Untere Schranken für heuristische Algorithmen[1]

Christoph Berkholz[2]

Abstract: Dieser Beitrag ist eine deutschsprachige Zusammenfassung der Dissertation des Autors. In der Dissertation werden drei verwandte heuristische Verfahren zum Lösen schwerer Probleme untersucht: der k-Konsistenztest für das Constraint-Satisfaction-Problem, Resolution beschränkter Weite für 3-SAT und der Knotenpartitionierungsalgorithmus für das Graphisomorphieproblem. Die Hauptergebnisse der Dissertation sind untere Schranken an die Zeitkomplexität der Verfahren. In diesem Beitrag werden die untersuchten Verfahren eingeführt und die erzielten unteren Schranken vorgestellt.

1 Einführung

In allgemeinen Suchproblemen, sogenannten Constraint-Satisfaction-Problemen, soll eine Menge von Variablen Werte erhalten, sodass alle „Constraints“ erfüllt sind. Die meisten Probleme in NP lassen sich auf natürliche Weise in diesem Rahmen beschreiben. Ein Beispiel ist das Erfüllbarkeitsproblem für aussagenlogische Formeln in konjunktiver Normalform (3-SAT); hier sind die Klauseln die Constraints, welche die Belegung der Booleschen Variablen einschränken. Auch logische Puzzle wie Sudoku passen in diesen Rahmen. Hier sind die Variablen die leeren Kästchen, welchen Werte von 1 bis 9 zugewiesen werden sollen. Die Constraints sind hierbei, dass in jeder Zeile, jeder Spalte und in jedem Block keine Zahl doppelt vorkommt. Solche Suchprobleme lassen sich im Allgemeinen nicht in Polynomialzeit lösen. Grund dafür ist, dass die Anzahl der möglichen Lösungen exponentiell in der Eingabegröße ist. Aus theoretischer Sicht impliziert die etablierte komplexitätstheoretische Annahme $\mathsf{P} \neq \mathsf{NP}$, dass es keine exakten Polynomialzeitalgorithmen für das Constraint-Satisfaction-Problem gibt.

Da das CSP und der prominente Spezialfall 3-SAT von hoher praktischer Relevanz sind, ist es dennoch nötig möglichst effiziente Algorithmen zu entwickeln. Aus dieser Einsicht heraus beschäftigen sich ganze Forschungsfelder, *Constraint Programming* und *SAT-Solving*, mit dem effizienten Lösen des CSPs und des 3-SAT Problems.

Um den Suchraum zu verkleinern, werden in den Algorithmen häufig heuristische Verfahren verwendet. Das einfachste und zugleich am weitesten verbreitete Verfahren ist, lokal inkonsistente Belegungen auszuschließen. Hierbei werden iterativ Mengen von k Variablen betrachtet, um inkonsistente Belegungen aufzuspüren. Dieses generische Verfahren wird für kleine Werte von k intensiv in praktischen Algorithmen für solche Suchprobleme eingesetzt, beispielsweise als *Arc Consistency Test* für allgemeine Constraint-Netzwerke, als *Unit Propagation* in SAT-Solvern, oder als *Color Refinement* in Algorithmen für das

[1] Englischer Originaltitel der Dissertation [Be14a]: „Lower Bounds for Heuristic Algorithms“.

[2] KTH Royal Institute of Technology, CSC, 10044 Stockholm, Sweden, berkho@kth.se.

Graphisomorphieproblem. Wenn die Anzahl k der lokal betrachteten Variablen größer gewählt wird, werden diese Heuristiken wesentlich stärker. Allerdings erhöht sich damit auch die Laufzeit, welche meist die Form $n^{O(k)}$ hat. Dies bedeutet, dass für jedes feste k der Konsistenztest zwar in Polynomialzeit liegt, der Grad des Polynoms allerdings linear in k wächst. Das wirft die Frage auf, ob dieser (aus praktischer Sicht) dramatische Anstieg der Laufzeit vermeidbar und der Konsistenztest beispielsweise in $O(2^k n)$ oder wenigstens $O(n^{\sqrt{k}})$ durchgeführt werden kann. Die Hauptergebnisse der Dissertation schließen diese Möglichkeit aus. Sowohl für binäre Constraint-Netzwerke als auch für 3-SAT Formeln ist es nicht möglich, den Konsistenztest in $O(n^{\varepsilon k})$, für ein absolutes $\varepsilon > 0$, zu implementieren. Diese unteren Schranken an die Zeitkomplexität der Entscheidungsprobleme gelten für allgemeine Berechnungsmodelle (etwa Mehrband-Turingmaschinen) und kommen ohne komplexitätstheoretische Annahmen aus. Dies ist insofern überraschend, als es kaum natürliche Entscheidungsprobleme gibt, für die explizite untere Schranken an die Laufzeit bewiesen werden können.

