung 4.23) ist eine deutlichere Änderung zu beobachten. Beide Varianten werden mit steigendem m besser. Ausgehend von 96 % / 60 % werden bis zu 99 % (Absolut-Variante) beziehungsweise 100 % (Gain-Variante) Genauigkeit auf den Trainings- und 70 % auf den Validierungsdaten erzielt. Die größte Verbesserung findet dabei bei m > 4 statt und fällt bei der Gain-Variante stellenweise stärker aus.

Ein großer Unterschied zwischen den zwei Varianten zeigt sich bei *monks-2* (Abbildung 4.24). Beide erreichen konstant 100 % Genauigkeit auf den Trainingsdaten. Bis zu m <= 4 schwankt der Wert auf den Validierungsdaten bei beiden um 86 %. Ab m > 4 verbessert er sich bei der Gain-Variante auf bis zu 97 %, während sie bei der Absolut-Variante nahezu unverändert bleibt. Bei diesem Datensatz verringert also ein großes m das Overfitting bei der Gain-Variante.

Dieses Verhalten bestätigt sich bei Verwendung von *hill-valley-noise* (Abbildung 4.25), wo die Absolut-Variante nur geringfügig von einem steigenden m profitiert. Es findet lediglich eine Verbesserung von 74 % / 54 % auf 77 % / 59 % statt. Unter Einsatz der Gain-Variante fällt diese deutlich stärker aus, vom gleichen Ausgangswert wird 98 % Genauigkeit auf den Trainings- und 74 % auf den Validierungsdaten erreicht.

Mit den propositionalen Datensätzen zeigen sich erste Unterschiede zwischen verschiedenen Werten für m, wobei steigende Werte meist einen Gewinn an Genauigkeit auf den Testdaten bedeuten. Dieser fällt diese jedoch meist sehr gering aus. Es ist zudem festzustellen, dass die Auswirkungen bei der Gain-Variante in der Regel etwas deutlicher ausfallen. Außerdem deuten diese Ergebnisse darauf hin, dass der Einfluss des Parameters auch sehr stark von verwendeten Datensatz abhängig ist.

Wie auch in den anderen Gruppen gibt es bei den relationalen Datensätzen eine Problemstellung, bei dem eine Änderung des Parameters keinen Einfluss auf das Ergebnis hat: bei *uwcse* (Abbildung 4.26) liegen die Werte konstant bei 96 % / 96 % mit der Absolut- und 97 % / 96 % mit der Gain-Variante.

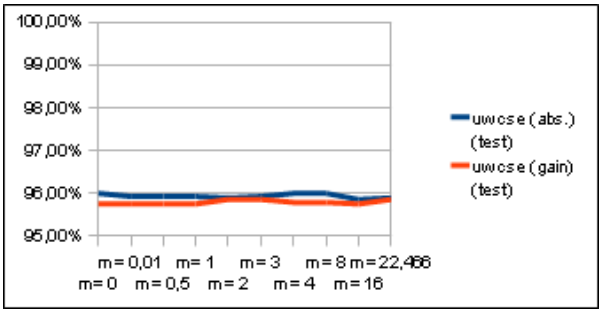

Abbildung 4.26.: Genauigkeit auf den Validierungsdaten (uwcse)

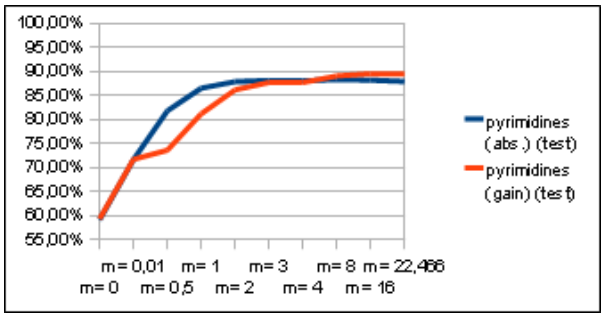

Abbildung 4.27.: Genauigkeit auf den Validierungsdaten (pyrimidines)

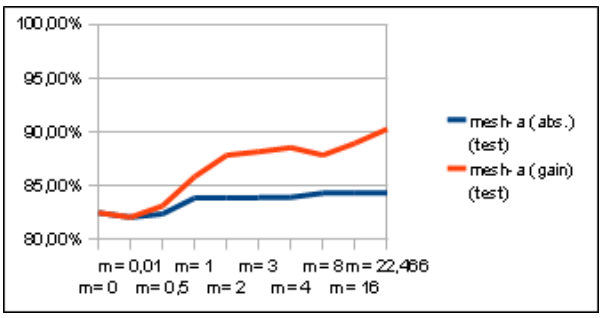

Abbildung 4.28.: Genauigkeit auf den Validierungsdaten (mesh-a)

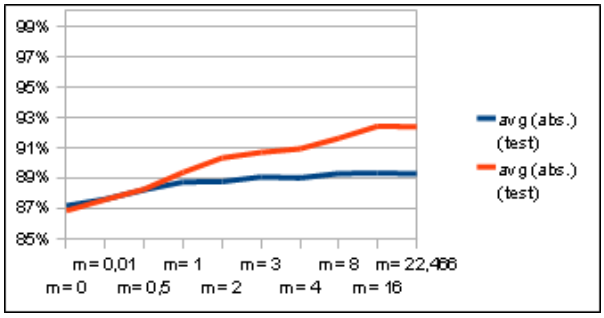

Abbildung 4.29.: Genauigkeit auf den Validierungsdaten (Durchschnitt)

Im Gegensatz dazu ändert sich die Genauigkeit auf den Testdaten bei *pyrimidines* (Abbildung 4.27) sehr stark. Sie steigt von anfangs 59 % bis zu m = 2 auf 90 % (Absolut-Variante) und 88 % (Gain-Variante). Bei weiter steigendem m zeigt sich keine erneute Änderung der Messwerte. Dabei ist die Gain-Variante für $0.01 < m < 2$ kurzzeitig um ungefähr 5 % besser. Von beiden Varianten wird durchgängig 100 % Genauigkeit auf den Trainingsdaten erreicht.

Die Datensätze *mesh-a* (Abbildung 4.28), *mesh-b*, *mesh-c*, *mesh-d* und *mesh-e* (siehe Anhang A.4) verhalten sich alle sehr ähnlich, weswegen hier nur *mesh-a* exemplarisch betrachtet wird. Mit einem $m < 1$ werden hier von beiden Varianten Genauigkeiten von 100 % beziehungsweise 82 % erreicht. Mit steigendem m steigt sie auf den Testdaten mit der Absolut-Variante auf 84 % und mit der Gain-Variante auf 90 %. Ab $m > 4$ fällt zusätzlich die Genauigkeit der Gain-Variante auf den Trainingsdatensätzen, wodurch erkennbar ist, dass an dieser Stelle Overfitting vermieden wird.

Bei den relationalen Datensätzen bestätigt sich also die Beobachtung, dass ein steigendes m insbesondere bei der Gain-Variante die Genauigkeit auf den Testdaten erhöhen kann. Aber auch hier fällt diese Verbesserung bei den meisten Datensätzen gering aus.

Zuletzt wurde der Durchschnitt über alle Datensätze gebildet (Abbildung 4.29), um eine datensatzunabhängige Bewertung der Werte für m vornehmen zu können. Diese stimmt mit dem zuvor festgestellten Trend überein, beide Varianten der Heuristik profitieren unterschiedlich stark von einem wachsenden m . Die Absolut-Variante erreicht mit jedem m eine Genauigkeit von 98 % auf den Trainingsdaten, auf den Testdaten steigt sie dabei mit wachsendem Wert des Parameters geringfügig von 87 % auf 89 %. Unter Verwendung der Gain-Variante verstärkt sich dieser Effekt, auf den Testdaten erhöht sich die Genauigkeit von 87 % auf 92 %. Gleichzeitig verbessert sie sich auf den Trainingsdaten von 98 % auf 99 %.

Auch wenn dieser Trend klar erkennbar ist, schwanken die Werte bei Betrachtung der Nachkommastellen leicht. So wird von beiden Varianten mit dem Wert m = 16 die höchste Genauigkeit auf den Testdaten erzielt, der Unterschied zu m = 22.466 beträgt allerdings nur jeweils 0.1 %.

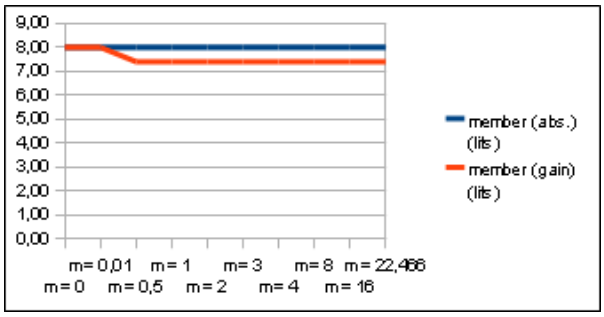

Abbildung 4.30.: Literalanzahl der Theorie (member)

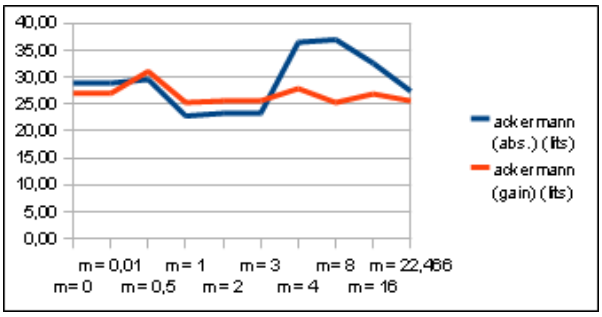

Abbildung 4.31.: Literalanzahl der Theorie (ackermann)

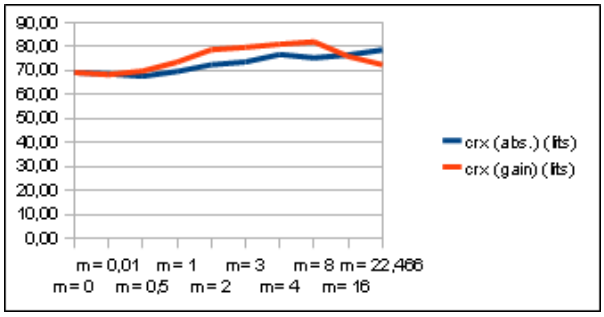

Abbildung 4.32.: Literalanzahl der Theorie (crx)

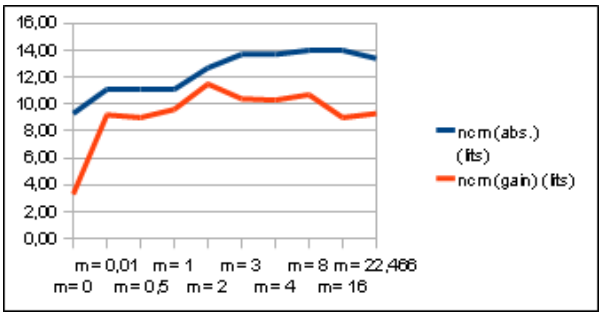

Abbildung 4.33.: Literalanzahl der Theorie (ncm)

Theoriegröße

Wie auch beim Vergleich der Heuristiken wird an dieser Stelle nur die Anzahl der Literale in Form von Graphen wiedergegeben.

