Leibniz Universität Hannover
Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Beweis und Bedeutung der Sensitivity Conjecture in der Komplexitätstheorie

Mathis Kruse

Erstprüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Maurice Chandoo
Betreuer: M.Sc. Timon Barlag

Hannover, den 23. August 2020.

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 23. August 2020

_____

Vorname Nachname

Inhaltsverzeichnis

1 Einleitung und Motivation

Eine Boole'sche Funktion beschreibt auf den ersten Blick nur eine Menge an Nullen und Einsen, genannt Bits, welche in ihrer Gesamtheit entweder auf die Null oder die Eins abbilden. In mathematischer Notation ließe sie sich für eine Eingabemenge mit n Bits als

$$f : \{0,1\}^n \to \{0,1\}$$

niederschreiben. Dass sich die Studie dieser Funktionen durchaus auch lohnen kann ist ersichtlich, wenn man sich klar macht, wie weitreichend ihr Einfluss ist. Bricht man einen Computer auf das niedrigste Abstraktionsniveau herunter, so handelt es sich nur noch um ein Zusammenspiel von Boole'schen Funktionen und deren Berechnung.

Demnach ist es auch lohnenswert sich tiefgründig mit der Komplexität dieser Funktionen zu beschäftigen. Innerhalb von so gut wie jedem Algorithmus sind lauter Berechnungen Boole'scher Funktionen enthalten. Dies schließt zum Beispiel alle möglichen Arten von auszuwertenden logischen Ausdrücken mit ein. Aber auch andere Teilgebiete der Informatik, wie zum Beispiel die Digitaltechnik, sind mit ihnen durchdrungen. Dementsprechend kommt man als Informatiker um den Begriff der Boole'schen Funktion nur sehr schwer herum.

Tiefergehend befasst man sich insbesondere innerhalb der theoretischen Informatik mit den Eigenschaften der Boole'schen Funktion, sei es nur auf dem Papier oder in direkter Verbindung mit einem Rechnermodell.

Zu einem besonderen Ergebnis auf diesem Gebiet kam der Mathematiker Hao Huang im Jahr 2019. Er veröffentlichte einen Beweis der sogenannten Sensitivity Conjecture [9]. Dabei handelte es sich um eine Vermutung, die über 25 Jahre lang nicht bewiesen werden konnte. Diese trifft eine Aussage über das Verhältnis zwischen Sensitivität und Block-Sensitivität einer Boole'schen Funktion.

Die Sensitivität von einer Boole'schen Funktion sagt etwas darüber aus, wie anfällig unsere Funktion darauf ist ihren Wert zu ändern, wenn wir ein Bit aus ihrer Eingabe von Null auf Eins (oder umgekehrt) 'kippen' lassen. Weiter stellt die Block-Sensitivität eine Verallgemeinerung dessen dar, weil sie es auch erlaubt ganze 'Blöcke' von Bits zu kippen.

In Anbetracht der vergeblichen Beweisversuche vieler Jahre ist es umso eindrucksvoller, auf was für eine präzise und elegante Art der erwähnte Beweis von Huang vollzogen wurde. Gerade deshalb, und weil mit ihm viele weitere Folgen für die Komplexität von Boole'schen Funktionen einhergehen, wird er zentrales Stück dieser Arbeit sein. Es wird einerseits darauf eingegangen wie genau der Beweis funktioniert, aber auch welche Komplexitätsmaße mit der (Block-)Sensitivität verwandt sind und welche Folgen für diese aus alledem resultieren.

2 Grundlagen

2.1 Mathematische Grundlagen

Bei Beweisen rund um die Komplexität Boole'scher Funktionen ist es oft nötig sich Erkenntnisse aus vielen mathematischen Gebieten zu Nutze zu machen. Dafür ist die Anwendung von Methoden aus der Kombinatorik, der linearen Algebra und anderen möglichen (mathematischen) Teilbereichen oft sehr wichtig. Gerade für die Sensitivity Conjecture bilden Graphentheorie und algebraische Methoden den Mittelpunkt. Eine wichtige Verbindung der Sensitivität und der Graphentheorie, beleuchtet in Kapitel 3.1, ist sogar elementar um die Conjecture an sich zu beweisen. Die meisten Definitionen entspringen direkt dem Beweis der Sensitivity Conjecture aus [9]. Einige weitere sind sinngemäß [17] entnommen.

Sei $G = (V, E)$ immer ein ungerichteter Graph, welcher durch seine Knotenmenge V und Kantenmenge E definiert ist.

Definition 2.1. *Sei der Graph $H = (V_H, E_H)$ ein* induzierter Subgraph *vom Graphen $G = (V_G, E_G)$. Dafür muss $V_H \subseteq V_G$ sowie $E_H \subseteq E_G$ gelten. Weiter darf eine Kante $(v_1, v_2) \in E_G$ nur in E_H vorkommen, wenn auch $v_1, v_2 \in V_H$.*

Wenn wir einen Subgraphen induzieren wollen, so ist es zum Beispiel auch denkbar den Graphen zu betrachten, der komplementär zum Subgraphen ist. Gewissermaßen teilen wir G in die zwei Subgraphen H und $G - H$ auf.

Definition 2.2. *Sei H ein induzierter Subgraph von G wie in Definition 2.1. Der Graph $G - H$ ist ebenfalls induzierter Subgraph von G. Für seine Knotenmenge V_{G-H} gilt zusätzlich, dass $V_{G-H} = V_G \backslash V_H$ ist.*

Beispiel 2.3. Wir konstruieren zur Veranschaulichung von induzierten Subgraphen einen Graphen G mit der Knotenmenge $V_G = \{A, B, C, D, E, F\}$ wie in Abbildung 2.1. Für unseren Subgraphen H wählen wir die Knotenmenge $V_H \subseteq V_G$ zufällig als $\{A, C, E, F\}$.

Mit der Kenntnis all dieser Knotenmengen ist es ebenfalls leicht den Graphen $G - H$ zu konstruieren. Wir erhalten für diesen die Menge

$$V_{G-H} = V_G \backslash V_H = \{B, D\}.$$

Da zwischen diesen Knoten in G keine Kante existiert, besteht der Graph $G - H$ nur aus den Knoten B und D und keinerlei Kanten.

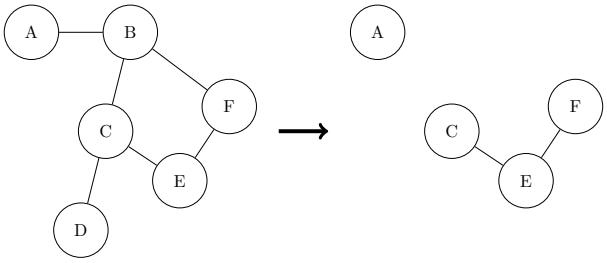

Abbildung 2.1: Der Graph G und der daraus mit $V_H = \{A, C, E, F\}$ induzierte Subgraph H

Definition 2.4. *Sei G ein ungerichteter Graph. $\Delta(G)$ bezeichnet den* maximalen Knotengrad *aller Knoten von G. Dabei ist der Grad eines einzelnen Knotens die Anzahl an Kanten, die mit ihm verbunden sind.*

Definition 2.5. *Die* Adjazenzmatrix *eines Graphen $G = (V, E)$ ist eine Matrix der Größe $|V| \times |V|$ gefüllt mit Einträgen $a_{i,j}$ für die gilt:*

$$a_{i,j} = \begin{cases} 1 & \textit{falls } (v_i, v_j) \in E \\ 0 & \textit{sonst} \end{cases}$$

Die Matrix wird an Stelle $a_{i,j}$, also in i-ter Zeile und j-ter Spalte, eine Eins stehen haben, falls Knoten i und j durch eine Kante verbunden sind. Es lässt sich sehr simpel feststellen, dass eine Adjazenzmatrix immer quadratisch sein muss (sie hat immer die Größe $|V| \times |V|$). Ferner lässt sich feststellen, dass für ungerichtete Graphen die Adjazenzmatrix immer symmetrisch sein muss, da eine Kante (i, j) äquivalent mit der Kante (j, i) ist.

Definition 2.6. *Zu einer quadratischen Matrix A sind die* Eigenvektoren *alle Vektoren $\vec{x} \neq 0$, für die*

$$A\vec{x} = \lambda\vec{x}$$

gilt, wobei λ den zum Eigenvektor dazugehörigen Eigenwert *von A bezeichnet.*

Für komplexe Matrizen gilt zudem, dass jede Matrix der Größe $n \times n$ genau n Eigenwerte besitzt. Es können jedoch Eigenwerte mehrfach auftreten. Dafür wird jedem Eigenwert eine Multiplizität beziehungsweise eine algebraische Vielfachheit zugeordnet. Diese bestimmt, wie oft dieser Eigenwert in dieser Matrix 'auftaucht'.

Definition 2.7. *Sei A eine quadratische Matrix der Dimension n gefüllt mit den Einträgen $a_{i,j}$. Die* Spur *der Matrix A ist definiert als*

$$Spur(A) = \sum_{i=1}^{n} a_{i,i}.$$

Für die Berechnung der *Spur* einer Matrix existiert jedoch noch eine weitere Möglichkeit.

Lemma 2.8. *[17] Sei A eine quadratische Matrix der Dimension n und* $\lambda_1, \ldots, \lambda_n$ *ihre Eigenwerte. Dabei ist es möglich, dass Paare* $i \neq j$ *existieren, sodass* $\lambda_i = \lambda_j$*. Dann ist die* Spur *der Matrix A berechenbar mittels*

$$Spur(A) = \sum_{i=1}^{n} \lambda_i.$$

Das bedeutet, dass die Spur einer Matrix nicht nur die Summe aller Elemente der Hauptdiagonalen ist, sondern auch die Summe aller Eigenwerte. Dabei werden Eigenwerte mit Multiplizitäten höher als Eins mehrmals in die Summe mit einfließen.

Definition 2.9. *Sei G ein ungerichteter Graph.* $\lambda_1(G)$ *bezeichnet den* größten Eigenwert *der Adjazenzmatrix von G.*

Definition 2.10. *Sei A eine* $n \times n$*-Matrix, B eine* $m \times m$*-Matrix und* $m < n$*. Dann ist B eine* quadratische Teilmatrix *von A (engl. principal submatrix), wenn sie aus A durch weglassen von* $n - m$ *Zeilen und Spalten entstanden ist.*

Leicht lässt sich die Defintion der quadratischen Teilmatrizen mit einem Beispiel nachvollziehen.

Beispiel 2.11. Sei $n = 3$ und $m = 2$. Damit ist $m < n$ und $n - m = 1$. Also lassen wir jeweils eine Zeile und eine Spalte von der unten beschriebenen Matrix A weg. Für unser Beispiel erzeugen wir B durch das Weglassen der ersten Zeile und der ersten Spalte.