Ein weiterer Beitrag der Dissertation sind untere Schranken in eingeschränkten Berechnungsmodellen. Hier werden zwei algorithmische Techniken untersucht. Zum einen *Constraint-Propagierung*, welches das gängige Verfahren ist, um k-Konsistenz-Algorithmen für das CSP zu implementieren, und zum anderen *Partitionsverfeinerung*, die dem Knotenpartitionierungsalgorithmus für das Graphisomorphieproblem zu Grunde liegt. Für solche eingeschränkten Berechnungsmodelle können wir noch stärkere untere Schranken beweisen. In ihrer Aussagekraft sind diese etwa vergleichbar mit der $\Omega(n \log n)$ unteren Schranke für vergleichsbasierte Sortieralgorithmen.

In den nächsten drei Abschnitten werden die untersuchten heuristischen Verfahren für CSP, 3-SAT und Graphisomorphie vorgestellt und die erzielten unteren Schranken diskutiert.

2 Lokale Konsistenz von Constraint-Netzwerken

Eine Instanz des CSPs besteht aus einer Menge von Variablen X, einem Wertebereich D und einer Menge C von Constraints. Ein Constraint hat die Form $((x_1,\ldots,x_r),R)$, wobei $(x_1,\ldots,x_r)$ ein r-Tupel von Variablen und R eine r-stellige Relation über dem Wertebereich ist. Die Relation R beschreibt hierbei die Menge aller zulässigen Wertekombinationen für die Variablen $x_1,\ldots,x_r$. Ziel ist es nun eine Belegung der Variablen $\alpha\colon X \to D$ zu finden, die alle Constraints erfüllt. Das heißt, $(\alpha(x_1),\ldots,\alpha(x_r)) \in R$ für alle Constraints $((x_1,\ldots,x_r),R) \in C$. Viele Entscheidungsprobleme in NP lassen sich auf natürliche Weise als CSP formulieren. Ein Beispiel ist zu entscheiden, ob ein gegebener Graph $G = (V,E)$ 3-färbbar ist. Die entsprechende CSP-Instanz enthält eine Variable für jeden Knoten ($X := V$), welche als Werte eine der drei Farben annehmen kann ($D := \{1,2,3\}$). Weiterhin gibt es für jede Kante $\{v_1,v_2\} \in E$ ein Constraint $((v_1,v_2),\{(1,2),(1,3),(2,1),(2,3),(3,1),(3,2)\})$, welches besagt, dass zwei benachbarte Knoten mit unterschiedlichen Farben gefärbt werden müssen.

Der *k-Konsistenztest* [Fr78] ist eine Heuristik zum Einschränken des Suchraumes. Dies wird dadurch erreicht, dass iterativ die möglichen Belegungen der Variablen eingeschränkt

werden. Zunächst einmal ist jede partielle Belegung von Variablen, welche keine Constraints verletzt, lokal *konsistent*. Anschließend werden wiederholt konsistente Belegungen von $\ell < k$ Variablen betrachtet. Wenn sich eine solche Belegung nicht zu einer konsistenten Belegung von $\ell + 1$ Variablen erweitern lässt, kann sie auch nicht Teil einer globalen erfüllenden Belegung sein und wird daher als inkonsistent markiert. Dieser Prozess wird so lange wiederholt, bis keine neuen Belegungen inkonsistent werden. Wenn am Ende des Verfahrens noch konsistente Belegungen der Variablen möglich sind, dann *besteht* die CSP-Instanz den k-Konsistenztest. Im Folgenden wird der Algorithmus in Pseudocode angegeben, hierbei bezeichnet $\mathscr{H}$ die Menge der konsistenten Belegungen.

Algorithm 1 k-Konsistenztest

Eingabe: Eine CSP-Instanz (X, D, C).
$\mathscr{H} \leftarrow$ Menge aller Belegungen von $\leq k$ Variablen, die keine Constraints verletzen.
repeat
  if es gibt $h \in \mathscr{H}$, $|h| < k$, $x \in X$, sodass $h \cup \{x \mapsto a\} \notin \mathscr{H}$ für alle $a \in D$ **then**
    Entferne h und alle Erweiterungen $g \supseteq h$ aus $\mathscr{H}$.
until $\mathscr{H}$ bleibt unverändert.
if $\mathscr{H} \neq \emptyset$ **then** akzeptiere.
else verwerfe.