Beim mitgelieferten Datensatz *member* (Abbildung 4.30) erzeugen alle Heuristiken 3 Regeln. Diese bestehen meist aus 8 Literalen, bei der Gain-Variante sind sie für m > 0.01 mit 7.4 etwas kleiner.

Auch mit *ackermann* (Abbildung 4.31) zeigt sich kein klarer Trend, die Theoriegröße schwankt bei beiden Varianten zwischen 5 und 6 Regeln sowie 23 und 37 (Absolut-Variante) beziehungsweise 25 und 31 Literalen (Gain-Variante).

Auch bei *crx* (Abbildung 4.32) schwankt die Anzahl der gelernten Regeln bei der Absolut-Variante zwischen 35 und 42, die Literalanzahl steigt mit wachsendem m von 68 auf 79. Mit der Gain-Variante fällt die Anzahl der Regeln von 42 auf 28, die Anzahl der Literale schwankt zwischen 68 und 82.

Betrachtet man *ncm* (Abbildung 4.33), zeigt sich, dass die Absolut-Variante bei diesem Datensatz für jedes m eine größere Theorie als die Gain-Variante lernt. Die Regelanzahl fällt dabei von 4 auf 3, die Anzahl der Literale steigt schwankend von 9 auf 14. Mit der Gain-Variante fällt die Anzahl der Regeln von Stelle 3 auf 2, die Literalanzahl erhöht sich von 3 auf 11.

Bei dem Datensatz *qs44* (Abbildung 4.34) ist hingegen die mit der Gain-Variante gelernte Theorie meist komplexer. Während das Ergebnis der Absolut-Variante zwischen 5 und 6 Regeln sowie 26 und 32 Literalen schwankt, fallen die Werte der Gain-Variante mit steigendem m von 7 auf 4 Regeln und von 53 auf 26 Literale.

Auf *sort* (Abbildung 4.35) fällt die Regelanzahl bei beiden Varianten von 8 auf 7 Regeln. Für große m fällt zudem die Anzahl der benötigten Literale, bei der Absolut-Variante von 46 auf 34 und bei der Gain-Variante von 56 auf 41.

Bei den Datensätzen, die FOIL beiliegen, zeigt sich somit kein klarer Trend in der Theoriegröße. Die gelernten Regeln bestehen im Mittel aus 1.5 bis 8 Literalen.

Betrachtet man die propositionalen Datensätze, zeigt sich bei *monks-3* (Abbildung 4.36) zunächst kein Unterschied

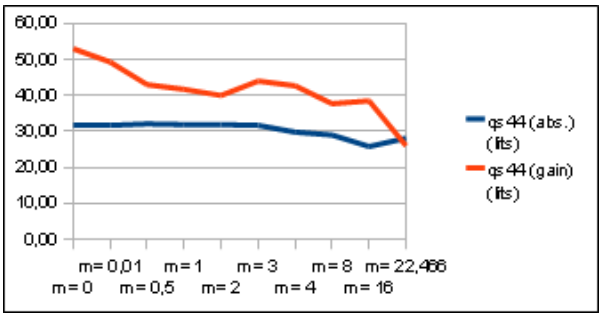

Abbildung 4.34.: Literalanzahl der Theorie (qs44)

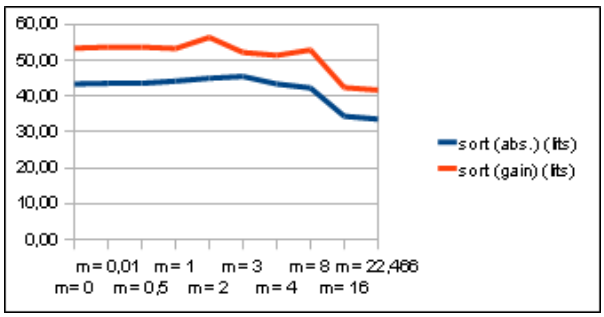

Abbildung 4.35.: Literalanzahl der Theorie (sort)

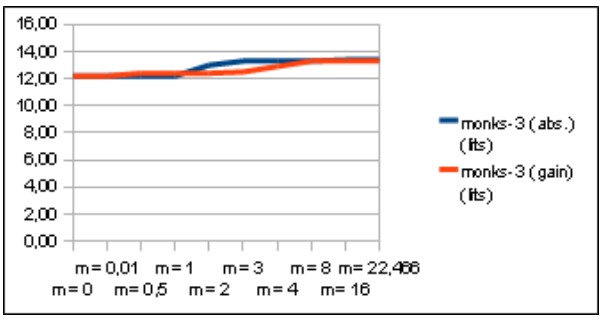

Abbildung 4.36.: Literalanzahl der Theorie (monks-3)

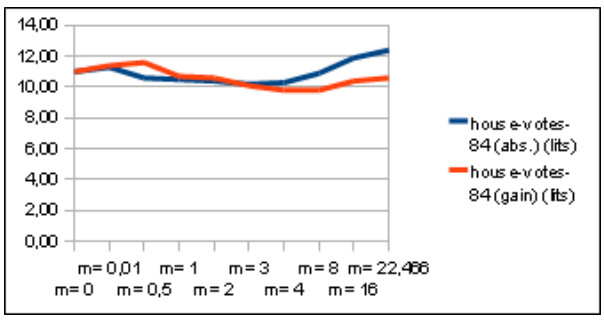

Abbildung 4.37.: Literalanzahl der Theorie (house-votes-84)

zwischen den Varianten des m-estimates. Bei beiden besteht die Theorie aus 9 Regeln, die Literalanzahl steigt mit wachsendem m von 12 auf 13.

Bei *house-votes-84* (Abbildung 4.37) verhalten sich die Varianten dagegen sehr unterschiedlich. Die Regelanzahl steigt mit der Absolut-Variante von 10 auf 12, mit der Gain-Variante fällt sie dagegen von 11 auf 9. Die Anzahl der Literale steigt mit der Absolut-Variante für große m von 10 auf 12, bei der Gain-Variante schwankt sie im gleichen Bereich unregelmäßig. Auffallend ist, dass die gelernten Regeln in beiden Fällen meist nur aus einem Kopf bestehen.

Die mit der Absolut-Variante auf *hill-valley-noise* (Abbildung 4.38) gelernte Theorie wird mit wachsendem m kleiner, die Anzahl der Regeln fällt von 87 auf 65 und die der Literale von 109 auf 87. Auch bei der Gain-Variante fällt die Regelanzahl von 87 auf 31, die Literalanzahl schwankt unregelmäßig zwischen 106 und 115.

Ein vergleichbares Verhalten lässt sich bei *krkopt* (Abbildung 4.39) beobachten, bei der Absolut-Variante fallen die Werte von 292 auf 123, beziehungsweise von 330 auf 201. Eine Ausnahme bildet jedoch m = 22.466, hier steigt die Anzahl der Literale erneut auf 252 an. Die Regelanzahl mit der Gain-Variante fällt von 292 auf 257, bei einer zwischen 314 und 330 schwankenden Literalanzahl.

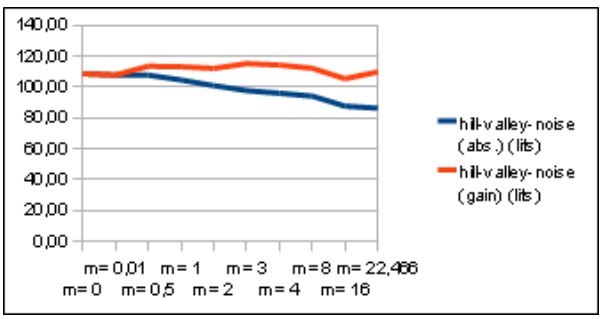

Abbildung 4.38.: Literalanzahl der Theorie (hill-valley-noise)

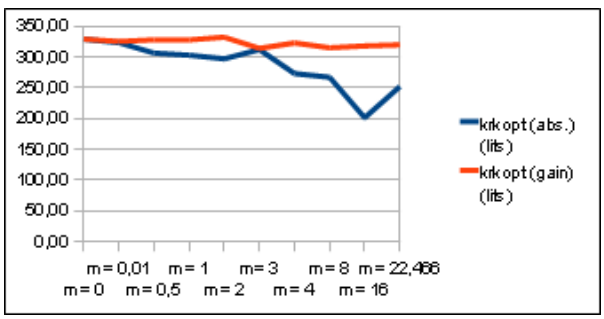

Abbildung 4.39.: Literalanzahl der Theorie (krkopt)

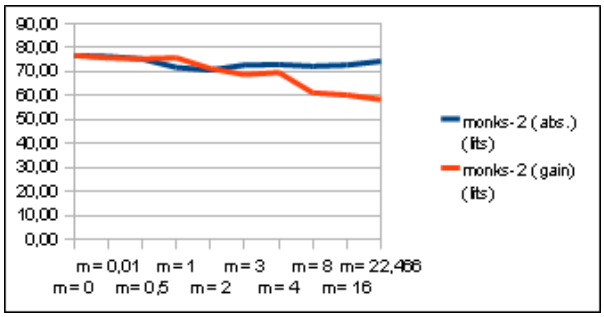

Abbildung 4.40.: Literalanzahl der Theorie (monks-2)