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, B = \begin{pmatrix} 5 & 6 \\ 8 & 9 \end{pmatrix}$$

Offensichtlich ist B dadurch eine quadratische Teilmatrix von A. Abschließend lassen wir die zweite Zeile und die dritte Spalte von A weg und erhalten die folgende quadratische Teilmatrix C.

$$C = \begin{pmatrix} 1 & 2 \\ 7 & 8 \end{pmatrix}$$

Ein etwas komplizierteres Theorem, welches sich mit den Eigenwerten quadratischer Teilmatrizen beschäftigt, bildet Cauchy's Interlace Theorem. Es findet im Beweis der Sensitivity Conjecture Verwendung, wird hier jedoch nicht weiter hergeleitet.

Satz 2.12. *(Cauchy's Interlace Theorem [5]) Sei A eine symmetrische* $N \times N$*-Matrix und B eine quadratische Teilmatrix von A der Größe* $m \times m$ *für ein* $m < N$*. Sind* $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$ *die Eigenwerte von A und* $\mu_1 \geq \mu_2 \geq ... \geq \mu_m$ *die Eigenwerte von B, dann gilt für alle* $1 \leq i \leq m$*,*

$$\lambda_i \geq \mu_i \geq \lambda_{i+N-m}.$$

2.2 Hypercube

Q^n bezeichnet den Graphen des n-dimensionalen Hypercubes (dt. Hyperwürfel). Der Hypercube ist dabei die Verallgemeinerung eines Würfels auf mehrere Dimensionen. Anwendung findet er vor allem in der Netzwerktechnik oder in parallelen Systemen (siehe auch [15]). Aber auch mathematisch ist der Hypercube sehr interessant. Die Knotenmenge besteht aus allen Vektoren aus $\{0,1\}^n$. Eine Kante wird gezogen, wenn zwei Knoten v_1 und v_2 sich in genau einer Position unterscheiden. Dies ist äquivalent dazu, dass sie eine Hamming-Distanz von 1 besitzen, denn die Hamming-Distanz zählt die Anzahl unterschiedlicher Stellen in Vektoren. Ermitteln lässt sie sich sehr leicht mit Anwenden der XOR-Funktion (zugehöriges Symbol $\oplus$), denn diese Funktion bildet nur auf die 1 ab, wenn sich die zwei Eingabebits unterscheiden.

Definition 2.13. *Sei $x \in \{0,1\}^n$ ein Vektor. Mit dem* Hamming-Gewicht *bezeichnen wir die Anzahl Einsen eines Vektors und schreiben dafür $|x|_{ham}$.*

Seien weiter x_1 und x_2 Vektoren aus $\{0,1\}^n$. Dann bezeichnet HAMM*(x_1,x_2) die* Hamming-Distanz *von x_1 und x_2.*

$$\text{HAMM}(x_1, x_2) = |x_1 \oplus x_2|_{ham}$$

Die Konstruktion des Hypercubes lässt sich insbesondere für niedrigere Dimensionen sehr gut mit Bildern veranschaulichen.

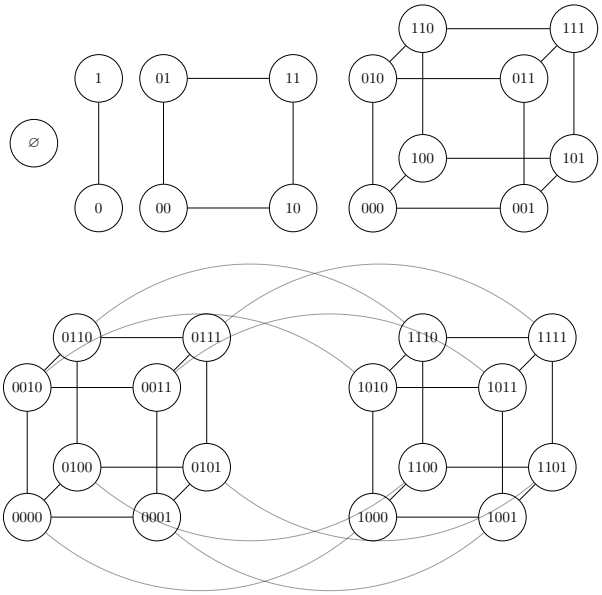

Abbildung 2.2: Konstruktion des Hypercubes Q^i für $0 \leq i \leq 4$

Wir beginnen für unsere Konstruktion in der Dimension Null mit einem einzelnen Knoten, welcher repräsentativ für den leeren Vektor steht. Dieser Graph enthält bisher keinerlei Kanten. Wollen wir den Hypercube der höheren Dimension konstruieren, so müssen wir zuerst gedanklich einen zweiten identischen Hypercube zum vorherigen hinzufügen. Beim Erhöhen der Dimension erhöht sich auch die Dimension aller Knoten, beziehungsweise aller Vektoren, die diese repräsentieren. Alle bereits existierenden Knoten erhalten dabei als Most-Significant-Bit eine Null. Die Knoten aus dem 'neuen' Hypercube erhalten hingegen als Most-Significant-Bit eine Eins.

Alle auf diese weise erzeugten Knoten werden dann wieder nach dem obigen Prinzip verbunden (Kante nur bei Hamming-Distanz von 1). Dazu ist leicht zu erkennen, dass sich mit jeder Dimension die Anzahl der Knoten verdoppelt. Genauer verdeutlicht wird die beschriebene Konstruktionsvorschrift durch Abbildung 2.1. Formal schreiben wir also:

Definition 2.14. *Sei $Q^n = (V, E)$ der* Graph des n-dimensionalen Hypercubes. *Dann ist $V = \{0,1\}^n$. Dazu ist $(v_1, v_2) \in E$ genau dann, wenn. $HAMM(v_1, v_2) = 1$.*

Hiermit können wir auch formale Betrachtungen zum Hypercube durchführen, die in dieser Arbeit noch relevant werden. Weil wir uns viel mit den Eigenschaften der Knotengrade unserer Graphen beschäftigen, benötigen wir ein weiteres Maß. Dieses ist für Subgraphen des Hypercubes definiert und befasst sich mit dem induzierten Subgraphen H und dessen Komplement $Q^n - H$.

Definition 2.15. *Sei Q^n der Graph des n-dimensionalen Hypercubes und H ein induzierter Subgraph vom Q^n. Dann definieren wir*

$$\Gamma(H) = max\{\Delta(H), \Delta(Q^n - H)\}$$

als ein Maß über die Eigenschaften der Subgraphen.

Beispiel 2.16. Sei $Q^2 = (V_{Q^2}, E_{Q^2})$ der Graph des zweidimensionalen Hypercubes. Dabei gilt nach Def. 2.14, dass $V_{Q^2} = \{00, 01, 10, 11\}$. Für den induzierten Subgraph H von Q^2 wählen wir $V_H = \{00, 11\}$. Für den zu H komplementären Graphen $Q^2 - H$ gilt, dass seine Knotenmenge V_{Q^2-H} nach Def. 2.2 festgelegt ist als

$$V_{Q^2-H} = V_{Q^2} \backslash V_H = \{00, 01, 10, 11\} \backslash \{00, 11\} = \{01, 10\}.$$

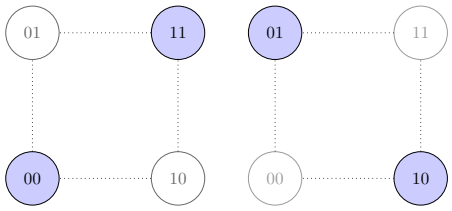

Abbildung 2.3: Zweidimensionales Beispiel für H (links) und $Q^2 - H$ (rechts)

Zu erkennen ist dies in Abbildung 2.3. Insbesondere sehen wir, dass wenn H halb so viele Knoten wie Q^2 besitzt, es möglich ist, dass sowohl H als auch $Q^2 - H$ keine Kanten besitzen. Das bedeutet wiederrum, dass $\Gamma(H) = 0$ ist. Die Konstruktion leerer Subgraphen ist für alle Dimensionen des Hypercubes möglich. Dafür muss lediglich für alle Knoten v_i des Hypercubes gelten, dass $v_i \in V_H$ genau dann wenn $|v_i|_{ham}$ gerade ist.

Speziell in dieser Arbeit wird der Hypercube aufgrund seiner Verwandschaft zur Sensitivity Conjecture wichtig.

2.3 Übersicht Boole'scher Komplexitätsmaße

Für unsere Betrachtungen Boole'scher Komplexitätsmaße bezeichnen wir mit

$$f : \{0,1\}^n \to \{0,1\}$$

eine Boole'sche Funktion.

Dazu sei $x = (x_1 x_2 \dots x_n) \in \{0,1\}^n$ ein binärer Vektor. Mit $f(x)$ bezeichnen wir die Funktion f mit dem Vektor x als Eingabe. Ergänzend bezeichnen wir mit dem Kippen eines Bits den Vorgang, bei dem der Wert eines Bits geändert wird (von Eins auf Null oder umgekehrt). Dies ist nämlich essenziell für Boole'sche Maße rund um die Sensitivität. Passend dazu definieren wir weitere Notationen. Um die einzelnen Bits x_i unseres Vektors x zu adressieren, definieren wir eine Menge von Indizes S, die zu den Bits korrespondieren.

$$S \subseteq [n] = \{1, \dots, n\}$$

Außerdem bezeichnen wir nun mit x^S den Vektor x, bei dem alle Bits x_i gekippt werden, wenn $i \in S$.

Das Hamming-Gewicht $|x|_{ham}$ ist, wie auch in Kapitel 2.2, als Anzahl von Einsen im Vektor x definiert.

Beispiel 2.17. Sei $x = 0101$ und $S = \{2, 4\}$.

Durch das Kippen von Bits erhalten wir so: $x^{\{1\}} = 1101$, $x^{\{4\}} = 0100$ und $x^S = 0000$. Durch das Zählen der Einsen kommen wir auf $|x|_{ham} = 2$ oder $|x^{\{1\}}|_{ham} = 3$.

Im Folgenden werden eine Reihe Boole'scher Komplexitätsmaße eingeführt, zwischen denen im Verlauf der Arbeit verschiedene Verwandtschaften und Ähnlichkeiten erläutert werden. Die Definition dieser erfolgt in Anlehnung an [2] und [10], wobei jedoch teilweise eigene Begrifflichkeiten und Notation verwendet werden.