Da höchstens $|X|^{k-1}|D|^{k-1}$ partielle Belegungen als inkonsistent markiert werden, kann der Algorithmus in Polynomialzeit implementiert werden. Der beste bekannte Algorithmus [Co89] hat eine Laufzeit von $O(n^{2k})$, wobei n die Größe der Eingabe bezeichnet. Die Dissertation beschäftigt sich nun mit der Frage, ob der k-Konsistenztest schneller implementiert werden kann.

2.1 Untere Schranken für Konsistenztests

Das Hauptergebnis ist, dass es prinzipiell unmöglich ist (wesentlich) schneller zu entscheiden, ob eine gegebene CSP-Instanz den k-Konsistenztest besteht.

Satz 1 ([Be12a])**.** *Für jedes k ist es nicht möglich in Zeit $O(n^{\frac{k-3}{12}})$ zu entscheiden, ob eine gegebene CSP-Instanz der Größe n den k-Konsistenztest besteht.*

Diese untere Schranke gilt für deterministische Mehrband Turing-Maschinen und beruht nicht auf komplexitätstheoretischen Annahmen. Dass es Entscheidungsprobleme in P gibt, welche für eine feste Konstante c nicht schneller als in Zeit $O(n^c)$ berechenbar sind, geht aus dem deterministischen Zeithierarchiesatz hervor und kann mit einem Diagonalisierungsargument gezeigt werden. So kann man beispielsweise nicht in Zeit $O(n^c)$ entscheiden, ob eine gegebene Turing-Maschine nach n^{c+1} Schritten hält. Weitere Probleme, für die man solche unteren Schranken beweisen kann, sind meist ähnlich beschaffen. Anwendung auf „natürliche“ Entscheidungsprobleme hat diese Methode bisher nicht direkt ermöglicht. Satz 1 zeigt nun, dass solche unteren Schranken auch für den aus der Praxis

motivierten Konsistenztest gelten. Um diesen Satz zu beweisen, wird in der Dissertation eine Spielcharakterisierung des Konsistenztests, das *existentielle Pebblespiel*, verwendet. Der Zusammenhang zu diesem logischen Pebblespiel, welches die Ausdrucksstärke des existentiell-positiven Fragments der Logik erster Stufe mit beschränkter Variablenzahl charakterisiert, wurde von Kolaitis und Vardi aufgedeckt [KV00].

Dieses Spiel wird von zwei Spielern, Spoiler und Duplicator, auf einer CSP-Instanz gespielt. Die Konfigurationen des Spiels sind partielle Belegungen $h\colon X \to D$ von höchstens k Variablen. Ausgehend von der leeren Belegung $h = \emptyset$ werden in jeder Runde die folgenden Schritte wiederholt.

- Falls $|h| = k$, löscht Spoiler Belegungen aus h, sodass $|h| < k$.
- Spoiler fragt nach der Belegung einer Variablen $x \in X$.
- Duplicator antwortet mit einem Wert $a \in D$.
- Die neue Konfiguration ist $h := h \cup \{(x \mapsto a)\}$.

Das Spiel dauert solange, bis die aktuelle Belegung h ein Constraint verletzt. Falls dies irgendwann geschieht, gewinnt Spoiler das Spiel. Wenn Duplicator so spielen kann, dass dies niemals geschieht, hat sie eine *Gewinnstrategie* für das Spiel. Der Zusammenhang zum k-Konsistenztest ist nun, dass Duplicator genau dann eine Gewinnstrategie für das Spiel hat, wenn die CSP-Instanz den k-Konsistenztest besteht.