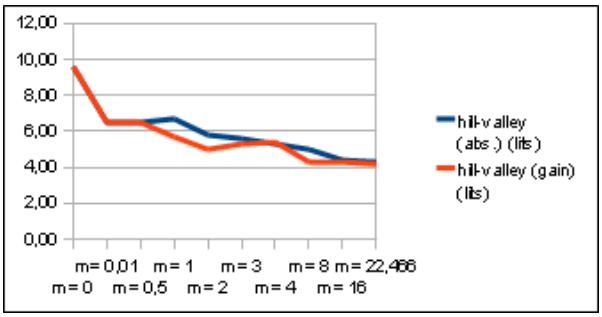

Abbildung 4.41.: Literalanzahl der Theorie (hill-valley)

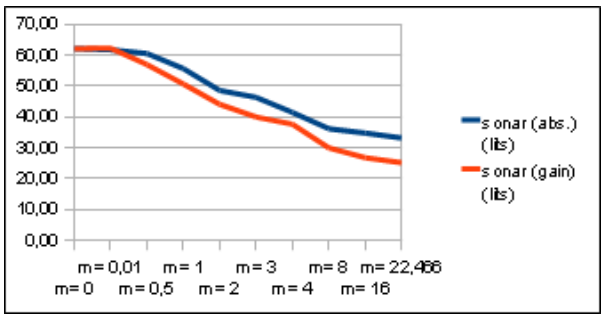

Abbildung 4.42.: Literalanzahl der Theorie (sonar)

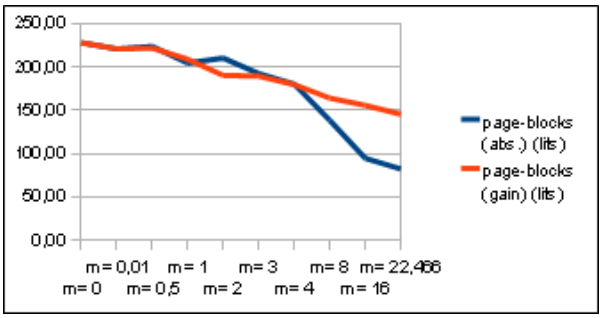

Abbildung 4.43.: Literalanzahl der Theorie (page-blocks)

Bei *monks-2* (Abbildung 4.40) verringern sich dagegen die Werte der Gain-Variante konstant mit wachsendem Parameter m. Die Anzahl der Regeln fällt von 54 auf 27, die der Literale von 76 auf 58. Mit der Absolut-Variante fällt die Regelanzahl unregelmäßig von 54 auf 43, die Literale schwanken zwischen 77 und 71.

Auf *hill-valley* (Abbildung 4.41) verhalten sich die Varianten dagegen identisch, bei beiden fällt die Anzahl der gelernten Regeln von 6 auf 2. Auch die Anzahl der Literale, aus denen diese bestehen, verringert sich von 10 auf 4.

Auch bei *sonar* (Abbildung 4.42) verringert sich die Größe der Theorie mit steigendem m. Mit der Absolut-Variante fallen die Werte von 40 auf 16 Regeln und von 62 auf 33 Literale. Noch deutlicher ist dieses Verhalten bei der Gain-Variante, deren Theoriegröße sich von den gleichen Startwerten auf 8 Regeln und 25 Literale verringert.

Umgekehrt verkleinert sich die Theorie der Absolut-Variante auf *page-blocks* (Abbildung 4.43) stärker. Ihre Werte fallen von 73 auf 29 Regeln und von 228 auf 82 Literale, von der Gain-Variante werden bei gleichem Ausgangswert 38 Regeln und 145 Literale erreicht.

Bei den meisten propositionalen Datensätzen verringert sich also bei mindestens einer Variante die Größe der Theorie mit wachsendem m. Auf welcher Variante dieser Effekt am deutlichsten ausfällt, ist dabei je nach Datensatz unterschiedlich. Die gelernten Regeln fallen bei dieser Gruppe mit durchschnittlich 1 bis 4 Literalen kleiner als bei den mitgelieferten Datensätzen aus.

Da sich die relationalen Datensätze *mesh-a* (Abbildung 4.44), *mesh-b* ,*mesh-c*, *mesh-d*, und *mesh-e* (siehe Anhang A.4) auch an dieser Stelle ähnlich verhalten, wird erneut nur *mesh-a* betrachtet. Bei diesem lernen beide Varianten des m-estimates für m = 0 eine Theorie aus 412 Regeln und 794 Literalen. Mit der Absolut-Variante fallen diese Werte mit wachsendem Parameter geringfügig auf 409 Regeln und 791 Literale. Deutlich größer sind die Auswirkungen des Parameters auf die Gain-Variante, hier verringert sich die Größe auf 87 Regeln und 297 Literale.

Auch bei *uwcse* (Abbildung 4.45) fallen die Werte mit wachsendem m, bei der Absolut-Variante von 5 auf 3 Regeln und von 26 auf 14 Literale, wobei m = 22.466 mit 15 Literalen eine Ausnahme bildet. Aus der Gain-Variante resultiert für jedes m eine größere Theorie als aus der Absolut-Variante. Sie verkleinert sich lediglich von 7 auf 6 Regeln und von 40 auf 35 Literale.

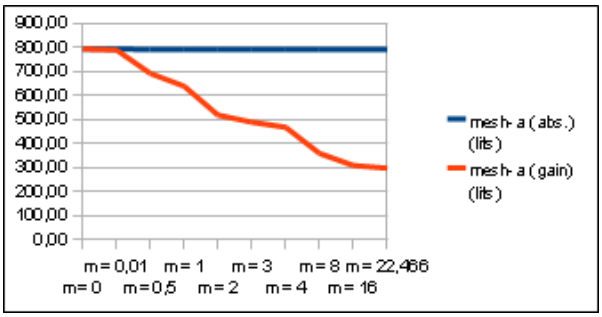

Abbildung 4.44.: Literalanzahl der Theorie (mesh-a)

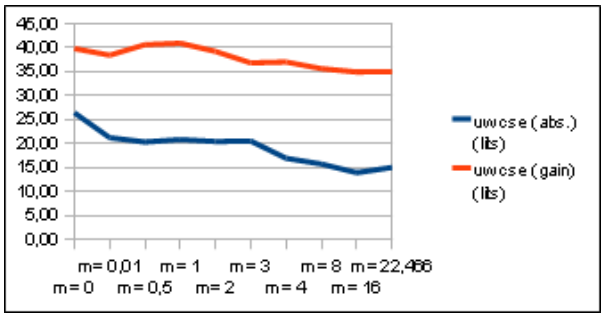

Abbildung 4.45.: Literalanzahl der Theorie (uwcses)

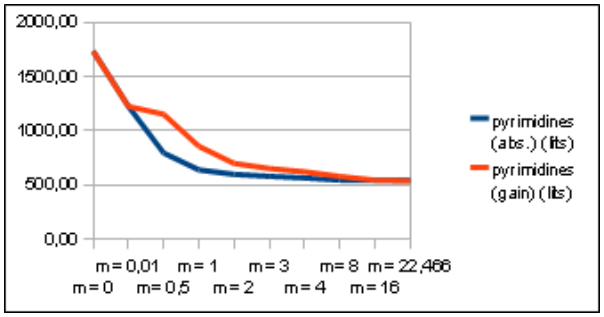

Abbildung 4.46.: Literalanzahl der Theorie (pyrimidines)

Bei *pyrimidines* (Abbildung 4.46) zeigt sich ebenfalls eine Verkleinerung der Theorie mit wachsendem Parameter. Unter Benutzung der Absolut-Variante verringert sich die Anzahl der Regeln von 1734 auf 309 und die der Literale von 1737 auf 541 (Ausnahme: m = 22.466 mit 545 Literalen). Die Werte der Gain-Variante fallen noch etwas stärker von 1731 auf 221 Regeln und von 1735 auf 538 Literale. Der steilste Fall findet dabei bis m = 1 statt, für $0.01 < m < 4$ produziert die Gain-Variante eine deutlich größere Theorie als die Absolut-Variante. Auffallend ist zudem bei beiden Varianten, dass die Regellänge mit steigendem m von 1 auf 2 zunimmt, also die zunächst nur aus einem Kopf bestehenden Regeln durch solche ersetzt werden, die auch einen Körper besitzen.

Der vorher beobachtete Effekt der fallenden Theoriegröße bei wachsendem Parameterwert zeigt sich somit bei allen relationalen Datensätzen. Die Regeln bestehen durchschnittlich aus 1 bis 6 Literalen.

Zeitaufwand

Wie beim Vergleich der Heuristiken werden an dieser Stelle wieder nur Datensätze mit einer Laufzeit von über 5 Sekunden betrachtet.