2.3.1 Decision Tree Complexity

Ein sehr wichtiges Berechnungsmodell für Boole'sche Funktion ist das Modell der Decision Trees (dt. Entscheidungsbäume). Es wird auf eine spezielle Art von Graphen

zugegriffen, um Boole'sche Funktionen auf einer Eingabe auszuwerten, beziehungsweise ihre Komplexität weiter zu charakterisieren. Ein Decision Tree T ist ein gerichteter binärer Baum. Jeder der internen Knoten ist mit einer Variable x_i der zu berechnenden Funktion beschriftet. An den Blättern des Baumes stehen jeweils die Werte 0 oder 1. Für das Auswerten einer Funktion beginnen wir bei der Wurzel des Baumes und halten erst beim Erreichen eines Blattes.

An jedem erreichten Knoten wird der Wert, den unsere Eingabe für x_i vorschreibt abgefragt (man spricht hier von einer 'Query'). Bei einer 0 gehen wir rekursiv in den linken Teilbaum und bei einer 1 in den rechten. Erreichen wir ein Blatt, so haben wir $f(x)$ ausgewertet und erhalten den Wert des Blattes als Ergebnis.

Wir sagen, dass ein Decision Tree T die Funktion f berechnet, wenn für alle möglichen Eingaben x gilt, dass das erreichte Blatt von T, also seine Ausgabe, gleich $f(x)$ ist.

Definition 2.18. *Die* Decision Tree Complexity *D(f) einer Boole'schen Funktion f ist die minimale Tiefe eines Decision Trees, der f berechnet.*

Dies bildet die Definition der deterministischen Decision Tree Complexity. Jedoch ist dies nicht die einzige Möglichkeit um Decision Trees zur Analyse der Komplexität zu nutzen. So gibt es auch Modelle für nichtdeterministische, zufallsorientierte oder sogar quantenmechanische Decision Trees. Diese werden hier nicht weiter erwähnt, sind jedoch zum Beispiel in [2] zu finden.

Beispiel 2.19. Sei nun $n = 3$ die Dimension unserer Funktion f. Sei $f(x) = 1$ immer dann, wenn x die binäre Darstellung einer Zahl größer als 4 ist. Somit ist $f(x) = 1$ für $x \in \{101, 100, 111\}$. Am schlauesten ist es, das Bit mit höchster Wertung zuerst

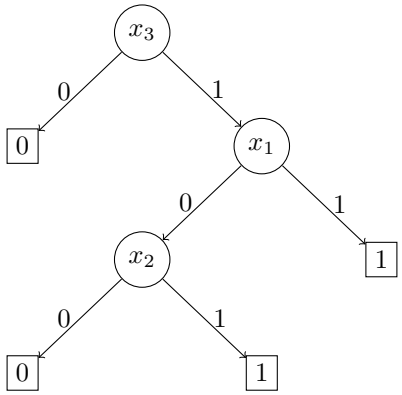

Abbildung 2.4: Decision Tree zu f

anzufragen. Jede 1, die danach als Antwort auf eine Anfrage geliefert wird, setzt danach auch f auf den Wert 1. Dieser spezielle Decision Tree hat eine Tiefe von 3. Tatsächlich ist er sogar minimal, weshalb $D(f) = 3$ für unsere Funktion gilt.

2.3.2 Certificate Complexity

Weiterhin sei f eine Boole'sche Funktion und $x \in \{0,1\}^n$ ein Vektor. Erneut ist $S \subseteq [n]$ eine Menge von Indizes. Eine partielle Belegung $C : S \to \{0,1\}$ ordnet allen Bits x_i mit $i \in S$ jeweils einen Boole'schen Wert zu. Die Größe einer partiellen Belegung ist festgelegt durch $|S|$. Die Belegung C ist konsistent mit Eingabe x, wenn für alle $i \in S$ gilt, dass $C(i) = x_i$.

Wir wollen mit der Certificate Complexity ausdrücken, von welcher Anzahl Bits wir den Wert kennen müssen, um den Wert unserer Funktion f sicher bestimmen zu können. Dabei kann unterschieden werden ob $f(x) = 1$ oder $f(x) = 0$ 'erzwungen' werden soll. Allerdings ist die allgemeine Certificate Complexity unabhängig von einem Ausgabewert definiert. Das heißt sie beschreibt die Anzahl Bits, die nötig sind, um $f(x)$ sicher auf einen beliebigen Wert bestimmen zu können.

Definition 2.20. *Eine partielle Belegung C von Variablen einer n-stelligen Boole'schen Funktion heißt* 1-Zertifikat *(analog* 0-Zertifikat*), wenn für alle mit C konsistenten Belegungen $x \in \{0,1\}^n$ gilt, dass $f(x) = 1$ (analog $f(x) = 0$).*

Beispiel 2.21. Sei $n = 5$. Außerdem sei $f(x) = 1$, genau dann wenn $|x|_{ham} \geq 2$. Wir wählen $x = 11111$ als unsere Eingabe für f. Eine Belegung C_1 die ebenfalls alle fünf Variablen auf 1 setzt, würde $f(x) = 1$ erzwingen und wäre konsistent mit x, womit sie ein 1-Zertifikat wäre.

Eine Belegung C_2, die nur zwei Variablen auf 1 setzt wäre ebenso ein 1-Zertifikat für $f(x)$. Mit $|C_1| = 5 \leq |C_2| = 2$ fällt direkt auf, dass C_2 das kleinere 1-Zertifikat für $f(x)$ ist. C_2 wäre sogar ein kleinstmögliches 1-Zertifikat.

Definition 2.22. *Allgemein sei nun $b \in \{0,1\}$. Die* lokale b-Certificate Complexity *$C_b(f,x)$ von f auf Eingabe x ist die Größe des kleinsten möglichen b-Zertifikats, welches $f(x) = b$ 'erzwingt'.*

Um unser lokales Maß nun unabhängig von der Eingabe x zu machen, suchen wir die größte lokale b-Certificate Complexity über alle möglichen Eingaben. Wir definieren

$$C_b(f) = max_x\, C_b(f,x)$$

als die b-Certificate Complexity *für unsere Funktion f. Diese gibt für die Worst-Case-Eingabe die Größe des kleinsten möglichen b-Zertifikats zurück. Wollen wir dies auch unabhängig von unserem b machen, dann definieren wir*

$$C(f) = max\{C_0(f), C_1(f)\}$$

als die Certificate Complexity *von f.*

Also zählt die b-Certificate Complexity die minimale Anzahl an Bits, dessen Wert wir wissen müssen, um den Wert von f sicher auf $b = 0$ oder $b = 1$ bestimmen zu können. Dabei erhält man jedoch keine Informationen darüber, welche Bits belegt werden müssen oder ob es Bits gibt, die keinen Einfluss auf das Ergebnis haben. Abschließend zählt die allgemeine Certificate Complexity die Anzahl Bits die benötigt sind, um den Wert von $f(x)$ unabhängig von einem festgelegtem Ausgabewert bestimmen zu können.

Beispiel 2.23. Sei OR_n die n-stellige Oder-Funktion. $OR_n(x) = 0$ genau dann, wenn alle x_i aus x gleich 0 sind. Daraus folgt, dass $C_0(OR_n) = n$, da alle Bits bekannt sein müssen.

Für $OR_n(x) = 1$ genügt es, dass ein einziges Bit x_i gleich 1 ist. Somit folgt, dass $C_1(OR_n) = 1$. Daraus leiten wir ab, dass $C(OR_n) = max\{1, n\} = n$ die Certificate Complexity der Oder-Funktion mit n Variablen ist.

2.3.3 Repräsentierende multilineare Polynome

Auch mit mathematischen Methoden kann man die Komplexität einer Boole'schen Funktion genauer beschreiben. Dabei betrachten wir hier Polynome, die unsere Boole'schen Funktionen repräsentieren. Es gibt auch andere Maße, bei denen versucht wird sie nur zu approximieren.

Definition 2.24. *Sei $x_1, ..., x_n$ eine Menge von Variablen und $S \subseteq [n]$ eine Menge von Indizes dieser Variablen. Das Produkt*

$$X_S = \prod_{i \in S} x_i$$

nennen wir ein multilineares Monom. *Der* Grad *eines multilinearen Monoms ist festgelegt durch die Anzahl der Indizes in S, also durch $|S|$.*

Definition 2.25. *Ein* multilineares Polynom *mit n Variablen ist eine Funktion, festgelegt durch die Summe mehrerer multilinearer Monome. Ausgewertet werden diese im Kontext der Boole'schen Komplexität über Vektoren $x \in \{0,1\}^n$. Jedes Monom erhält einen Koeffizienten $c_S \in \mathbb{R}$. Aus beliebigen Mengen von Indizes lässt sich unser multilineares Polynom $p : \mathbb{R}^n \to \mathbb{R}$ also schreiben als*

$$p(x) = \sum_{S \subseteq [n]} c_S X_S.$$

Der Grad *des Polynoms p ist gleich dem Maximum aller Grade der Monome von p.*

Sofort zu sehen ist, dass alle Variablen in einem multilinearen Polynom nie einen Exponenten höher als Eins besitzen. Deshalb bezeichnet man mit ihrem Grad auch stattdessen die Anzahl Variablen im größten Monom des Polynoms.

Beispiel 2.26. Seien x_1, x_2, x_3 unsere Variablen und x der Vektor aus diesen Variablen. Aus Mengen von den Indizes 1 bis 3 können wir nun Monome konstruieren. Die Koeffizienten sind zufällig gewählt und könnten auch jede andere reelle Zahl sein.

i	S_i	Monom X_{S_i}	Koeffizient c_{S_i}	Grad
1	$\{1,2,3\}$	$x_1x_2x_3$	-1	3
2	$\{1,2\}$	x_1x_2	1	2
3	$\{1,3\}$	x_1x_3	1	2

Mit den Beispielwerten aus obiger Tabelle können wir nun unser erstes multilineares Polynom konstruieren.

$$p(x) = \sum_{1 \leq i \leq 3} c_i X_{S_i} = -x_1x_2x_3 + x_1x_2 + x_1x_3$$

Dieses hat offensichtlich Grad 3, da das erste Monom alle Variablen besitzt.

Definition 2.27. *Sei f eine n-stellige Boole'sche Funktion und $p : \mathbb{R}^n \to \mathbb{R}$ ein multilineares Polynom. Wir sagen, dass das Polynom p die Funktion f* repräsentiert*, genau dann wenn*

$$\forall x \in \{0,1\}^n \text{ gilt, dass } p(x) = f(x).$$

Tatsächlich stellt sich heraus, dass wenn zwei Polynome auf allen möglichen Eingaben Boole'scher Vektoren das gleiche Ergebnis liefern, es folgt, dass die Polynome identisch sein müssen. Ein Beweis dieser Aussage ist in [2] zu finden.