Um Satz 1 zu beweisen, wird in der Dissertation eine binäre[3] CSP-Instanz konstruiert, auf der es schwer ist zu entscheiden, ob Duplicator eine Gewinnstrategie besitzt. Diese Konstruktion ist sehr komplex und besteht aus mehreren Bausteinen (*Gadgets*). Um die Existenz von Gewinnstrategien auf dieser Konstruktion zu beweisen, ist es nötig die Strategien modular zu zerlegen. Daher werden zunächst Methoden entwickelt, um partielle Strategien auf den Gadgets zu globalen Strategien zusammenzusetzen. Unter anderem werden *kritische Strategien* eingeführt, welche Duplicator erlauben den Spielfluss zu kontrollieren. Mit Hilfe dieser Methoden wird anschließend gezeigt, dass beide Spieler auf der konstruierten CSP-Instanz Turing-Maschinen mit Laufzeit n^k simulieren können. Der Beweis nutzt dabei aus, dass deterministische Turing-Maschinen mit Laufzeit $O(n^k)$ von alternierenden Turing-Maschinen mit Platz $O(k \log n)$ simuliert werden können [CKS81]. Weiterhin können diese alternierenden Turing-Maschinen als Zwei-Personen-Spiel aufgefasst werden [AIK84], in welchem der erste Spieler die existentiellen und der zweite Spieler die universellen nichtdeterministischen Schritte simuliert. Dieses alternierende Zwei-Personen-Spiel wird nun von Spoiler und Duplicator auf der konstruierten CSP-Instanz nachgeahmt.

[3] Eine CSP-Instanz ist *binär*, wenn jedes Constraint auf zwei Variablen definiert ist. Ein Beispiel ist die eingangs beschriebene CSP-Instanz für 3-Färbbarkeit.

2.2 Untere Schranken für Constraint-Propagation

Als *Constraint-Propagation* wird in der künstlichen Intelligenz das Vorgehen bezeichnet neue Constraints abzuleiten, welche aus den bereits gegebenen folgen. Der k-Konsistenztest, für $k = 2$ auch als Kantenkonsistenz und für $k = 3$ als Pfadkonsistenz bekannt, gilt als der Prototyp für solche Verfahren. Hier besteht ein Ableitungsschritt daraus, eine neue inkonsistente Belegung abzuleiten, welche sich – gemäß der oben beschriebenen Regel – nicht zu einer konsistenten Belegung erweitern lässt. Alle bekannten Algorithmen, insbesondere die zahlreichen Algorithmen für Knoten- und Pfadkonsistenz, folgen diesem Ableitungsprozess. Sie unterscheiden sich im Wesentlichen darin, welche Datenstruktur sie verwenden und in welcher Reihenfolge die Ableitungsschritte durchgeführt werden. Auch bestehende Ansätze zur Parallelisierung des k-Konsistenztests folgen dem Constraint-Propagation Paradigma und führen die Ableitungsschritte parallel aus. Einen gute Einführung in diese Thematik gibt der Übersichtsartikel von Bessiere im *Handbook of Constraint Programming* [Be06].

Einen negativen Effekt auf die Laufzeit von sequentiellen und parallelen Constraint-Propagation Algorithmen haben lange Ketten von sequentiell abhängigen Ableitungsschritten, in denen Belegungen erst inkonsistent werden, wenn vorhergehende Belegungen als inkonsistent markiert wurden. Das wirft die Frage auf, wie groß die Anzahl der sequentiell abhängigen Ableitungsschritte höchstens werden kann. Eine triviale obere Schranke ist hier $|X|^{k-1}|D|^{k-1}$, die Anzahl aller möglichen partiellen Belegungen. Das nächste Ergebnis zeigt, dass im schlimmsten Fall fast alle Ableitungsschritte nacheinander ausgeführt werden müssen.

Satz 2 ([Be14b])**.** *Für jedes $k \geq 2$ gibt es CSP-Instanzen (X,D,C), welche $\Omega(|X|^{k-1}|D|^{k-1})$ sequentielle Ableitungsschritte benötigen, bevor der k-Konsistenztest fehlschlägt.*

Dieser Satz liefert eine untere Schranke an die Laufzeit aller gängigen parallelen und sequentiellen k-Konsistenzalgorithmen und trifft insbesondere auch Aussagen zu den praktisch relavanten Fällen $k = 2$ und $k = 3$. Auf der anderen Seite gilt diese untere Schranke nur in einem eingeschränkten Berechnungsmodell und setzt (im Gegensatz zu Satz 1) voraus, dass die Algorithmen das Constraint-Propagation Schema umsetzen. Der Beweis des Satzes beruht wiederum auf der oben erwähnten Spielcharakterisierung. Der Zusammenhang besteht darin, dass die minimale Anzahl der sequentiellen Ableitungsschritte der minimalen Anzahl von Runden entspricht, die Spoiler benötigt, um das existentielle k-Pebblespiel zu gewinnen. Auch in diesem Fall wird eine recht komplexe CSP-Instanz konstruiert, auf der Spoiler zwar in jedem Fall gewinnt, Duplicator aber die Möglichkeit hat Spoilers Sieg $\Omega(|X|^{k-1}|D|^{k-1})$ Runden hinauszuzögern. Um Duplicators Strategie zu entwerfen, werden auch hier die bereits skizzierten modularen Zerlegungen in kritische Strategien verwendet.