Bei dem Datensatz *page-blocks* (Abbildung 4.47) benötigen beide Varianten des m-estimates mit m = 0 10 s, bei

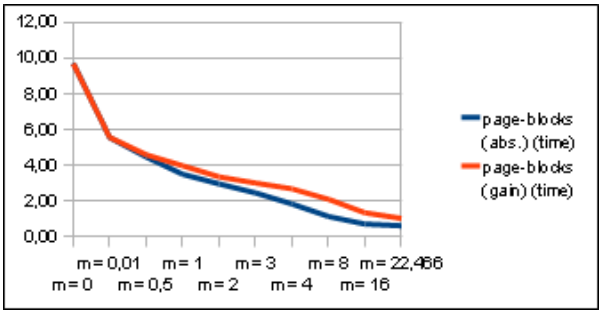

Abbildung 4.47.: Zeitaufwand (page-blocks)

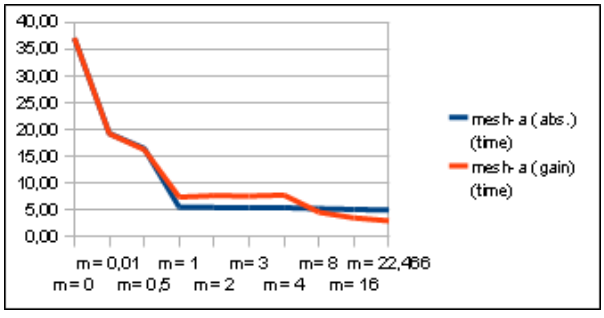

Abbildung 4.48.: Zeitaufwand (mesh-a)

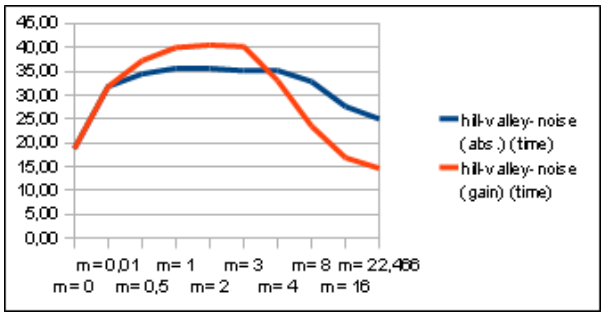

Abbildung 4.49.: Zeitaufwand (hill-valley-noise)

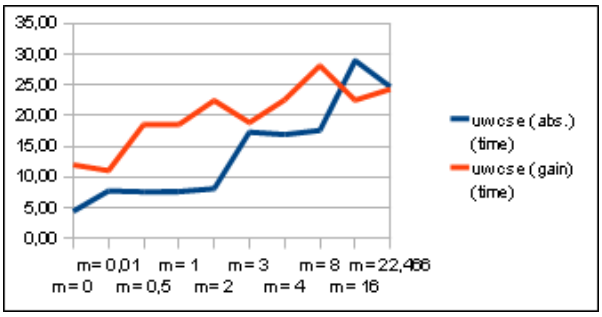

Abbildung 4.50.: Zeitaufwand (uwcse)

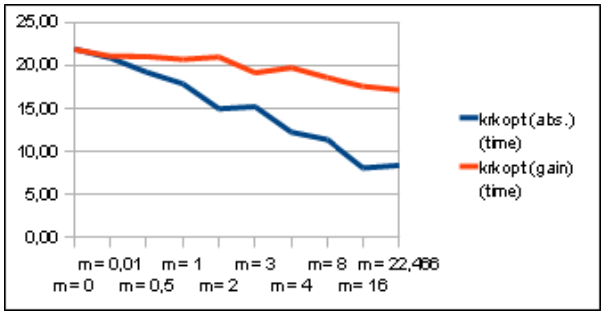

Abbildung 4.51.: Zeitaufwand (hill-krkopt)

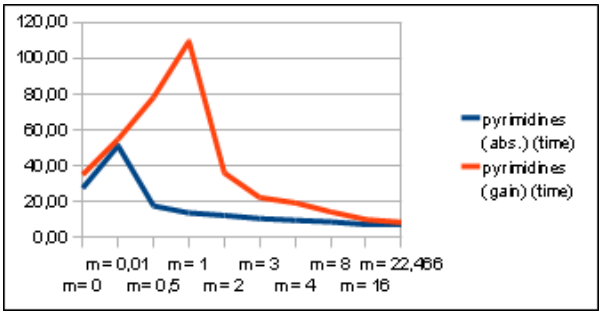

Abbildung 4.52.: Zeitaufwand (pyrimidines)

steigendem m fällt dieser Wert monoton auf 1 s. Der stärkste Abstieg findet im Intervall zwischen 0 und 0.01 statt. Dabei liegt die Gain- etwas oberhalb der Absolut-Variante.

Die Datensätze *mesh-a* (Abbildung 4.48), *mesh-b*, *mesh-c*, *mesh-d* und *mesh-e* (siehe Anhang A.4) verhalten sich wie bei den anderen Messungen auch hier annähernd gleich. Die Laufzeit für m = 0 beträgt 37 s. Bis m = 1 fällt dieser Wert auf 5 s (Absolut-Variante) und 7 s (Gain-Variante). Ab m = 4 fällt dieser für die Gain-Variante erneut und erreicht bei großen Werten für m 3 s.

Ein anderes Zeitverhalten zeigt sich bei *hill-valley-noise* (Abbildung 4.49). Hier steigt der Zeitaufwand von 19 s bis m = 1 zunächst auf 36 s (Absolut-Variante) beziehungsweise 40 s (Gain-Variante). Mit der Absolut-Variante fällt er ab m = 4 auf 25 s und mit der Gain-Variante ab m = 3 auf 15 s.

Betrachtet man *uwcse* (Abbildung 4.50), zeigt sich bei beiden Varianten ein stark schwankender Anstieg. Die Absolut-Variante beginnt mit 4 s bei m = 0 und steigt dann bis zu 29 s. Die Laufzeit mit der Gain-Variante erhöht sich von 12 s auf 28 s. Dabei ist die Absolut-Variante bis m = 8 schneller, für größere m liegt die Gain-Variante vorne und bei m = 22.466 benötigen sie ungefähr gleich viel Zeit.

Ein deutlicher Unterschied im Verhalten der beiden Variante zeigt sich bei *krkopt* (Abbildung 4.51). Beide Varianten benötigen mit m = 0 22 s, mit steigendem Parameter fällt die Dauer bei der Absolut-Variante auf 8 s. Die Laufzeit der Gain-Variante verringert sich deutlich weniger, es werden nur 17 s erreicht.

Auch bei *pyrimidines* (Abbildung 4.52) ist die Absolut-Variante schneller. Deren Laufzeit steigt nach einem Beginn mit 29 s auf bis zu 51 s (m = 0.01) an und fällt anschließend wieder sehr schnell auf 7 s. Im Gegensatz dazu benötigt die Gain-Variante anfangs 35 s und auch der zwischenzeitliche Anstieg fällt mit 109 s an der Stelle m = 1 deutlich stärker aus. Mit großen Werten für m nähert sie sich mit 8 s der Absolut-Variante an.

Noch größer fällt der Unterschied bei *qs44* (Abbildung 4.53) aus. Hier benötigt die Absolut-Variante durchgängig ungefähr 12 s. Die Laufzeit der Gain-Variante schwankt zwischen 69 s und 109 s, wobei das Maximum bei m = 0.01 erreicht wird und die Laufzeit mit steigendem m fällt.

Zusammenfassend stellt man fest, dass die Laufzeit bei größeren Werten für m zumeist abnimmt. Bei gleichen Werten für m benötigt die Absolut-Variante meist etwas weniger Zeit für einen Programmdurchlauf als die Gain-Variante.

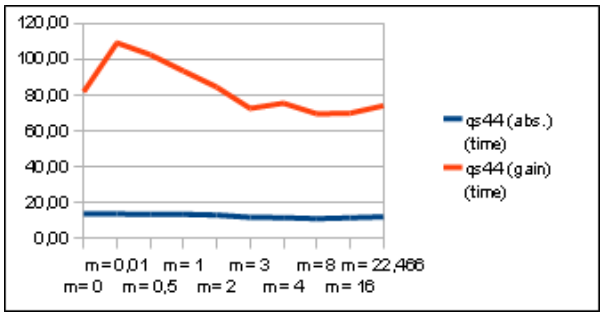

Abbildung 4.53.: Zeitaufwand (qs44)

4.2.3 Bewertung

Jeweils zwei der vier untersuchten Heuristiken liefern ähnliche Ergebnisse. Während die Genauigkeit der gelernten Regeln auf den Trainingsdaten, mit Ausnahme von *hill-valley-noise* und *proteins*, durchgängig gegen 100 % strebt, fällt sie auf den Validierungsdaten bei Laplace und der Absolut-Variante des m-estimates regelmäßig geringer als bei der Gain-Variante oder dem Weighted Information Gain aus. Dieser Unterschied macht bei ungefähr der Hälfte der untersuchten Datensätze bis zu 10 % aus. Dies ist ein Hinweis, dass die beiden erstgenannten Heuristiken eher zu Overfitting neigen.

Diese Gruppierung kann man dadurch interpretieren, dass sowohl Laplace als auch die Absolut-Variante des m-estimates bei der Regelbewertung von der anfänglichen Problemstellung, genauer gesagt der dort vorhandenen Verteilung von positiven und negativen Tupeln, ausgehen. Im Gegensatz dazu berücksichtigt die Gain-Variante, welche Tupel bereits von der unvollständigen Theorie abgedeckt werden, und das Weighted Information Gain berechnet den Fortschritt innerhalb der aktuell konstruierten Regel. Es ist daher möglich, dass die beiden ersten Heuristiken zu wenig Informationen über den bisherigen Fortschritt der Suche nach einer Theorie verwenden.

Bezüglich der Größe der gelernten Theorie lassen sich keine Regelmäßigkeiten erkennen. Bei den meisten Datensätzen werden von den Heuristiken ähnliche Ergebnisse erzielt, bei den übrigen lernen jeweils unterschiedliche Heuristiken die kleinsten Theorien. Auffallend ist jedoch, dass beide Varianten des m-estimates eher zu Extremwerten neigen, also häufiger die kleinste beziehungsweise größte Theorie erzeugen als Laplace und das Weighted Information Gain.

Die Laufzeit kann sich dabei sehr stark unterscheiden, selbst wenn ähnliche Ergebnisse bei den Genauigkeiten erzielt werden. Diese Beobachtung lässt darauf schließen, dass die verschiedenen Heuristiken unterschiedliche Pfade bei der Suche zur Konstruktion der Theorie wählen, auch wenn das Endergebnis ähnlich ist. Außerdem ist es möglich, dass zunächst komplexere Regeln während der Nachbearbeitung gekürzt werden. Insgesamt zeigt sich bei der Laufzeit eine sehr starke Abhängigkeit vom jeweils bearbeiteten Datensatz, es ist keine allgemeingültige Reihenfolge der Heuristiken nach der Dauer eines Programmlaufs erkennbar.

Beide Varianten des m-estimates bieten den anderen Heuristiken gegenüber den Vorteil, dass über den Parameter m ein datensatzabhängiges Feintuning durchgeführt werden kann. Dieses verbessert das Ergebnis im Bezug auf die Genauigkeiten meist nur in geringem Maße. Bei etwa 75 % der Datensätze kann allerdings, abhängig von der gewählten Variante, die Größe der gelernten Theorie durch einen höher gewählten Parameter verringert werden. Der Wert von m hat zudem in einigen Fällen große Auswirkungen auf die Laufzeit des Algorithmus. Insgesamt sind die erzielten Ergebnisse jedoch sehr datensatzspezifisch und erlauben keine allgemeine Beurteilung, wie der Parameter zu wählen ist.