Demnach ergibt sich auch weiterhin, dass es zu jeder Boole'schen Funktion ein einzigartiges Polynom gibt, welches dieses repräsentiert. Identische Polynome sind nur möglich, wenn auch die Funktionen identisch sind. Also lohnt es sich, für diese besondere Darstellung Boole'scher Funktionen ein Komplexitätsmaß einzuführen.

Definition 2.28. *Sei f eine n-stellige Boole'sche Funktion und $p : \mathbb{R}^n \to \mathbb{R}$ ein multilineares Polynom, welches f repräsentiert.*

Man nennt deg(f) den Grad *von f, wobei deg(f) gleich dem Grad von p ist.*

Beispiel 2.29. Sei $n = 3$ die Anzahl unserer Variablen sowie die Dimension unserer Boole'schen Funktion f. Außerdem sei $x = (x_1x_2x_3)$ der Vektor mit eben diesen Variablen. Wie in Beispiel 2.19 sei $f(x) = 1$ immer dann wahr, wenn x die binäre Darstellung einer Zahl größer als 4 ist.

Mit Vektoren der Länge 3 kommen nur die Eingaben $101, 110$ und 111 als erfüllend in Frage. Naiv lässt sich für jede dieser Eingaben ein Boole'scher Ausdruck aufschreiben. Aus der Eingabe 101 erhalten wir leicht den Ausdruck

$$x_1(1 - x_2)x_3.$$

Schreiben wir alle unsere erfüllenden Eingaben als Boole'sche Ausdrücke auf und summieren diese, so erhalten wir das Polynom

$$p(x) = x_1(1 - x_2)x_3 + x_1x_2(1 - x_3) + x_1x_2x_3.$$

Durch simples Einsetzen lässt sich verifizieren, dass $p(x)$ unsere Funktion f repräsentiert. Vereinfachen des Ausdrucks liefert uns

$$p(x) = -x_1x_2x_3 + x_1x_2 + x_1x_3,$$

was genau unserem multilinearen Polynom aus Beispiel 2.26 entspricht. Damit sehen wir, dass das multilineare Polynom $p(x)$ unsere Funktion f repräsentiert. An dem ersten Monom von p erkennen wir, dass $deg(f) = 3$ ist.

2.3.4 Sensitivität

Mit Maßen über die Sensitivität (engl. sensitivity) einer Boole'schen Funktion misst man zum Einen wie anfällig eine Funktion dafür ist, einen Bitfehler in der Eingabe bis zur Ausgabe durchschleifen zu lassen. Aber nicht nur zur Fehlerkorrektur, sondern auch zur Klassifizierung und Beschreibung von Funktionen eignet sich die Sensitivität. In erster Linie zählt sie alle Bits, die bei Änderung auch den Ausgabewert ändern würden.

Definition 2.30. *Die* lokale Sensitivität $s(f, x)$ *(engl. local sensivity) misst die Sensitivität einer Funktion bezüglich einer Eingabe x. Sie definiert sich als die Anzahl aller Indizes i, für die*

$$f(x) \neq f(x^{\{i\}})$$

gilt. Für die allgemeinere Definition der Sensitivität $s(f)$ *wird dann die maximale lokale Sensitivität über allen möglichen Eingaben gesucht.*

$$s(f) = max_x\, s(f, x)$$

Das macht die Sensitivität nur von der Funktion f abhängig, wohingegen die lokale Sensitivität immer zusammen mit potenziellen Eingaben betrachtet werden muss.

2.3.5 Block-Sensitivität

Die Block-Sensitivität (engl. block sensitivity) verallgemeinert die Sensitivität weiter. Hier werden nicht mehr einzelne Bits betrachtet, sondern disjunkte Blöcke beziehungsweise Mengen von Bitindizes. Dabei werden immer alle Bits dessen Indizes in einem Block enthalten sind gekippt. Erneut ist hier die lokale Variante des Komplexitätsmaßes nur mit einer Eingabe vollständig definiert.

Definition 2.31. *Für die* lokale Block-Sensitivität $bs(f, x)$ *betrachten wir disjunkte Blöcke $B_1, ..., B_k$. Dabei gilt zusätzlich für alle $1 \leq i \leq k$, dass $B_i \subseteq [n]$.*

f wird auf einer Eingabe x betrachtet und $bs(f, x)$ zählt die maximale Anzahl an Blöcken B_i für die

$$f(x) \neq f(x^{B_i})$$

gilt. Analog zur Sensitivität ist nun auch die allgemeinere Block-Sensitivität $bs(f)$ *(engl. block sensitivity) die maximale lokale Block-Sensitivität über allen möglichen Eingaben.*

$$bs(f) = max_x\, bs(f, x)$$

2.3.6 Sensitivity Conjecture

Die Verbindungen zwischen Sensitivität und Block-Sensitivität sind nicht immer allzu leicht zu sehen. Eine erste Eigenschaft folgt direkt aus der Definition heraus. Somit gelte

$$s(f) \leq bs(f)$$

für jede beliebige Boole'sche Funktion f. Schließlich ist direkt erkennbar, dass die Sensitivität nur ein Spezialfall der Block-Sensitivität ist. Man kann jedem sensitiven Bit

x_i somit seinen eigenen Block B_i zuweisen und hätte bereits $s(f)$ disjunkte sensitive Blöcke für f. Es ist umgekehrt nicht auszuschließen, dass Blöcke B_k existieren, für die f zwar sensitiv ist, aber dessen Bits $x_k \in B_k$ alleine noch keine sensitiven Bits sind.

Mit einem Beispiel (in Anlehnung an ein allgemeineres Beispiel in [13]) lässt sich dieser Zusammenhang besser verdeutlichen.

Beispiel 2.32. Sei $n = 5$ die Dimension von Vektoren und Funktionen für dieses Beispiel. Sei weiter $x \in \{0,1\}^5$ ein Vektor und außerdem gilt $f(x) = 1$ genau dann, wenn $3 \leq |x|_{ham} \leq 4$. Für die lokalen Sensitivitätsmaße sei außerdem $x = 11100$ eine Eingabe, worunter $f(x)$ zu 1 evaluiert wird.

Das Kippen der drei vorderen Bits würde jeweils ein Kippen von $f(x)$ bedeuten, da dann $|x|_{ham} < 3$ erfüllt wäre. Beim Kippen eines der hinteren Bits wäre weiterhin $f(x) = 1$. Somit ist hier $s(f,x) = 3$ und tatsächlich ist auch $s(f) = 3$. Interessanter ist hier die lokale Block-Sensitivität $bs(f,x)$. Denn sensitive Blöcke wären hier z.B. $\{1\},\{2\},\{3\},\{4,5\}$, womit $bs(f,x) = 4$ ist. Tatsächlich ist auch $bs(f) = 4$. Demnach erfüllt diese Funktion die Ungleichung $s(f) \leq bs(f)$.

Anders herum stellt sich die Frage, ob auch die Sensitivität die Block-Sensitivität nach oben hin beschränkt. Die Antwort darauf zu finden ist bedeutend schwieriger. Gemeinhin ist diese Frage als die Sensitivity Conjecture bekannt. Erstmals gestellt von Nisan und Szegedy in 1994 vermutet sie, dass es möglich sei die Block-Sensitivität mithilfe der Sensitivität polynomiell nach oben hin zu beschränken [14].

Vermutung 2.33. *(Sensitivity Conjecture) Für jede Boole'sche Funktion f existiert eine Konstante $C > 0$, sodass*

$$bs(f) \leq s(f)^C.$$

Die weitreichende Bedeutung dieser Vermutung, die ihr Beweis nach sich zieht, ist in Kapitel 4 weiter ausgeführt. Darüber hinaus existieren auch zahlreiche Verbindungen zu den anderen eingeführten Komplexitätsmaßen Boole'scher Funktionen.

3 Beweis der Sensitivity Conjecture

In diesem Abschnitt wird der Beweis der Sensitivity Conjecture (zu finden in [9]) nachvollzogen. Dazu wird zuerst erläutert, wie genau die mathematischen Konzepte aus Kapitel 2.1 in Verbindung zur Conjecture stehen. Danach werden die zwei zentralen Lemmata des Beweises ausführlich erklärt. Abschließend werden diese Lemmata zum Beweis der Sensitivity Conjecture zusammengefügt.

3.1 Boole'sche Komplexität und der Hypercube

Um den Beweis der Conjecture zu verstehen muss man wissen, dass es andere Aussagen gibt, zu denen sie verwandt ist. So existiert zum Beispiel ein Zusammenhang zwischen der Boole'schen Komplexitätstheorie und den Betrachtungen von Subgraphen des Hypercubes [7]. Sie bildet den Grundstein für den Beweis von Huang.

Satz 3.1. *(Gotsman und Linial [7]) Die folgenden Aussagen sind äquivalent für jede monotone Funktion* $h : \mathbb{N} \to \mathbb{R}$.

a) Für jeden induzierten Subgraphen H *von* Q^n *mit* $|V(H)| \neq 2^{n-1}$ *gilt, dass* $\Gamma(H) \geq h(n)$.

b) Für jede Boole'sche Funktion f *gilt, dass* $s(f) \geq h(deg(f))$.

Dieser Satz funktioniert nur mit einer monotonen Funktion $h(n)$. Für eine jede solche Funktion h sollen somit die zwei Aussagen a) und b) äquivalent sein. Für Aussage a) betrachten wir induzierte Subgraphen des n-dimensionalen Hypercubes Q^n. Dabei muss die Anzahl Knoten $|V_H|$ des Subgraphen H ungleich 2^{n-1} sein. Da der Graph Q^n insgesamt 2^n Knoten besitzt, bedeutet dies, dass H nicht genau die Hälfte an Knoten von Q^n besitzen darf. Daraus soll laut Satz dann folgen, dass $\Gamma(H)$ größer als die gewählten Funktion $h(n)$ sein soll. Hierfür erinnern wir uns, dass $\Gamma(H)$ dem größeren maximalen Grad der beiden Graphen H oder $Q^n - H$ entspricht.

Aussage b) ist hingegen deutlich simpler. Vorerst erinnern wir uns, dass $deg(f)$ den Grad des repräsentierenden Polynoms von f bezeichnet. In dieser Aussage wird vorgeschrieben, dass für jede Boole'sche Funktion f die Sensitivität größer ist, als der Grad des repräsentierenden Polynoms in h eingesetzt.

Bewiesen wurde dieser Satz von Gotsman und Linial im Jahre 1992 [7]. Gelinge es also jemandem die Aussage a) zu beweisen, so würde daraus sofort die Aussage b) folgen. Analog gilt auch, dass aus b) auch a) folgen würde.