3 Resolutionswiderlegungen kleiner Weite

Im 3-SAT Problem ist die Eingabe eine Menge disjunktiver Klauseln, von denen jede höchstens drei Literale enthält. Dieses klassische Problem war eines der ersten, die als NP-vollständig klassifiziert wurden, und nimmt eine herausragende Stellung in der Informatik ein. Wie eingangs erwähnt, ist das 3-SAT Problem ein spezielles Constraint-Satisfaction-Problem. Hierbei entspricht jede Klausel einem dreistelligem Constraint, das als Belegungen der zugrunde liegenden Variablen nur die erfüllenden Belegungen der Klausel zulässt. Eine grundlegende Methode zum Testen der Erfüllbarkeit einer Klauselmenge ist *Resolution*. Die Resolutionsregel erlaubt neue Klauseln nach folgender Vorschrift abzuleiten:

$$\frac{\gamma \cup \{x\} \qquad \delta \cup \{\neg x\}}{\gamma \cup \delta}$$

Es ist einfach zu sehen, dass diese Regel korrekt ist: eine Belegung, die $\gamma \cup \{x\}$ und $\delta \cup \{\neg x\}$ erfüllt, muss auch $\gamma \cup \delta$ erfüllen. Eine *Resolutionswiderlegung* ist eine durch mehrere Anwendungen der Resolutionsregel gewonnene Ableitung der leeren Klausel. Das Suchen nach einer Resolutionswiderlegung bietet eine vollständige Methode zum Testen der Erfüllbarkeit einer Klauselmenge, da eine Klauselmenge genau dann unerfüllbar ist, wenn sie eine Resolutionswiderlegung hat. Viele SAT-Solver beruhen im Kern auf dieser Form des Schließens. Beispielsweise kann die DPLL-Prozedur [DLL62] als Suche nach einer baumartigen Resolutionswiderlegung aufgefasst werden. Auch moderne CDCL SAT-Solver, welche das Prinzip des Klausel-Lernens [BS97, MSS99] benutzen und aktuell zu den schnellsten SAT-Solvern zählen, folgen ebenfalls einer Ableitung im Resolutionskalkül.

Im schlechtesten Fall kann die Länge einer Resolutionswiderlegung exponentiell in der Größe der Klauselmenge wachsen. Dies wurde zuerst von Haken [Ha85] für eine Klauselmenge, die auf dem Schubfachprinzip beruht, nachgewiesen. Eine Heuristik, um Resolutionswiderlegungen in Polynomialzeit zu finden, ist das Suchen nach Widerlegungen beschränkter Weite. Die *Weite* einer Resolutionswiderlegung ist die maximale Anzahl an Literalen, die in einer Klausel in der Ableitung vorkommen. Da es über n Variablen nur $O(n^k)$ verschiedene Klauseln mit höchstens k Literalen gibt, haben Resolutionswiderlegungen konstanter Weite höchstens polynomielle Länge und können mit dynamischer Programmierung in Zeit $O(n^{k+1})$ gefunden werden. Diese Heuristik wurde bereits in den 1970'er Jahren von Galil vorgeschlagen [Ga77] und erlangte erneute Aufmerksamkeit durch die Arbeit von Ben-Sasson und Widgerson [BSW01].

In der Dissertation wird nun die Frage untersucht, wie schwer es ist zu entscheiden, ob eine Resolutionswiderlegung der Weite k existiert. Wie oben beschrieben, kann man dieses Problem in Zeit $O(n^{k+1})$ lösen, indem nacheinander alle möglichen Klauseln der Größe höchstens k abgeleitet werden. Der nächste in der Dissertation bewiesene Satz besagt, dass dieser triviale Ansatz die (nahezu) bestmögliche Laufzeit liefert.