5 Zusammenfassung und Ausblick

In dieser Bachelorarbeit wurde ein Überblick über die Auswirkungen von vier verschiedenen Heuristiken auf FOIL gegeben. Dabei hat sich herausgestellt, dass die bereits vorhandene Heuristik, das Weighted Information Gain, zusammen mit der neu implementierten Gain-Variante des m-estimates im Vergleich am besten abschneidet. Die Absolut-Variante und Laplace zeigen sich dabei anfälliger für Overfitting und erzeugen bei ungefähr der Hälfte der getesteten Datensätze eine geringere Genauigkeit auf den Validierungsdatensätzen, bei annähernd gleicher Genauigkeit auf den Trainingsdaten.

Es wurde ebenfalls untersucht, wie sich die Wahl des Parameters m für das m-estimate auf das Ergebnis auswirkt. Hier wurde bei den meisten Datensätzen allerdings nur ein geringer Einfluss auf die Genauigkeiten festgestellt.

Außerdem wurde die Größe der gelernten Theorien und die Laufzeit des Algorithmus mit den unterschiedlichen Heuristiken und Werten für m betrachtet. Hierbei zeigten sich bei einigen Datensätzen starke Schwankungen, die aber eine starke Abhängigkeit vom konkreten Problem aufweisen und keinen allgemeingültigen Trend darstellen. Im Falle des m-estimates ist es jedoch in vielen Fällen möglich, die Größe der gelernten Theorie durch die Wahl eines höheren Parameterwertes zu verringern.

Basierend auf diesen Ergebnissen empfiehlt es sich daher nicht, die Heuristik für eine folgende Version des Algorithmus auszutauschen. Keine der neuen Bewertungsfunktionen konnte ein deutlich besseres Ergebnis als das bereits verwendete Weighted Information Gain erzielen.

Eine mögliche Erweiterung dieser Arbeit ist es, mehr Datensätze zu überprüfen. Der Prozess, verfügbare Datensätze für die Verwendung mit FOIL anzupassen, lässt sich nicht vollständig automatisieren. Bei dem Algorithmus wird zwar ein Tool mitgeliefert, das Daten aus dem C4.5-Format umwandelt, allerdings ist es hierfür nötig, Datentypen manuell zu ergänzen. Andere Datensätze liegen in keinem standardisierten Format vor und mussten daher gesondert behandelt werden. Aufgrund des daraus resultierenden Aufwandes war es nur möglich, eine begrenzte Anzahl von Datensätzen zu überprüfen. Zudem mussten einige Datensätze nach der Umwandlung entfernt werden, da der Algorithmus mit keiner der verwendeten Heuristiken ein Ergebnis erzielen konnte.

Des Weiteren kann man die erzielten Ergebnisse in Relation zu mFOIL setzen. Es wurde mit keiner der neuen Heuristiken ein besseres Ergebnis beim stark gestörten Datensatz *hill-valley-noise* erzielt. Teilweise fiel die Genauigkeit auf den Validierungsdaten mit diesen sogar deutlich höher als mit dem Weighted Information Gain aus. mFOIL, das diese Heuristiken verwendet, lieferte allerdings in einer Untersuchung von Lavrac und Dzeroski [LD94] bessere Ergebnisse als eine nicht modifizierte Version von FOIL. Dieser Umstand könnte weiter untersucht werden, indem man auch in anderen Datensätzen künstlich Noise einführt und die durchgeführten Messungen wiederholt. Bestätigt sich, dass mFOIL in dieser Domäne überlegen ist, könnte man weiterführend prüfen, welche der weiteren Modifikationen, beispielsweise die Verwendung einer Beam-Suche, für besseres Noise-Handling sorgen.

Literaturverzeichnis

[ACE] ACE Documentation - More data sets [http://www.cs.kuleuven.be/ dtai/ACE/doc/].

[AN] Asuncion, A. & Newman, D.J. (2007). UCI Machine Learning Repository [http://www.ics.uci.edu/ mlearn/MLRepository.html]. Irvine, CA: University of California, School of Information and Computer Science.

[BW03] Bell, S. & Weber, S. (1993). On the close logical realtionship between FOIL and the frameworks of Helft an Plotkin. In Proceedings of Third International Workshop on Inductive Logic Programming (127-147), Bled, Slovenia.

[DB92] Dzeroski, S. & Bratko I. (1992). Handling noise in inductive logic programming. In Proceedings of the Second International Workshop on Inductive Logic Programming, Tokyo, Japan.

[CN89] Clark, P. & Niblett T. (1989). The CN2 induction algorithm. Mach. Learn. 3, 4 (Mar. 1989) 261-283.

[FOIL] Ross Quinlan's personal homepage - FOIL Release 6 [http://www.rulequest.com/Personal/foil6.sh].

[ILPAD] ILP Applications and Datasets [http://www.doc.ic.ac.uk/ shm/applications.html].

[JF08] Janssen, F. & Fürnkranz, J. (2008). An empirical investigation of the trade-off between consistency and coverage in rule learning heuristics. In Horvath, T., Boulicaut, J.-F., and Berthold, M., editors, Proceedings of the 11th International Conference on Discovery Science (DS-08), Budapest, Hungary. Springer-Verlag.

[LD94] Lavrac, N. & Dzeroski, S. (1994). Inductive Logic Programming: Techniques and Applications. Ellis Horwood, New York, 1994.

[Q90] Quinlan, J.R. (1990). Learning Logical Definitions from Relations. Mach. Learn. 5, 3 (Sep. 1990), 239-266.

[QC93] Quinlan, J.R. & Cameron-Jones, R.M. (1993). FOIL: A Midterm Report. In Proceedings of the European Conference on Machine Learning (April 05 - 07, 1993). P. Brazdil, Ed. Lecture Notes In Computer Science, vol. 667. Springer-Verlag, London, 3-20.

[QC95] Quinlan, J.R. & Cameron-Jones, R.M. (1995). Induction of Logic Programs: FOIL and Related Systems. In New Generation Computing, Volume 13, 1995, 287-312.

[QR89] Quinlan, J.R. & Rives, R.L. (1989). Inferring decision trees using the Minimum Description Length Principle. In Information and Computation 89 (227-248). Academic Press, Inc.

A Anhang

A.1 Verwendung des Programms

Im Rahmen der Erweiterung sind drei neue Parameter hinzugekommen, mit denen man beim Programmstart die Heuristik auswählen kann.

-H Legt die Heuristik fest (0: Weighted Information Gain, 1: m-estimate, 2: Laplace). Voreinstellung: Weighted Information Gain.

-M Setzt bei Verwendung des m-estimates den Parameter m und hat keine Auswirkung, wenn eine andere Heuristik verwendet wird. Voreinstellung: 2.

-G Aktiviert die Gain-Variante des m-estimates. Hat ebenfalls keine Auswirkung auf die anderen Heuristiken. Voreinstellung: nicht gesetzt.

A.2 Geänderte Dateien

Um die durchgeführten Änderungen am Programmcode nachvollziehbar zu machen, werden im Folgenden die veränderten Dateien aufgeführt. Alle Erweiterungen und Änderungen wurden zudem im Quelltext mit Kommentaren kenntlich gemacht.

- defns.i
- extern.i
- evaluatelit.c
- finddef.c
- global.c
- input.c
- literal.c
- main.c
- search.c

A.3 Tabellarische Testdaten

Die Tabellen A.1 bis A.10 stellen die in Abschnitt 4.2.1 diskutierten Werte noch einmal tabellarisch dar.

	Trainingsdaten				Validierungsdaten			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
ackermann	99,97 %	99,96 %	99,98 %	99,99 %	99,92 %	99,95 %	99,96 %	99,91 %
crx	97,87 %	96,12 %	99,13 %	99,11 %	81,88 %	81,59 %	82,32 %	80,58 %
member	99,51 %	99,51 %	99,51 %	99,51 %	100,00 %	100,00 %	100,00 %	100,00 %
ncm	99,74 %	99,84 %	99,52 %	99,79 %	98,57 %	98,57 %	98,10 %	98,10 %
qs44	99,64 %	99,74 %	99,92 %	99,83 %	99,55 %	99,69 %	99,79 %	99,57 %
sort	99,93 %	99,95 %	99,99 %	99,97 %	99,86 %	99,91 %	99,86 %	99,88 %

Abbildung A.1.: Genauigkeit (mitgelieferte Datensätze)

	Trainingsdaten				Validierungsdaten			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
hill-valley	99,90 %	99,90 %	100,00 %	100,00 %	99,34 %	99,51 %	99,67 %	99,67 %
hill-valley-noise	74,08 %	77,16 %	96,86 %	99,72 %	53,72 %	58,75 %	74,92 %	74,67 %
house-votes-84	98,80 %	98,77 %	99,06 %	99,36 %	92,63 %	93,55 %	93,77 %	93,55 %
krkopt	99,95 %	99,91 %	99,95 %	99,89 %	99,33 %	99,28 %	99,35 %	99,27 %
monks-2	100,00 %	100,00 %	100,00 %	100,00 %	89,19 %	88,52 %	98,17 %	99,17 %
monks-3	98,48 %	98,96 %	98,96 %	98,96 %	96,91 %	98,00 %	98,00 %	98,00 %
page-blocks	97,02 %	97,25 %	98,59 %	99,33 %	95,03 %	94,54 %	96,22 %	96,82 %
sonar	96,05 %	98,13 %	99,84 %	100,00 %	60,62 %	71,62 %	70,26 %	75,93 %

Abbildung A.2.: Genauigkeit (propositionale Datensätze)