Außerdem wurde im Originalartikel rund um Satz 3.1 noch weiter vorgeschlagen, wie man die monotone Funktion h wählen könne. Sei somit für den restlichen Verlauf der Arbeit

$$h(n) = \sqrt{n}.$$

Würde man nun $\Gamma(H) \geq \sqrt{n}$ wie im obigen Satz beweisen, so hätte man ebenso die Beziehung $s(f) \geq \sqrt{deg(f)}$ gezeigt. Daraus würden wir durch Quadrieren dann die Gleichung $s(f)^2 \geq deg(f)$ erhalten. Dies ist nicht direkt eine Aussage über Block-Sensitivität und Sensitivität. Im nächsten Kapitel der Arbeit werden aber Zusammenhänge erläutert, die diese Verbindung herstellen und demnach auch die Bestätigung der Sensitivity Conjecture ermöglichen.

3.2 Grundlegender Aufbau des Beweises

Der Großteil der Arbeit von Huang beschäftigt sich mit einem Beweis der Aussage a) aus Satz 3.1. Allerdings ist diese Aussage allgemeiner gestellt, als es für uns nötig ist, weshalb sie ein wenig vereinfacht wird. Es werden nun, genau wie in Satz 3.1, zufällig induzierte Subgraphen H des Hypercubes betrachtet.

Die Aussage a) setzt als Restriktion fest, dass die Anzahl Knoten für H ungleich 2^{n-1} gewählt werden soll. Dies ist aus einem ähnlichen Phänomen wie auch im Beispiel 2.16 gewachsen. Denn Subgraphen, welche halb so groß sind wie der Hypercube aus dem sie induziert wurden, können leicht als Graphen ohne Kanten gewählt werden. In diesen Fällen wäre $\Gamma(H) = 0$ und es ließen sich keine weiteren Aussagen in Bezug auf Satz 3.1 treffen. Wenn wir also einen Graphen H mit $|V_H| \neq 2^{n-1}$ wählen, dann folgt daraus, dass entweder H oder $Q^n - H$ mindestens mehr als die Hälfte aller Knoten, also mindestens $2^{n-1} + 1$ Knoten besitzt. Ohne Beschränkung der Allgemeinheit können wir davon ausgehen, dass H immer der Graph mit mehr Knoten und mit den größeren maximalen Grad sei. Dadurch lässt sich auch unser Ausdruck für $\Gamma(H)$ vereinfachen.

$$\Gamma(H) = max\{\Delta(H), \Delta(Q^n - H)\} \geq max\{\Delta(H)\} = \Delta(H)$$

Da wir eine untere Schranke für $\Gamma(H)$ finden wollen, reicht es nun aus eine untere Schranke für $\Delta(H)$ zu finden.

Würden wir noch mehr Knoten des Hypercubes zu H anstatt zu $Q^n - H$ hinzuzählen, so könnte der maximale Grad $\Delta(H)$ nur steigen. Schließlich fügen wir nur Knoten und mögliche Kanten hinzu und entfernen keinerlei existierende Kanten. Dementsprechend genügt es uns nun H auf zufällig induzierte Subgraphen mit $2^{n-1} + 1$ Knoten zu beschränken. Denn wenn jeder zufällige Subgraph der Größe $2^{n-1} + 1$ die Ungleichung $\Delta(H) \geq \sqrt{n}$ erfüllt, so würde auch jeder zufällige Subgraph mit einem, oder beliebig vielen Knoten mehr, es erfüllen.

Damit sind wir bereit, die Aussage a), als einen neuen Satz zu formulieren.

Satz 3.2. *Für jedes $n \geq 1$, sei $H = (V_H, E_H)$ ein zufällig induzierter Subgraph von Q^n mit $|V_H| = 2^{n-1} + 1$. Dann gilt*

$$\Delta(H) \geq \sqrt{n}.$$

Um diese vereinfachte Formulierung der Aussage a) noch besser zu verstehen, werden Beispiele des Satzes 3.2 für die Dimensionen $n = 2$ und $n = 3$ vorgestellt.

Beispiel 3.3. Sei zuerst $n = 2$ die Dimension des Hypercubes. Die Größe des Subgraphen H ist durch Satz 3.2 als $2^{n-1} + 1 = 3$ festgelegt. Aus der Kombinatorik wissen wir, dass mit $\binom{4}{3} = 4$ folgt, dass nur 4 verschiedene Subgraphen H dieser Größe existieren. Denn $\binom{4}{3}$ sagt aus, wieviele Möglichkeiten es gibt 3-elementige Teilmengen einer Menge der Größe 4, welche hier unsere Knotenmenge ist, zu entnehmen.

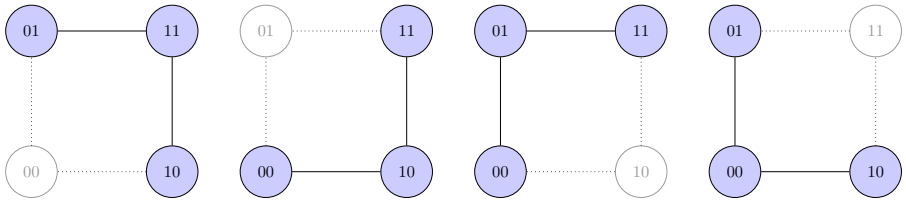

Abbildung 3.1: Induzierte Subgraphen von Q^2 mit 3 Knoten

Es ist nicht möglich einen Teilgraphen zu konstruieren, der nicht mindestens einen Knoten mit Grad 2 besitzt. Dies wird durch die Abbildung 3.1 weiter verdeutlicht. Damit gilt $\Delta(H) = 2 \geq \sqrt{2} \approx 1.41$. Dies ist konsistent mit Satz 3.2.

In dem zweidimensionalem Beispiel ist es noch relativ trivial die Korrektheit des Satzes zu sehen. Aber bereits in der dritten Dimension ist es deutlich mühsamer, dies nachvollziehen zu können.

Beispiel 3.4. Sei nun $n = 3$ die Dimension des Hypercubes. Damit ist jetzt $2^{n-1}+1 = 5$. Wir suchen also zufällige induzierte Subgraphen H vom Q^3, die 5 Knoten besitzen. Da mit $\binom{8}{5} = 56$ folgt, dass es 56 Möglichkeiten dafür gibt, stellen wir nur zwei interessante Subgraphen vor. Somit wählen wir als die ersten 4 Knoten die Menge $V_{temporär} = \{000, 110, 011, 101\}$.

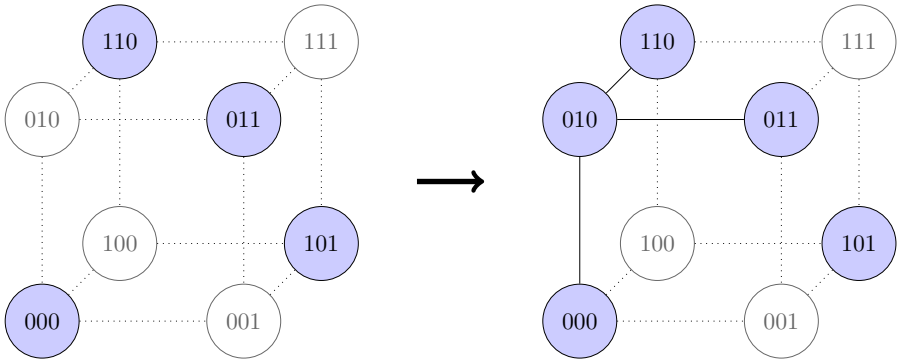

Abbildung 3.2: Möglicher induzierter Subgraph von Q^3 mit 5 Knoten

Diese Knoten haben paarweise eine Hamming-Distanz von 2 und deshalb existieren zwischen ihnen keine Kanten. Es liegt der gleiche Fall vor wie im Beispiel 2.16. Fürs Erste ist also $\Delta(H) = 0$. Egal welchen Knoten wir als fünftes hinzufügen, wird dieser dann plötzlich einen Grad von 3 haben, womit $\Delta(H) = 3 \geq \sqrt{3} \approx 1.73$. Beispielhaft haben wir in Abbildung 3.2 nun den Knoten (010) gewählt.

Mit einer anderen Wahl von Knoten ist es sogar möglich Subgraphen mit 6 Knoten zu erzeugen, die einen maximalen Grad von 2 besitzen. Dafür wählen wir stattdessen die Menge $V_{temporär} = \{000, 001, 011, 100, 110, 111\}$. Doch egal welchen der Knoten wir weglassen, um einen Subgraphen mit 5 Knoten zu erhalten, werden wir immer einen Graph mit mindestens Grad 2 behalten. Veranschaulicht ist dieser besondere Subgraph in Abbildung 3.3.

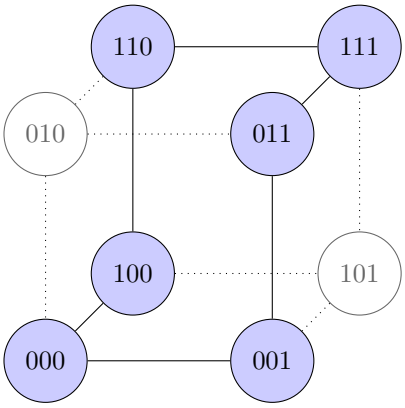

Abbildung 3.3: Zyklische Tour um den Hypercube mit $\Delta(H) = 2$

Ersichtlich wird also auch hier, dass mit $\Delta(H) = 2 \geq \sqrt{3} \approx 1.73$ auch in diesem Beispiel der Satz 3.2 seine Richtigkeit behält.

Bereits für die vierte Dimension ist es nicht mehr sinnvoll einzelne Subgraphen zu konstruieren um unser Kriterium zu überprüfen. Denn wie in den Beispielen erkennen wir durch $\binom{16}{9} = 11440$, dass es 11440 mögliche Subgraphen vom Q^4 gäbe. Demnach sind weniger naive, mathematische Methoden nötig um Satz 3.2 vollständig zu zeigen.

Ein Beweis von Satz 3.2 wäre weiterführend auch ein Beweis der Aussage a) aus Satz 3.1 mit der Funktion $h(n) = \sqrt{n}$. Unter der Annahme, dass Satz 3.2 bewiesen wäre, gilt laut Ausführungen in Kapitel 3.1, dass für jede Boole'sche Funktion f gilt, dass

$$s(f)^2 \geq deg(f). \tag{3.1}$$

Als Nächstes macht sich Huang ein weiteres wichtiges Forschungsergebnis Boole'scher Komplexitätsmaße zunutze. So wurde in [18] die Gleichung

$$deg(f)^2 \geq bs(f) \tag{3.2}$$

für jede Boole'sche Funktion f bewiesen.