Satz 3 ([Be12b])**.** *Für jedes k ist es nicht möglich in Zeit $O(n^{\frac{k-3}{12}})$ zu entscheiden, ob eine gegebene 3-SAT Formel der Größe n eine Resolutionswiderlegung der Weite k besitzt.*

Der Beweis dieses Satzes stützt sich wiederum auf eine Spielcharakterisierung. In der Beweiskomplexitätstheorie können Beweissysteme oft durch Zwei-Personen-Spiele, sogenannte *prover-adversary games*, charakterisiert werden. Für Resolution wurde ein entsprechendes Spiel von Pudlak [Pu00] eingeführt. Atserias und Dalmau [AD08] haben gezeigt, dass die Variante dieses Spiels für Resolution beschränkter Weite als existentielles Pebblespiel auf der CSP-Kodierung der 3-SAT Formel aufgefasst werden kann. Die Konfigurationen bestehen aus partiellen Belegungen von k booleschen Variablen. In jeder Runde löscht Spoiler ggf. Belegungen und fragt nach der Belegung einer Variablen x, Duplicator antwortet mit $x \mapsto 0$ oder $x \mapsto 1$. Spoiler gewinnt, wenn die aktuelle Belegung eine Klausel falsifiziert. Für den Beweis des Satzes können daher ähnliche Techniken zur Konstruktion schwerer Instanzen und zur Komposition von partiellen Strategien verwendet werden, wie für den k-Konsistenztest auf binären CSP-Instanzen. Eine direkte Reduktion der beiden Resultate aufeinander ist aber nicht möglich. (Dieser Ansatz wurde schon von Hertel und Urquhart verfolgt [HU06], letztlich aber wieder fallengelassen [HU09].) Daher wird in der Dissertation eine 3-SAT Formel konstruiert, die es beiden Spielern erlaubt das alternierende Zwei-Personen-Spiel (und damit alternierende Turing-Maschinen) zu simulieren.

Im Zuge des Beweises von Satz 3 werden zwei weitere Komplexitätsresultate über Resolutionswiderlegungen beschränkter Weite gezeigt. Das erste Resultat betrachtet die Komplexität des Resolutionsweiteproblems, wenn der Parameter k Teil der Eingabe ist, und löst ein offenes Problem von Vardi (siehe [He08]).

Satz 4 ([Be12b])**.** *Gegeben eine 3-SAT Formel α und ein Parameter k. Es ist* EXPTIME*-vollständig zu entscheiden, ob α eine Resolutionswiderlegung der Weite k besitzt.*

Ein zweites Resultat beschäftigt sich mit regulären Resolutionswiderlegungen, welche aus beweistheoretischer Sicht zwischen Resolution und baumartiger Resolution liegen. Unter zu Hilfenahme der von Hertel [He08] eingeführten regulären Variante des existentiellen Pebblespiels wird in der Dissertation mit dem Beweis des folgenden Satzes ein offenes Problem von Urquhart [Ur11] gelöst.

Satz 5 ([Be12b])**.** *Gegeben eine 3-SAT Formel α und ein Parameter k. Es ist* PSPACE*-vollständig zu entscheiden, ob α eine reguläre Resolutionswiderlegung der Weite k besitzt.*

4 Der Partitionierungsalgorithmus für Graphisomorphie

Im dritten Teil der Dissertation wird das Graphisomorphieproblem (GI) betrachtet. Ein Isomorphismus zwischen zwei Graphen $G = (V(G), E(G))$ und $H = (V(H), E(H))$ ist eine bijektive Abbildung $f\colon V(G) \to V(H)$, die Kanten auf Kanten und nicht-Kanten auf nicht-Kanten abbildet. Formal, $\{v,w\} \in E(G) \iff \{f(v), f(w)\} \in E(H)$ für alle $v,w \in V(G)$. Das Graphisomorphieproblem ist nun zu entscheiden, ob es einen Isomorphismus zwischen zwei gegebenen Graphen gibt. Für dieses Problem sind keine Polynomialzeitalgorithmen bekannt und es ist nötig mit heuristischen Ansätzen den Suchraum zu verkleinern.

Eine einfache Methode, die in den frühen 1970'er Jahren aufkam, ist Knotenpartitionierung (engl. *color refinement*). Dieses Verfahren hat sich als sehr nützlich herausgestellt und

wird routinemäßig in vielen GI-Solvern eingesetzt. Ziel des Verfahrens ist es, die Knotenmengen eines Graphen in Klassen ähnlicher Knoten zu unterteilen. Ähnlichkeit bedeutet in diesem Zusammenhang, dass kein Isomorphismus unähnliche Knoten aufeinander abbildet. Der erste Schritt in diesem Verfahren ist es, die Knoten bezüglich der Anzahl ihrer Nachbarn zu unterteilen. Dies ist gerechtfertigt, da kein Isomorphismus Knoten mit verschiedenen Graden aufeinander abbilden kann. Im nächsten Schritt werden die so entstandenen Klassen weiter unterteilt. Die Knoten werden nun nach der Anzahl der Nachbarn in einer anderen Klasse unterschieden. Zwei Knoten vom Grad fünf sind sich beispielsweise unähnlich, wenn der eine Knoten drei Nachbarn vom Grad sieben hat, der andere aber nur zwei. Dieses Verfahren wird solange iteriert, bis sich die Klassen stabilisieren und alle Knoten einer Klasse die gleiche Anzahl an Nachbarn in jeder anderen Klasse haben.