	Trainingsdaten				Validierungsdaten			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
mesh-a	99,95 %	100,00 %	97,21 %	99,77 %	83,39 %	84,32 %	90,28 %	92,82 %
mesh-b	99,91 %	100,00 %	97,18 %	99,53 %	83,48 %	83,96 %	90,47 %	91,85 %
mesh-c	99,91 %	100,00 %	97,22 %	99,71 %	84,00 %	84,06 %	89,67 %	92,56 %
mesh-d	99,90 %	100,00 %	97,51 %	99,65 %	83,99 %	84,61 %	90,63 %	92,88 %
mesh-e	99,89 %	100,00 %	97,64 %	99,79 %	83,61 %	84,32 %	89,90 %	92,37 %
pyrimidines	99,93 %	99,92 %	99,88 %	99,64 %	87,79 %	88,32 %	89,53 %	91,67 %
uwcse	96,19 %	96,36 %	96,56 %	96,49 %	95,96 %	96,00 %	95,86 %	95,93 %

Abbildung A.3.: Genauigkeit (relationale Datensätze)

	Regeln				Literale			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
ackermann	5,50	5,30	5,60	5,80	25,40	23,30	27,90	30,50
crx	40,00	35,30	39,90	24,70	73,80	68,60	79,70	76,40
member	3,00	3,00	3,00	3,00	8,00	8,00	8,00	8,00
ncm	3,30	3,40	2,50	2,50	10,60	11,10	10,70	13,20
qs44	7,10	6,40	8,00	7,50	38,30	33,60	51,40	46,90
sort	5,10	5,30	5,80	5,00	30,40	31,90	42,70	31,30

Abbildung A.4.: Theoriegröße (mitgelieferte Datensätze)

	Regeln				Literale			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
hill-valley	3,20	4,00	2,10	2,00	5,70	6,70	4,20	4,00
hill-valley-noise	81,00	65,30	36,80	20,10	101,00	86,40	105,50	109,10
house-votes-84	10,20	10,90	9,00	9,40	10,30	11,00	10,60	11,20
krkopt	289,60	292,10	294,20	242,00	330,10	329,50	332,60	385,80
monks-2	41,70	43,00	29,00	21,00	71,40	72,20	60,20	46,40
monks-3	9,10	8,50	9,40	8,50	12,30	13,30	12,40	13,80
page-blocks	39,70	73,40	38,20	28,30	130,40	228,00	145,30	140,50
sonar	29,20	17,90	10,70	6,30	50,60	34,80	30,00	22,50

Abbildung A.5.: Theoriegröße (propositionale Datensätze)

	Regeln				Literale			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
mesh-a	312,20	409,30	86,60	80,60	628,30	791,20	296,60	417,40
mesh-b	311,40	409,00	85,00	79,10	626,60	790,70	284,10	400,40
mesh-c	313,70	407,70	85,40	79,30	628,50	788,40	293,60	407,10
mesh-d	307,30	408,30	86,20	80,00	618,50	789,90	302,20	401,40
mesh-e	309,60	408,70	103,90	82,00	621,90	790,70	317,50	411,00
pyrimidines	407,50	309,00	241,30	170,90	598,30	544,90	544,70	459,10
uwcse	4,00	5,20	6,80	4,90	16,00	26,40	39,20	29,20

Abbildung A.6.: Theoriegröße (relationale Datensätze)

	laplace	**mest-abs**	**mest-gain**	**wig**
qs44	13,41	13,11	75,54	21,49
hill-valley-noise	35,48	25,02	16,91	12,96
krkopt	21,05	21,93	20,97	48,06
page-blocks	0,76	9,72	1,03	0,52
mesh-a	3,08	5,00	2,98	5,45
mesh-b	3,16	4,90	2,76	5,06
mesh-c	3,12	4,89	3,31	5,35
mesh-d	3,02	4,87	2,62	5,42
pyrimidines	12,29	8,79	10,03	56,91
uwcse	7,40	4,43	22,46	41,07

Abbildung A.7.: Zeitaufwand (verschiedene Datensätze)

	Trainingsdaten				Validierungsdaten			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
bongard4	96,05 %	97,60 %	92,31 %	96,59 %	93,23 %	95,18 %	90,15 %	94,01 %
proteins	95,37 %	90,47 %	77,66 %	99,34 %	65,14 %	63,51 %	62,18 %	62,28 %
satellite	97,40 %	100,00 %	98,52 %	100,00 %	95,74 %	97,05 %	98,52 %	100,00 %

Abbildung A.8.: Genauigkeit (zusätzliche Datensätze)

	Regeln				Literale			
	laplace	**mest-abs**	**mest-gain**	**wig**	**laplace**	**mest-abs**	**mest-gain**	**wig**
bongard4	17,70	16,30	18,10	14,80	167,50	164,50	162,40	156,30
proteins	181,10	97,00	37,70	64,30	846,60	599,90	272,00	868,00
satellite	6,00	7,00	3,00	4,10	19,10	23,10	6,00	16,30

Abbildung A.9.: Theoriegröße (zusätzliche Datensätze)

	laplace	**mest-abs**	**mest-gain**	**wig**
bongard4	1370,92	2608,46	1267,67	2540,21
proteins	11968,70	7331,49	4515,12	14741,93
satellite	17326,69	12495,03	1186,51	10,49

Abbildung A.10.: Zeitaufwand (zusätzliche Datensätze)

A.4 Weitere Grafiken

In Abschnitt 4.2.2 wurde nur die Genauigkeit auf den Testdaten grafisch dargestellt, Grafik A.15 bis A.36 zeigen die entsprechenden Werte auf den Trainingsdaten. A.11 bis A.14 und A.37 bis A.44 zeigen die Datensätze *mesh-b*, *mesh-c*, *mesh-d* und *mesh-d*, welche ebenfalls ausgelassen wurden.

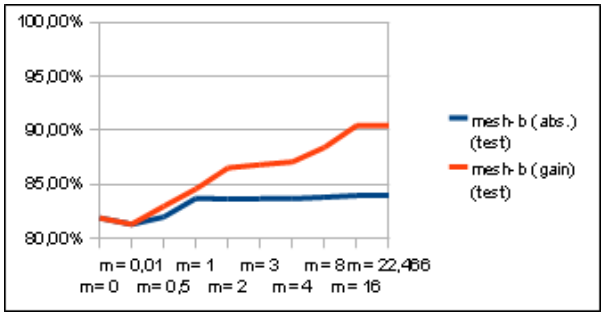

Abbildung A.11.: Genauigkeit auf den Validierungsdaten (mesh-b)

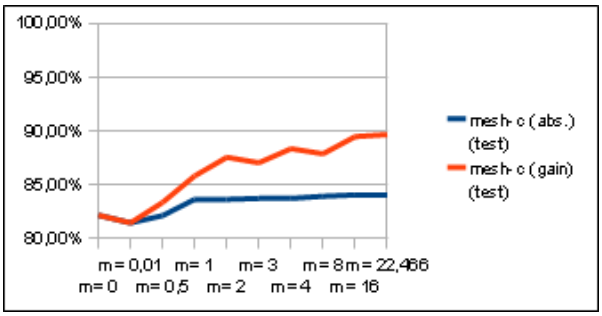

Abbildung A.12.: Genauigkeit auf den Validierungsdaten (mesh-c)

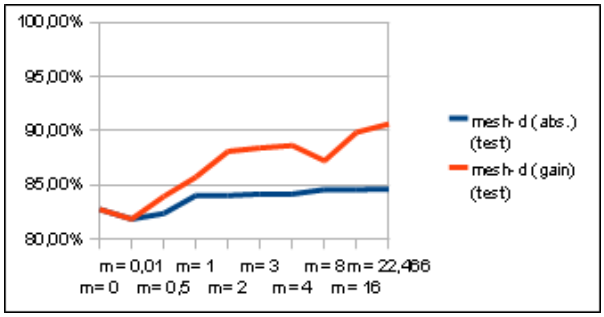

Abbildung A.13.: Genauigkeit auf den Validierungsdaten (mesh-d)

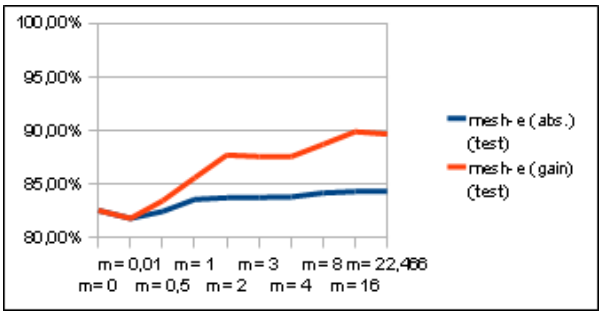

Abbildung A.14.: Genauigkeit auf den Validierungsdaten (mesh-e)

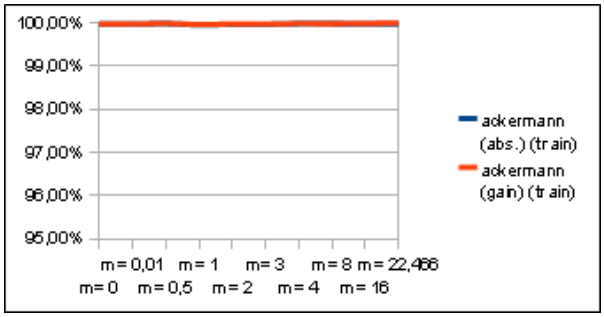

Abbildung A.15.: Genauigkeit auf den Trainingsdaten (ackermann)

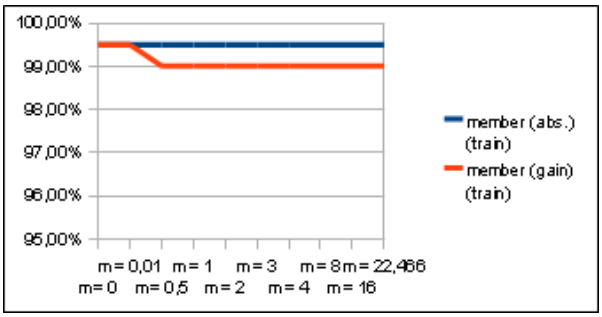

Abbildung A.16.: Genauigkeit auf den Trainingsdaten (member)

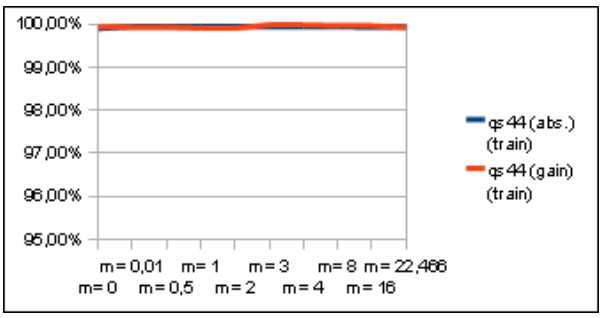

Abbildung A.17.: Genauigkeit auf den Trainingsdaten (qs44)