Dabei handelt es sich um eine Verbesserung der vorher bekannten Trennung der beiden Komplexitätsmaße. Insbesondere stellt die Gleichung 3.2 die Verbindung zwischen dem repräsentierendem Polynom und der Block-Sensitivität her. Erneutes Quadrieren der Gleichung 3.1 und das Kombinieren beider Gleichungen liefert uns die bestätigte Sensitivity Conjecture aus Vermutung 2.33 mit $C = 4$.

Satz 3.5. *Sei f eine beliebige Boole'sche Funktion. Es gilt, dass*

$$bs(f) \leq deg(f)^2 \leq s(f)^4.$$

3.3 Beweis der Sensitivity Conjecture

Aus Kapitel 3.2 geht hervor, dass ein Beweis von Satz 3.2 auch die Sensitivity Conjecture beweisen würde.

$$Satz\ 3.2 \Rightarrow Aussage\ a) \Rightarrow Aussage\ b) \Rightarrow Satz\ 3.5$$

Dafür benötigt Huang eine Reihe von Lemmata. Das Erste ist Cauchy's Interlace Theorem (siehe 2.12). Der Beweis davon wird nicht von Huang selbst gebracht, sondern kann zum Beispiel in [5] nachvollzogen werden. Für ein weiteres Lemma definiert Huang eine rekursive Matrix A_n und zeigt einige ihrer Eigenschaften.

3.3.1 Erstes Lemma: Rekursive Matrizen

Definition 3.6. *Die Matrix A_n ist rekursiv definiert als*

$$A_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ A_n = \begin{pmatrix} A_{n-1} & I \\ I & -A_{n-1} \end{pmatrix}.$$

Beispiel 3.7. Um A_n besser zu verstehen, lohnt es sich für die ersten Iterationen der Rekursion die Matrix zu konstruieren. Für $n = 2$ und $n = 3$ erhalten wir

$$A_2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & -1 \\ 0 & 1 & -1 & 0 \end{pmatrix},\ A_3 = \begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \\ 0 & 1 & 0 & 0 & -1 & 0 & 0 & -1 \\ 0 & 0 & 1 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & -1 & 1 & 0 \end{pmatrix}.$$

Sofort fällt auf, dass alle Einträge der Matrizen stets in $\{-1, 0, 1\}$ sind. Außerdem ist die Matrix A_n immer der Größe $2^n \times 2^n$. Zuletzt fällt noch die ständige Symmetrie von A_n auf.

Huang betrachtet nun die Eigenwerte dieser Matrix und definiert dazu das erste wichtige Lemma.

Lemma 3.8. *A_n besitzt nur die Eigenwerte $\sqrt{n}$ und $-\sqrt{n}$, welche jeweils mit einer Multiplizität von 2^{n-1} auftreten.*

Da A_n trickreich definiert ist, lässt sich jedoch eine herkömmliche Betrachtung der Eigenwerte nur schwer durchführen. Mit einem Lemma aus der linearen Algebra lässt es sich aber deutlich vereinfachen.

Lemma 3.9. *Sei A eine $n \times n$ Matrix und $A^2 = A * A$ dessen Quadrat. Es gilt, dass*

$$\lambda \text{ ist Eigenwert von } A \Rightarrow \lambda^2 \text{ ist Eigenwert von } A^2.$$

Beweis. (von Lemma 3.9)
Angenommen λ ist Eigenwert von A mit einem Eigenvektor $\vec{x}$. Dann ist

$$A\vec{x} = \lambda\vec{x}$$

die definierte Eigenschaft dieses Eigenvektors. Da die Multiplikation unter anderem auch bei Matrizen assoziativ ist, können wir schreiben, dass

$$A^2\vec{x} = A(A\vec{x}) = A(\lambda\vec{x}) = \lambda(A\vec{x}) = \lambda^2\vec{x}.$$

Da $A^2\vec{x} = \lambda\vec{x}$ gilt, muss λ^2 auch ein Eigenwert von A^2 sein. □

Hiermit sind wir nun auch bereit das Lemma 3.8 von Huang zu beweisen.

Beweis. (von Lemma 3.8)
Um unser Lemma 3.9 zu verwenden zeigen wir zuerst, dass $A_n^2 = nI$ gilt. Dazu ist es wichtig zu wissen, dass die Wurzel einer Matrix nur in bestimmten Fällen eindeutig ist. Demnach wäre ein Beweis der Identität $A_n^2 = nI$ kein Widerspruch zur Definition der Matrix A_n. Der Beweis geschieht per Induktion.

- **Anfang.** Sei $n = 1$. $A_1^2 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}^2 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} = I$.
- **Annahme.** Nehme an, dass $A_{n-1}^2 = (n-1)I$ gelte.
- **Schluss.**

$$\begin{aligned} A_n^2 &= \begin{pmatrix} A_{n-1} & I \\ I & -A_{n-1} \end{pmatrix} \begin{pmatrix} A_{n-1} & I \\ I & -A_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} A_{n-1}A_{n-1} + I^2 & (A_{n-1}I) - (A_{n-1}I) \\ (A_{n-1}I) - (A_{n-1}I) & A_{n-1}A_{n-1} + I^2 \end{pmatrix} \\ &= \begin{pmatrix} A_{n-1}^2 + I & 0 \\ 0 & A_{n-1}^2 + I \end{pmatrix} \\ &= \begin{pmatrix} (n-1)I + I & 0 \\ 0 & (n-1)I + I \end{pmatrix} = \begin{pmatrix} nI & 0 \\ 0 & nI \end{pmatrix} = nI \end{aligned}$$

Somit ist $A_n^2 = nI$ bewiesen und wir sind bereit die Eigenwerte von A_n zu betrachten. Die Matrix nI hat trivialerweise nur den Eigenwert n. Angenommen A_n hat nun den Eigenwert λ. Aus Lemma 3.9 folgt, dass λ^2 dann ein Eigenwert von nI sein muss, also $\lambda^2 = n$. Also ergibt sich die Aussage, dass

$$\lambda^2 = n \text{ genau dann, wenn } \lambda = \pm\sqrt{n},$$

da sowohl $\sqrt{n}^2 = n$ als auch $(-\sqrt{n})^2 = n$ wahr ist.

Hierdurch haben wir festgestellt, dass A_n nur die Eigenwerte $\sqrt{n}$ und $-\sqrt{n}$ besitzen kann. Für die Aussage über die Multiplizität der Eigenwerte betrachten wir zuletzt noch die Spur unserer Matrix A_n. Hierfür bezeichnen wir mit $a_{i,j}$ den Eintrag der Matrix A_n in Zeile i und Spalte j. Offensichtlich existieren nach Definition auf der Hauptdiagonalen von A_n nur Nullen. Wichtig ist aber, dass es laut Lemma 2.8 verschiedene Möglichkeiten gibt, um die Spur zu berechnen. Dafür erinnern wir uns daran, wie wir die Spur auch als Summe aller Eigenwerte schreiben können. Da es nur zwei mögliche Eigenwerte gibt definieren wir vorerst c_1 und c_2 als deren Multiplizitäten. Dann ergibt sich die Gleichung

$$Spur(A_n) = \sum_{i=1}^{2^n} a_{i,i} = \sum_{i=1}^{2^n} 0 = 0 = -c_1\sqrt{n} + c_2\sqrt{n}.$$

Dies ist nur erfüllt, wenn auch

$$c_1\sqrt{n} = c_2\sqrt{n}$$

erfüllt ist. Also muss $c_1 = c_2$ gelten, womit wir bewiesen haben, dass die Eigenwerte $\sqrt{n}$ und $-\sqrt{n}$ jeweils mit gleicher Multiplizität auftauchen. Würden wir vermuten, dass einer der möglichen Eigenwerte kein tatsächlicher Eigenwert ist, so wäre es nicht mehr möglich auf eine Spur von Null zu kommen. Demnach müssen $\sqrt{n}$ und $-\sqrt{n}$ tatsächlich die einzigen Eigenwerte von A_n sein und beide mit algebraischer Vielfachheit von 2^{n-1} auftauchen, was genau die Aussage aus Lemma 3.8 ist. □

Wozu diese Matrix nun nützlich ist, ist nicht direkt auf den ersten Blick zu sehen. Allerdings stellt sich heraus, dass man durch Verändern der Einträge von -1 zu 1 genau die Adjazenzmatrix des jeweiligen Hypercubes Q^n erhält.

Vermutung 3.10. *Sei A_n wie in Definition 3.6 definiert. Ändert man alle Einträge von A_n die ungleich 0 sind zu einer 1, so ist A_n die Adjazenzmatrix des n-dimensionalen Hypercubes.*

Bezeichnen wir also nun mit B_n die tatsächliche Adjazenzmatrix des Hypercubes Q^n. Die Zeilen und Spalten unserer Matrizen adressieren jeweils unsere Knoten, welche als binäre Vektoren $x \in \{0,1\}^n$ dargestellt werden.

Beispiel 3.11. Sei $1 \leq n \leq 2$. Die dazugehörigen Graphen des Hypercubes Q^1 und Q^2 sind zu sehen in Abbildung 2.2. Die Adjazenzmatrizen sehen aus wie folgt.

$$B_1 = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ B_2 = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Schauen wir zurück auf die Matrizen A_1 und A_2 in Beispiel 3.7, so gilt hier unsere Vermutung.

Ein formaler Beweis der Vermutung 3.10 ist schwer durchzuführen. Macht man sich die Konstruktion des Hypercubes aber noch einmal deutlich, so wird klar, dass sie stimmen muss. Laut Kapitel 2.2 können wir den Hypercube leicht konstruieren.

Wir bezeichnen mit B_n weiter die Adjazenzmatrix des Q^n und mit A_n unsere Matrix aus Definition 3.6.

B_1 und B_2 sind trivial aus unserem Beispiel 3.11 gegeben. Wir beginnen für unsere Konstruktion also mit B_n. Der Graph Q^n besitzt 2^n Knoten, welche wir mit $k_1, \dots, k_{2^n}$ bezeichnen. Da jeder Knoten einem Vektor aus $\{0,1\}^n$ entspricht, können wir jedem Knoten als Index den Wert geben, den er in Binärdarstellung kodiert. Beim Erhöhen der Dimension verdoppelt sich die Anzahl der Knoten auf 2^{n+1}. Analog verdoppelt sich die Dimension der Matrizen ebenfalls. Alle bereits bekannten Knoten $k_1, \dots, k_{2^n}$ erhalten beim Erhöhen der Dimension also als Most-Significant-Bit eine Null. Alle neu hinzugefügten Knoten $k_{2^n+1}, \dots, k_{2^{n+1}}$ ergeben sich indem man stattdessen eine Eins hinzufügt.