Algorithm 2 Knotenpartitionierung

Eingabe: Ein Graph $G = (V(G), E(G))$.
Partition $\pi \leftarrow \{V(G)\}$.
repeat
  for all $R \in \pi$, $S \in \pi$ **do**
    Sei $S_1, \ldots, S_\ell$ die Unterteilung von S bezüglich der Anzahl von Nachbarn in R.
    $\pi \leftarrow (\pi \setminus S) \cup \{S_1, \ldots, S_\ell\}$.
until π ist stabil.

Der teuerste Schritt im Knotenpartitionierungsalgorithmus ist das Zählen der Nachbarn in R für jeden Knoten aus S, wofür alle Kanten zwischen R und S betrachtet werden müssen. An dieser Stelle kann der Algorithmus durch eine geschickte Auswahl der zu verfeinernden Klassen optimiert werden. Cardon und Crochemore [CC82] haben gezeigt, dass damit eine Laufzeit von $O(n \log n)$ erreicht werden kann (hierbei wird mit $n = |V(G)| + |E(G)|$ die Größe der Eingabe bezeichnet). Die verwendete Verfeinerungsstrategie folgt dabei im Wesentlichen dem von Hopcroft entwickelten Algorithmus zum Minimieren endlicher Automaten [Ho71].

In der Dissertation wird nun der Frage nachgegangen, ob die Laufzeit mit einer noch geschickteren Auswahl der zu verfeinernden Klassen weiter verbessert werden kann. Die Antwort ist auch hier wieder negativ. Satz 6 besagt, dass selbst bei einer nichtdeterministischen optimalen Auswahl an Klassen die Zeitkomplexität des Algorithmus nicht verbessert werden kann. Die untere Schranke gilt allerdings nicht allgemein für Turing-Maschinen sondern nur unter der Annahme, dass der Algorithmus iterativ Klassen miteinander verfeinert und dafür alle Kanten zwischen den beiden Klassen betrachtet werden müssen.

Satz 6 ([BBG13])**.** *Es gibt eine Familie von Graphen, auf der jede Verfeinerungsstrategie* $\Omega(n \log n)$ *Berechnungsschritte benötigt.*

5 Fazit

In der Dissertation wurden Techniken entwickelt und angewandt, um untere Schranken für heuristische Algorithmen zu beweisen. Der Fokus lag dabei auf Heuristiken für CSP, 3-SAT und Graphisomorphie, welche durch lokale Einschränkungen der Variablenbelegung den Suchraum dieser Probleme verkleinern. Als Hauptergebnisse wurden explizite untere Schranken an die Laufzeit von Turing-Maschinen für den k-Konsistenztest (Satz 1) und Resolutionswiderlegungen beschränkter Weite (Satz 3) bewiesen. Es wurden außerdem in eingeschränkteren Berechnungsmodellen scharfe untere Schranken für den k-Konsistenztest (Satz 2) und den Knotenpartitionierungsalgorithmus (Satz 6) erzielt.

Literaturverzeichnis

[AD08] Atserias, Albert; Dalmau, Víctor: A combinatorial characterization of resolution width. J. Comput. Syst. Sci., 74(3):323–334, Mai 2008.

[AIK84] Adachi, Akeo; Iwata, Shigeki; Kasai, Takumi: Some combinatorial game problems require $\Omega(n^k)$ time. J. ACM, 31, 1984.

[BBG13] Berkholz, Christoph; Bonsma, Paul; Grohe, Martin: Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement. In: Proceedings of the 21st European Symposium on Algorithms (ESA'13). S. 145 – 156, 2013.

[Be06] Bessiere, Christian: Chapter 3 - Constraint Propagation. In (Francesca Rossi, Peter van Beek; Walsh, Toby, Hrsg.): Handbook of Constraint Programming, Jgg. 2, S. 29 – 83. 2006.