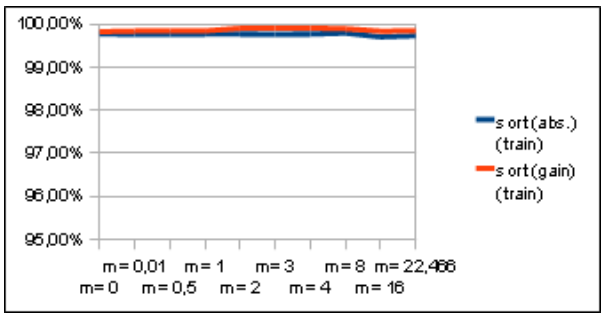

Abbildung A.18.: Genauigkeit auf den Trainingsdaten (sort)

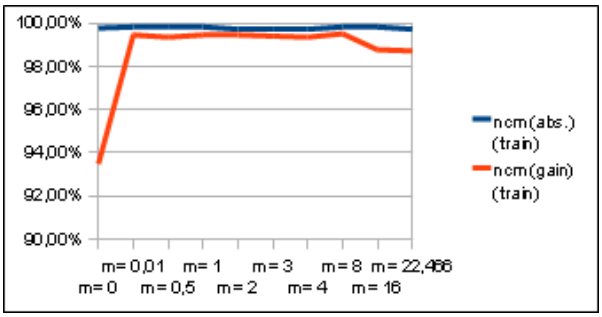

Abbildung A.19.: Genauigkeit auf den Trainingsdaten (ncm)

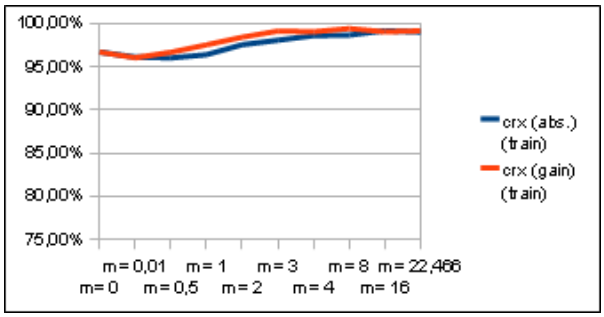

Abbildung A.20.: Genauigkeit auf den Trainingsdaten (crx)

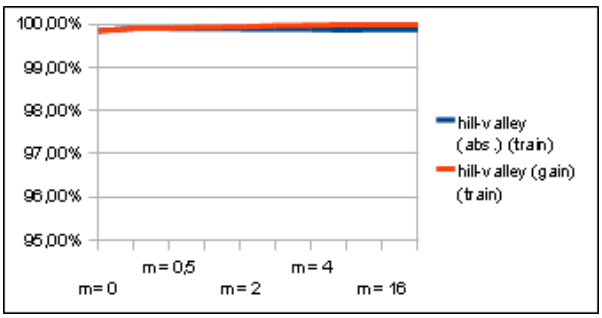

Abbildung A.21.: Genauigkeit auf den Trainingsdaten (hill-valley)

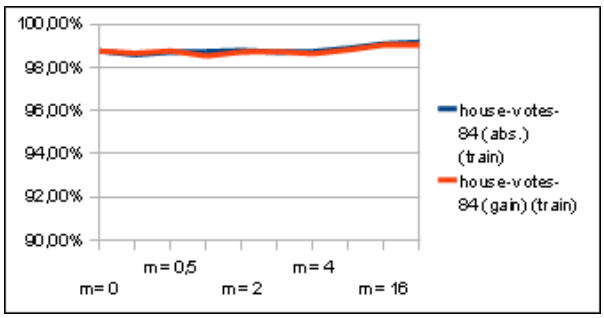

Abbildung A.22.: Genauigkeit auf den Trainingsdaten (house-votes-84)

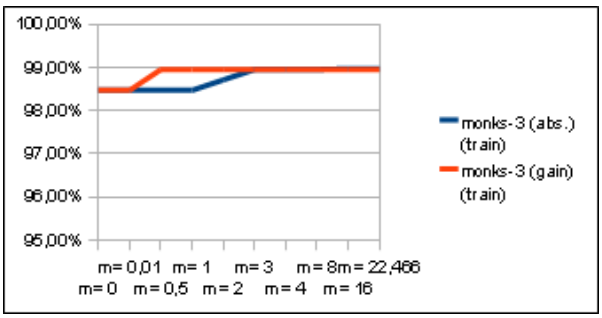

Abbildung A.23.: Genauigkeit auf den Trainingsdaten (monks-3)

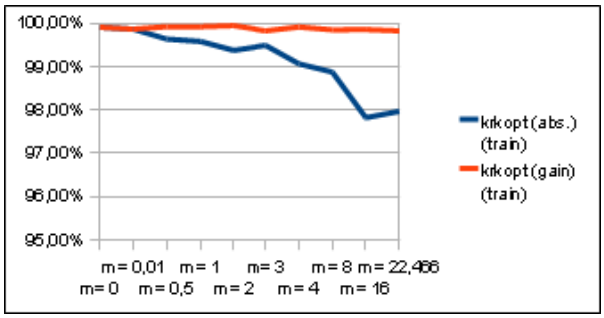

Abbildung A.24.: Genauigkeit auf den Trainingsdaten (krkopt)

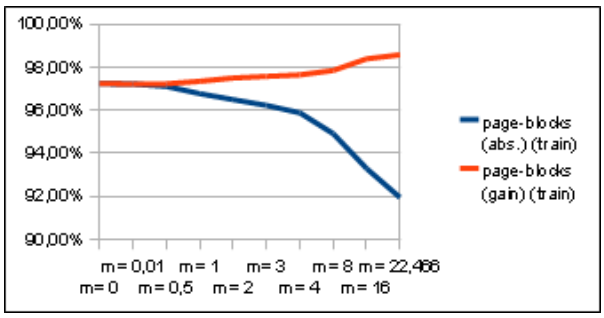

Abbildung A.25.: Genauigkeit auf den Trainingsdaten (page-blocks)

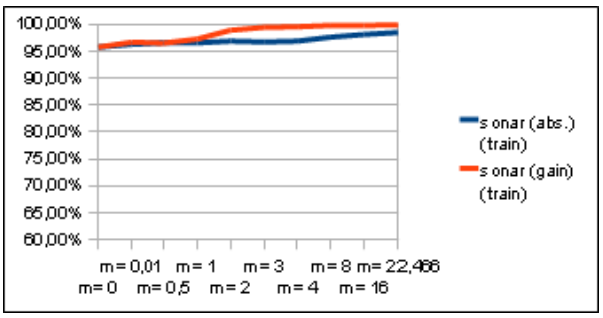

Abbildung A.26.: Genauigkeit auf den Trainingsdaten (sonar)

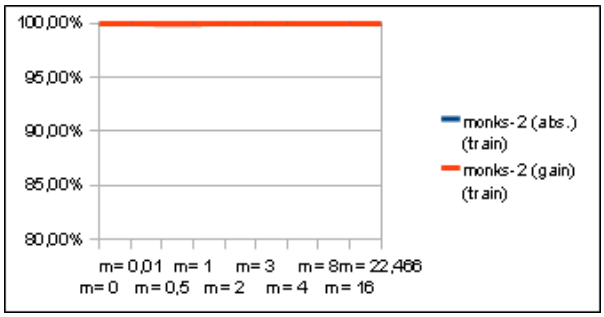

Abbildung A.27.: Genauigkeit auf den Trainingsdaten (monks-2)

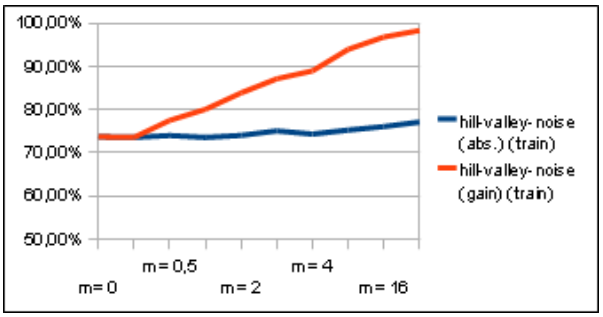

Abbildung A.28.: Genauigkeit auf den Trainingsdaten (hill-valley-noise)

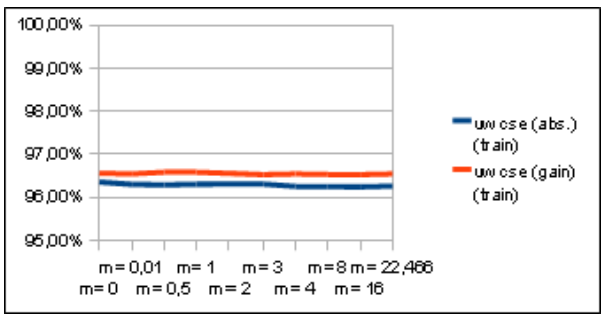

Abbildung A.29.: Genauigkeit auf den Trainingsdaten (uwcse)

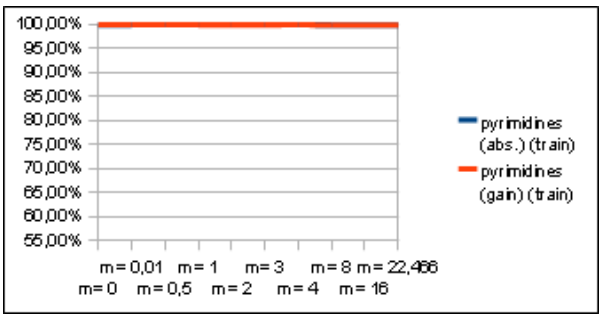

Abbildung A.30.: Genauigkeit auf den Trainingsdaten (pyrimidines)