Laut Konstruktionsvorschrift des Hypercubes bilden die Knotenmengen $k_1, \dots, k_{2^n}$ sowie $k_{2^n+1}, \dots, k_{2^{n+1}}$ jeweils einen n-dimensionalen Hypercube. Für den Anfang ist unsere Adjazenzmatrix B_{n+1} also

$$B_{n+1} = \begin{pmatrix} \overbrace{B_n}^{2^n} & \overbrace{\dots}^{2^n} \\ \dots & B_n \end{pmatrix}.$$

In roter Färbung sehen wir die komplette Adjazenzmatrix des vorherigen Hypercubes der Dimension n. Diese bildet eine $2^n \times 2^n$ große Submatrix in der oberen Ecke von B_{n+1}. Alle 2^n neu erzeugten Zeilen und Spalten korrespondieren somit zu den Knoten des 'zweiten' Hypercubes der Dimension n, der neu konstruiert wurde. Seine Adjazenzmatrix beginnt also bei Zeile und Spalte $2^n + 1$.

Im nächsten Schritt der Konstruktion müssen die Knoten beider Teilcubes verbunden werden, wenn sie eine Hamming-Distanz von Eins besitzen. In Kapitel 2.2 wurde erwähnt, dass die neuen Knoten einfach erzeugt werden, indem sie eine führende Eins als Most-Significant-Bit erhalten. Alle vorher existierenden Knoten erhalten eine Null.

Fixieren wir also einen Knoten k_i aus dem n-dimensionalen Hypercube. Die i-te Spalte und i-te Zeile der Matrix B_n korrespondiert jeweils zu diesem Knoten. Das Hinzufügen einer Null als Most-Significant-Bit ändert nichts an dem Wert, den k_i binär kodiert. Demnach korrespondiert auch die i-te Zeilte und Spalte der Matrix B_{n+1} zu dem Knoten k_i.

Für jeden Knoten k_i muss jedoch im neu erzeugten Hypercube ein Knoten mit führender Eins entstanden sein. Die führende Eins erhöht den kodierten Wert um 2^n, weshalb der Knoten k_{i+2^n} der 'Zwillingsknoten' von k_i ist. Diese beiden Knoten unterscheiden sich nur im Most-Significant-Bit. Dadurch haben sie eine Hamming-Distanz von Eins und es muss eine Kante zwischen ihnen existieren. Also haben wir

festgestellt, dass die Kante (k_i, k_{i+2^n}) oder auch konkreter die Kanten

$$(k_1, k_{1+2^n}), \ldots, (k_i, k_{i+2^n}), \ldots, (k_{2^n}, k_{2^{n+1}})$$

hinzugefügt werden müssen. Weitere Kanten existieren zwischen den beiden Teilcubes nicht, da aufgrund der unterschiedlichen Most-Significant-Bits nur 'Zwillingsknoten' eine Hamming-Distanz von Eins haben können. Innerhalb der Teilcubes sind auch bereits alle nötigen Kanten in der Adjazenzmatrix vermerkt. Wir setzen nun also alle verbleibenden Einträge der Art $b_{i,i+2^n}$ in der Adjazenzmatrix B_{n+1} auf Eins und die restlichen auf Null und erhalten die Matrix in Abbildung 3.4.

	k_1	$\ldots$	k_i	$\ldots$	k_{2^n+1}	$\ldots$	k_{i+2^n}	$\ldots$	$k_{2^{n+1}}$
k_1					1	0	$\ldots$	$\ldots$	0
$\vdots$			B_n		0	$\ddots$	$\ddots$		$\vdots$
k_i					$\vdots$	$\ddots$	1	$\ddots$	$\vdots$
$\vdots$					$\vdots$		$\ddots$	$\ddots$	0
k_{2^n}					0	$\ldots$	$\ldots$	0	1
k_{2^n+1}									
$\vdots$			$\cdots$				B_n		
$k_{2^{n+1}}$									

Abbildung 3.4: Die bisher konstruierte Matrix B_{n+1}

Sofort ist zu erkennen, dass sich im ersten Quadranten unserer Matrix eine Einheitsmatrix bildet. Weil der Hypercube ungerichtet ist, muss die Adjazenzmatrix symmetrisch sein, weshalb sich die gleiche Einheitsmatrix auch im dritten Quadranten bilden muss. Als finale Adjazenzmatrix des Q^{n+1} haben wir damit

$$B_{n+1} = \begin{pmatrix} B_n & I \\ I & B_n \end{pmatrix}.$$

Zur Erinnerung, unsere Matrix A_n ist rekursiv definiert als

$$A_n = \begin{pmatrix} A_{n-1} & I \\ I & -A_{n-1} \end{pmatrix}.$$

Weil A_1 und B_1 laut Definition 3.6 und Beispiel 3.11 offensichtlich gleich sind, bestätigt sich auch unsere Vermutung 3.10.

3.3.2 Zweites Lemma: Graphen und ihr maximaler Grad

Das zweite zentrale Lemma von Huangs Arbeit befasst sich direkt mit Graphen. So wird für dieses Lemma ein Graph $H = (V_H, E_H)$ betrachtet. Die Knotenmenge V_H wird beschrieben durch die Knoten $k_1, \ldots, k_m$, wobei $m = |V_H|$.

Damit dieses Lemma auch zusammen mit dem Lemma 3.8 später zu Aussagen führen kann, wird dazu noch eine Matrix A der Größe $m \times m$ betrachtet, wobei jeder Knoten aus H zu einer Spalte und einer Zeile von A korrespondiert, ähnlich wie es bei einer Adjazenzmatrix der Fall wäre. Diese Matrix A darf nur Einträge aus $\{-1, 0, 1\}$ besitzen. Außerdem soll gelten, dass ein Eintrag in A immer dann gleich Null ist, wenn zwischen den Knoten der dazugehörigen Spalte und Reihe keine Kante in H existiert.

Schnell wird klar, dass im späteren Verlauf der Arbeit die Matrix A noch als A_n gewählt werden kann. Insbesondere kann Huang mit dem folgenden Lemma die Verbindung zwischen dem maximalen Grad $\Delta(H)$ und dem größten Eigenwert $\lambda_1(A)$ herstellen.

Lemma 3.12. *Sei $H = (V_H, E_H)$ ein Graph mit $m = |V_H|$ Knoten und A eine symmetrische Matrix der Größe $m \times m$, dessen Einträge alle in $\{-1, 0, 1\}$ liegen. Die Zeilen und Spalten von A korrespondieren jeweils zu den Knoten von H. Außerdem gilt für alle Einträge $a_{i,j}$ aus A, dass*

$$a_{i,j} = 0 \text{ genau dann wenn } (k_i, k_j) \notin E_H.$$

Dann gilt, dass

$$\Delta(H) \geq \lambda_1(A).$$

Für den größten Eigenwert von A führt Huang zwecks Übersichtlichkeit die Notation

$$\lambda_1 := \lambda_1(A)$$

ein. Das Lemma lässt sich sofort beweisen.

Beweis. (von Lemma 3.12)
Wir interessieren uns für den größten Eigenwert von A, genannt λ_1. Die Einträge der Matrix A adressieren wir mit dem Ausdruck $a_{i,j}$ für alle $0 \leq i, j \leq m$. Außerdem gehen wir davon aus, dass $\vec{v}$ der Eigenvektor zu dem Eigenwert λ_1 ist. Mit dem Ausdruck v_j bezeichnen wir den Eintrag eines Vektors an j-ter Stelle. Insgesamt folgt aus der Definition von Eigenwerten, dass

$$\lambda_1 \vec{v} = A\vec{v}.$$

Außerdem gehen wir ohne Beschränkung der Allgemeinheit davon aus, dass für ein fest gewähltes i der Eintrag v_i den größten absoluten Wert aller Einträge vom Vektor $\vec{v}$ besitzt. Für den Beweis beschränken wir uns dann bei der Formel für Eigenwerte nur auf die Berechnungen an i-ter Stelle des Vektors $\vec{v}$.

$$|\lambda_1 v_i| = |\lambda_1(\vec{v})_i| = |(A\vec{v})_i| \tag{3.1}$$

Das Ergebnis von $A\vec{v}$ wird ein Vektor sein. Bei diesem interessieren wir uns aber nur für das Ergebnis an i-ter Stelle. Also können wir die Definition der Matrix-Vektor-Multiplikation in Summenform aufschreiben.

$$|(A\vec{v})_i| = \left| \sum_{j=1}^{m} a_{i,j} v_j \right| \tag{3.2}$$

Laut unserer Annahme, dass v_i den betragsmäßig größten Wert besitzt, gilt für jedes $1 \leq j \leq m$, dass $v_j \leq v_i$. Außerdem können wir die Betragsfunktion im gleichen Schritt mit in die Summe ziehen. Also können wir die Summe weiter abschätzen.

$$\left| \sum_{j=1}^{m} a_{i,j} v_j \right| \leq \sum_{j=1}^{m} |a_{i,j}||v_i| \tag{3.3}$$

In Gleichung 3.3 sehen wir einerseits, dass wir $|v_i|$ aus der Summe rausziehen könnten, weil es nicht mehr von j abhängt. Weiter sehen wir, dass der Ausdruck

$$\sum_{j=1}^{m} |a_{i,j}|$$

alle Elemente in der i-ten Reihe von A aufsummiert. Dabei sind alle Einträge aufgrund der Betragsfunktion in $\{0, 1\}$. Die Restriktionen in Lemma 3.12 fordern, dass

$$a_{i,j} = 0 \text{ genau dann wenn } (k_i, k_j) \notin E_H.$$

Das bedeutet, dass in der genannten Summe immer dann eine Eins mit hineinfließt, wenn der Knoten k_i eine Kante zu k_j besitzt. Da wir über alle möglichen Knoten $k_1, \ldots, k_m$ iterieren, berechnen wir den Grad von Knoten k_i.

Nach Definition gilt immer, dass der Grad eines Knotens kleiner dem maximalen Grad seines Graphens sein muss. Dementsprechend erhalten wir zuletzt die Gleichung

$$\sum_{j=1}^{m} |a_{i,j}||v_i| \leq \Delta(H)|v_i|. \tag{3.4}$$

Durch das Zusammenfügen aller vier Gleichungen kommen wir insgesamt auf die Ungleichung

$$|\lambda_1 v_i| = |\lambda_1(\vec{v})_i| = |(A\vec{v})_i| = \left| \sum_{j=1}^{m} a_{i,j} v_j \right| \leq \sum_{j=1}^{m} |a_{i,j}||v_i| \leq \Delta(H)|v_i|. \tag{3.5}$$

Wenn wir durch $|v_i|$ dividieren erhalten wir die Aussage unseres Lemmas. Offensichtlich ist $|v_i| > 0$, da der Vektor $\vec{v}$ sonst nur aus Nullen bestehen würde. Jedoch sollten Eigenvektoren immer ungleich dem Nullvektor sein.