[Be12a] Berkholz, Christoph: Lower Bounds for Existential Pebble Games and k-Consistency Tests. In: Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science (LICS'12). S. 25–34, 2012.

[Be12b] Berkholz, Christoph: On the Complexity of Finding Narrow Proofs. In: Proceedings of the 53th IEEE Symposium on Foundations of Computer Science (FOCS'12). S. 351–360, 2012.

[Be14a] Berkholz, Christoph: Lower Bounds for Heuristic Algortihms. Dissertation, RWTH Aachen University, 2014.

[Be14b] Berkholz, Christoph: The Propagation Depth of Local Consistency. In: Proceedings of the 20th International Conference on Principles and Practice of Constraint Programming (CP'14). S. 158–173, 2014.

[BS97] Bayardo, Jr., Roberto J.; Schrag, Robert C.: Using CSP Look-back Techniques to Solve Real-world SAT Instances. In: Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Conference on Innovative Applications of Artificial Intelligence (AAAI'97/IAAI'97). S. 203–208, 1997.

[BSW01] Ben-Sasson, Eli; Wigderson, Avi: Short proofs are narrow - resolution made simple. J. ACM, 48(2):149–169, 2001.

[CC82] Cardon, A.; Crochemore, M.: Partitioning a graph in $O(|A|\log_2|V|)$. Theoretical Computer Science, 19(1):85 – 98, 1982.

[CKS81] Chandra, Ashok K.; Kozen, Dexter C.; Stockmeyer, Larry J.: Alternation. J. ACM, 28(1):114 – 133, Januar 1981.

[Co89] Cooper, Martin C.: An optimal k-consistency algorithm. Artificial Intelligence, 41(1):89 – 95, 1989.

[DLL62] Davis, Martin; Logemann, George; Loveland, Donald: A Machine Program for Theorem-proving. Commun. ACM, 5(7):394–397, Juli 1962.

[Fr78] Freuder, Eugene C.: Synthesizing constraint expressions. Commun. ACM, 21:958–966, November 1978.

[Ga77] Galil, Zvi: On Resolution with Clauses of Bounded Size. SIAM Journal on Computing, 6(3):444–459, 1977.

[Ha85] Haken, Armin: The intractability of resolution. Theoretical Computer Science, 39(0):297 – 308, 1985.

[He08] Hertel, Alexander: Applications of Games to Propositional Proof Complexity. Dissertation, University of Toronto, 2008.

[Ho71] Hopcroft, J.E.: An n log n algorithm for minimizing states in a finite automaton. In: Theory of Machines and Computations, S. 189–196. Academic Press, 1971.

[HU06] Hertel, Alex; Urquhart, Alasdair: The Resolution Width Problem is EXPTIME-Complete. Bericht 133, 2006.

[HU09] Hertel, Alex; Urquhart, Alasdair: Comments on ECCC Report TR06-133: The Resolution Width Problem is EXPTIME-Complete. Bericht 003, 2009.

[KV00] Kolaitis, Phokion G.; Vardi, Moshe Y.: A Game-Theoretic Approach to Constraint Satisfaction. In: Proceedings of the 17th National Conference on Artificial Intelligence (AAAI'00). S. 175–181, 2000.

[MSS99] Marques-Silva, J.P.; Sakallah, K.A.: GRASP: a search algorithm for propositional satisfiability. Computers, IEEE Transactions on, 48(5):506–521, 1999.

[Pu00] Pudlak, Pavel: Proofs as Games. The American Mathematical Monthly, 107(6):541–550, 2000.

[Ur11] Urquhart, Alasdair: , Width and size of regular resolution proofs. Talk given at the Banff Proof Complexity Workshop, 2011.

Christoph Berkholz hat im Jahr 2005 sein Abitur am Theodor-Fontane Gymnasium in Strausberg abgelegt. Anschließend studierte er an der Humboldt-Universität zu Berlin den Diplomstudiengang Informatik. Seine Diplomarbeit im Gebiet der Schaltkreiskomplexitätstheorie wurde dort mit dem Institutspreis ausgezeichnet. Nach Abschluss des Studiums im Jahr 2010 arbeitete er zunächst als Doktorand an der Humboldt-Universität und wechselte zwei Jahre später zusammen mit seinem Doktorvater Martin Grohe an die RWTH Aachen, wo er im Dezember 2014 seine Dissertation verteidigte. Zur Zeit forscht er im Rahmen eines eingeworbenen Postdoktorandenstipendiums des DAAD an der KTH Stockholm im Bereich der Beweiskomplexitätstheorie.