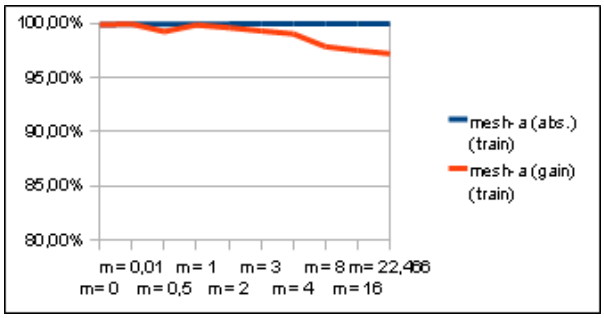

Abbildung A.31.: Genauigkeit auf den Trainingsdaten (mesh-a)

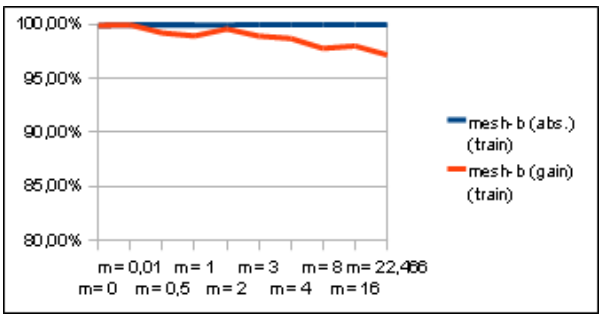

Abbildung A.32.: Genauigkeit auf den Trainingsdaten (mesh-b)

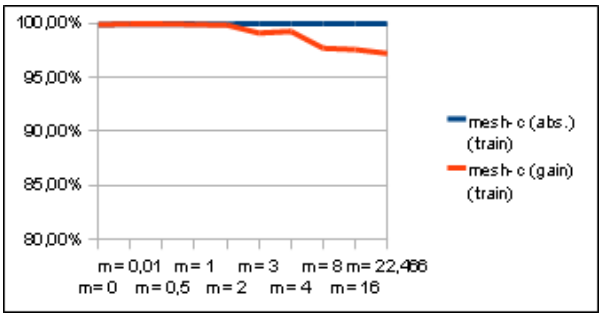

Abbildung A.33.: Genauigkeit auf den Trainingsdaten (mesh-c)

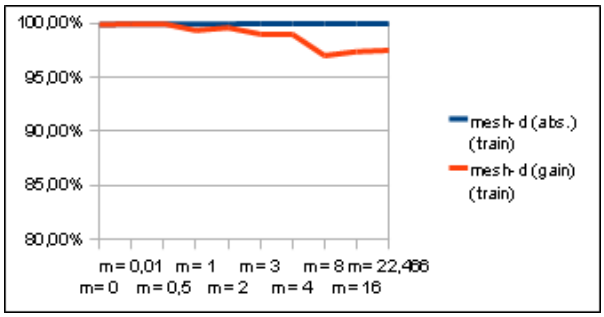

Abbildung A.34.: Genauigkeit auf den Trainingsdaten (mesh-d)

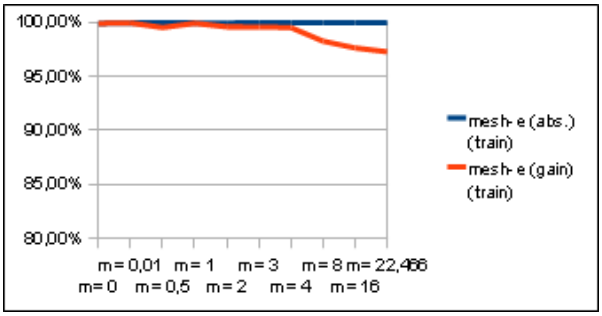

Abbildung A.35.: Genauigkeit auf den Trainingsdaten (mesh-e)

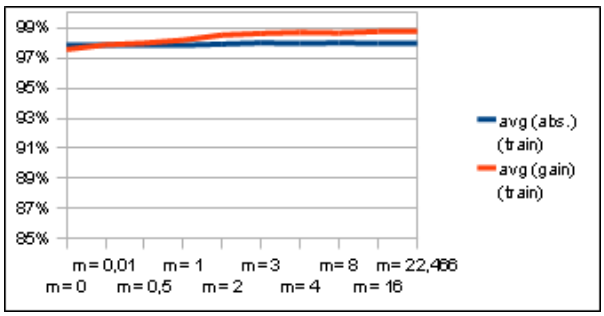

Abbildung A.36.: Genauigkeit auf den Trainingsdaten (Durchschnitt)

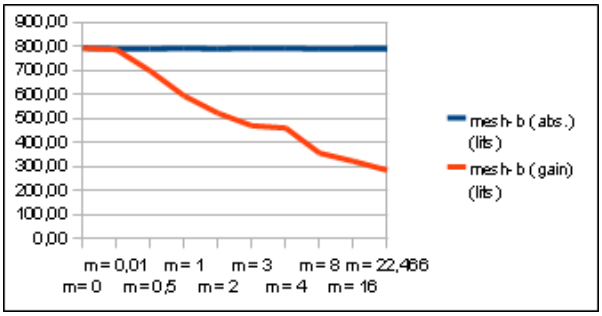

Abbildung A.37.: Literalanzahl der Theorie (mesh-b)

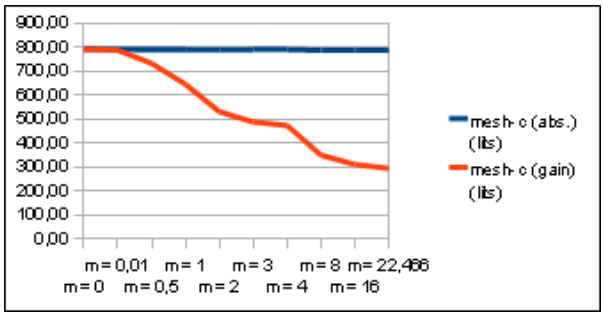

Abbildung A.38.: Literalanzahl der Theorie (mesh-c)

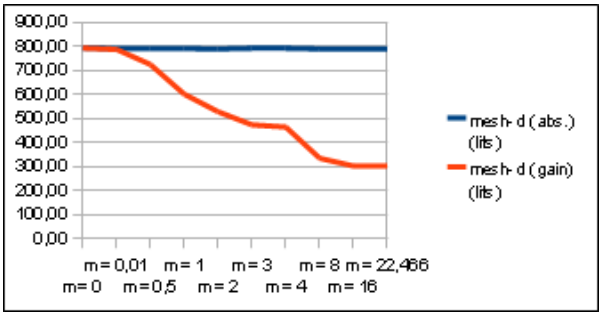

Abbildung A.39.: Literalanzahl der Theorie (mesh-d)

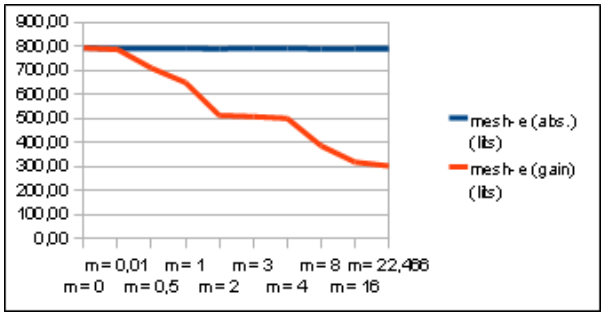

Abbildung A.40.: Literalanzahl der Theorie (mesh-e)

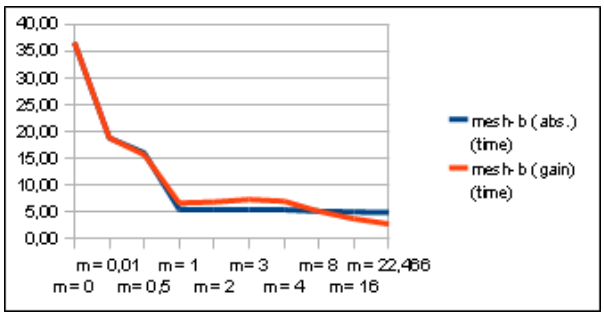

Abbildung A.41.: Zeitaufwand (mesh-b)

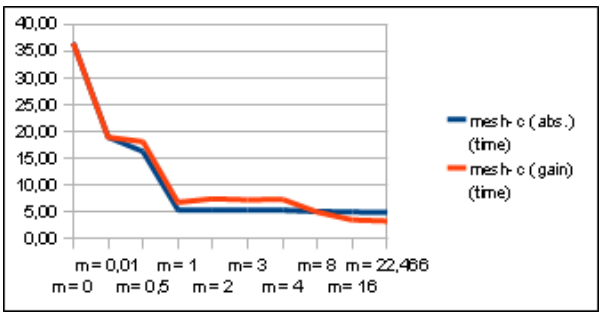

Abbildung A.42.: Zeitaufwand (mesh-c)

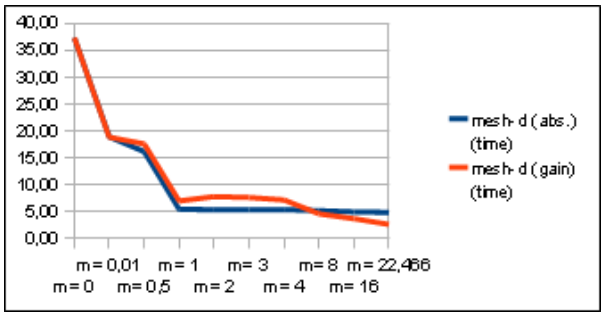

Abbildung A.43.: Zeitaufwand (mesh-d)

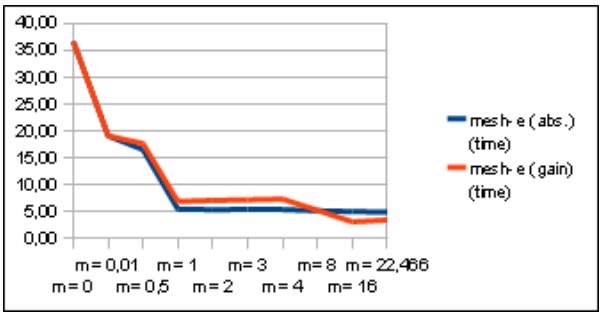

Abbildung A.44.: Zeitaufwand (mesh-e)