Insgesamt gilt also mit allen Annahmen des Lemmas, dass

$$|\lambda_1| \leq \Delta(H).$$

□

3.3.3 Kombination von Lemma 3.8 und 3.12

Mit dem Beweis der Lemmata 3.8 und 3.12 sind wir bereit Satz 3.2 und damit auch die Sensitivity Conjecture zu belegen. Dafür erinnern wir uns an die Aussage von Satz 3.2.

Satz 3.2. *Für jedes $n \geq 1$, sei $H = (V_H, E_H)$ ein zufällig induzierter Subgraph von Q^n mit $|V_H| = 2^{n-1} + 1$. Dann gilt*

$$\Delta(H) \geq \sqrt{n}.$$

Beweis. (von Satz 3.2)
Wir beginnen mit dem Graph $Q^n = (V_{Q^n}, E_{Q^n})$ des n-dimensionalen Hypercubes. Außerdem erinnern wir uns an die in Definition 3.6 beschriebene Matrix A_n. Offensichtlich sind alle Einträge von A_n in $\{-1, 0, 1\}$. Die Knoten $v_1, \dots v_{2^n}$ des Hypercubes korrespondieren jeweils zu einer Zeile und Spalte aus A_n. Mit unserer Vermutung 3.10 bestätigt sich außerdem, dass für alle Einträge $a_{i,j}$ von A_n gilt, dass

$$a_{i,j} = 0, \text{ genau dann wenn } (v_i, v_j) \notin E_{Q^n}.$$

Dadurch stellen wir fest, dass Q^n und A_n die Forderungen für Lemma 3.12 erfüllen.

Im nächsten Schritt betrachten wir nun zufällig induzierte Subgraphen H des Q^n mit genau $2^{n-1} + 1$ Knoten. Alle Knoten aus dem komplementären Graphen $Q^n - H$ kamen nun im Q^n, aber nicht mehr im Subgraph H vor.

Für jeden Knoten v_i, der Teil von $Q^n - H$ ist, löschen wir nun in A_n die jeweils korrespondierende Spalte und Zeile. Nach dem Löschen erhalten wir die Matrix A_H. Sie hat Größe $V_H \times V_H$ und ist eine quadratische Teilmatrix von A_n, da die gleiche Anzahl Spalten und Zeilen gelöscht wurde und A_n vorher auch quadratisch war. Wir bezeichnen weiter alle Eigenwerte von A_n mit $\lambda_1 \geq \dots \geq \lambda_{2^n}$ und alle Eigenwerte von A_H mit $\mu_1 \geq \dots \geq \mu_{2^{n-1}+1}$.

Erneut sehen wir mit der Vermutung 3.10 und der Definition eines Subgraphen, dass H und A_H die Vorraussetzungen für Lemma 3.12 erfüllen. Dadurch können wir schließen, dass der maximale Grad von H größer gleich dem größten Eigenwert von A_H sein muss. Formell gilt also

$$\Delta(H) \geq \mu_1, \tag{3.6}$$

denn die jeweils größten Eigenwerte unserer Matrizen A_n und A_H sind λ_1 respektive μ_1. Weil A_H eine quadratische Teilmatrix von der symmetrischen Matrix A_n ist, können wir Cauchy's Interlace Theorem (siehe 2.12) anwenden. Hierfür ist $N = |V_{Q^n}| = 2^n$ und $m = |V_H| = 2^{n-1} + 1$. Wählen wir letztlich $i = 1$, so gilt die Gleichung

$$\lambda_1 \geq \mu_1 \geq \lambda_{i+N-m} = \lambda_{1+2^n-2^{n-1}-1} = \lambda_{2^{n-1}}.$$

Wichtig ist hierbei nur die Tatsache, dass

$$\mu_1 \geq \lambda_{2^{n-1}}. \tag{3.7}$$

Schließlich ist μ_1 der größte Eigenwert von A_H. Außerdem wissen wir mit Lemma 3.8, dass die erste Hälfte der Eigenwerte $\lambda_1, \ldots, \lambda_{2^{n-1}}$ von A_n allesamt gleich $\sqrt{n}$ sind. Damit ist auch $\lambda_{2^{n-1}} = \sqrt{n}$. Kombiniert mit Gleichung 3.7 zeigen wir demnach, dass

$$\mu_1 \geq \sqrt{n}. \tag{3.8}$$

Zusammen mit Gleichung 3.6 ergibt sich der bestätigte Satz.

$$\Delta(H) \geq \mu_1 \geq \sqrt{n}.$$

□

Der Beleg von Satz 3.2 gibt nun Grund dazu, den Beweis rückwirkend zu betrachten. Mit Satz 3.2 haben wir bewiesen, dass jeder induzierte Subgraph des n-dimensionalen Hypercubes mit insgesamt $2^{n-1} + 1$ Knoten einen Grad von mindestens $\sqrt{n}$ besitzt. Indirekt haben wir damit durch die Überlegungen zum $\Gamma(H)$, präsentiert in Kapitel 3.2, auch die Aussage a) aus Satz 3.1 belegt.

Satz 3.1. *(Gotsman und Linial [7]) Die folgenden Aussagen sind äquivalent für jede monotone Funktion $h : \mathbb{N} \to \mathbb{R}$.*

a) Für jeden induzierten Subgraphen H von Q^n mit $|V(H)| \neq 2^{n-1}$ gilt, dass $\Gamma(H) \geq h(n)$.

b) Für jede Boole'sche Funktion f gilt, dass $s(f) \geq h(deg(f))$.

Folglich haben wir auch die Aussage b) aus diesem Satz belegt, weshalb wir die Gleichung

$$s(f) \geq \sqrt{deg(f)}$$

bewiesen haben. Zusammen mit dem bereits erwähnten Forschungsergebnis aus [18], welches besagt, dass

$$deg(f)^2 \geq bs(f)$$

gilt, können wir also sofort die bestätigte Sensitivity Conjecture hinschreiben.

Satz (Sensitivity Conjecture). *Sei f eine beliebige Boole'sche Funktion. Es gilt, dass*

$$bs(f) \leq s(f)^4.$$

Dies konkludiert eine über 20 Jahre offen stehende Fragestellung auf präzise und gut nachvollziehbare Art und Weise.

4 Auswirkungen der Sensitivity Conjecture

Die Bestätigung der Sensitivity Conjecture bringt vielerlei Folgen für die Komplexitätstheorie mit sich. Einige Resultate und Zusammenhänge werden dazu in diesem Kapitel erläutert.

4.1 Komplexitätsmaße Boole'scher Funktionen und ihre Relationen

In den Grundlagen aus Kapitel 2.3 wurden verschiedene Komplexitätsmaße Boole'scher Funktionen eingeführt. Bevor die Sensitivity Conjecture als gelöst galt war bekannt, dass alle diese Funktionen eng miteinander verwandt sind. Trotz aller Ähnlichkeiten zur Block-Sensitivität galt die Sensitivität als Ausnahme, welche sich nur schwer einordnen ließ.

Definition 4.1. *Sei f eine beliebige Boole'sche Funktion und $A(f)$ sowie $B(f)$ Komplexitätsmaße. Wir nennen A und B polynomiell verwandt, wenn für konstante Faktoren c_1 und c_2 gilt, dass*

$$A(f) \leq B(f)^{c_1} \text{ und } B(f) \leq A(f)^{c_2}.$$

Es stellt sich letztlich heraus, dass alle der in Kapitel 2.3 eingeführten Komplexitätsmaße, namentlich

$$D(f),\ C(f),\ deg(f) \text{ und } bs(f),$$

miteinander polynomiell verwandt sind. Außerdem können mit einer Vermutung aus Kapitel 2.3.6 erkennen, dass

$$s(f) \leq bs(f).$$

gilt. Mit unserer bestätigten Sensitivity Conjecture haben wir weiterhin die Gleichung

$$bs(f) \leq s(f)^4$$

für alle Boole'schen Funktionen f. Damit können wir sofort sehen, dass die Sensitivität und Block-Sensitivität beide nach Definition 4.1 polynomiell verwandt sind. Aufgrund der transitiven Natur der polynomiellen Verwandtschaft ist die Sensitivität somit auch mit allen anderen erwähnten Komplexitätsmaßen verwandt. Demnach hat Huang die Familie dieser Maße sinnvoll erweitern können. Inbesondere ist dabei die Einfachheit der Sensitivität hervorzuheben, da viele andere Maße deutlich schwieriger zu berechnen sind.

Um da nicht den Überblick zu verlieren lohnt sich die Konstruktion einer überschaubaren Tabelle der einzelnen Beziehungen. Andere ausführliche Versionen dessen befinden sich in [1] oder auch in [8]. Eine für unsere Zwecke sinnvoll reduzierte Version ist hingegen mit Tabelle 4.1 gegeben.

So können wir für alle Kombinationen dieser Maße die Konstanten c, durch die sie nach oben hin beschränkt werden, angeben. Es wird je ein Maß in einer Zeile betrachtet, welches nach oben hin durch die Maße in den Spalten beschränkt ist.

$\leq$	$bs(f)$	$D(f)$	$deg(f)$	$C(f)$	$s(f)$
$bs(f)$	1	1 [13]	2 [18]	1 [13]	4 [9]
$D(f)$	3 [2]	1	3 [11, 18]	2 [13]	6 [9]
$deg(f)$	3 [2]	1 [14]	1	2 [2]	2 [9]
$C(f)$	2 [13]	1 [13]	3 [11, 18]	1	5 [2, 9]
$s(f)$	1 [13]	1 [13]	2 [9]	1 [13]	1

Tabelle 4.1: Polynomielle Beziehungen zwischen den Komplexitätsmaßen mit dazugehörigen Quellen [1, 8].

Aus der Tabelle können wir somit für jedes Komplexitätsmaß eine Reihe von Gleichungen aufstellen. Allein für die Block-Sensitivität haben wir aus der Tabelle

$$bs(f) \leq bs(f),\ bs(f) \leq D(f),\ bs(f) \leq deg(f)^2,\ bs(f) \leq C(f) \text{ und } bs(f) \leq s(f)^4$$

für jede beliebige Boole'sche Funktion f. Nicht alle dieser Relationen sind direkt ersichtlich. Es lohnt sich aber trotzdem beispielhaft einen dieser Beweise kurz zu präsentieren.

Satz 4.2. *Für jede beliebige Boole'sche Funktion f gilt laut Tabelle 4.1 die Beziehung*

$$bs(f) \leq D(f).$$

Beweis. (von Satz 4.2, erstmals erschienen in [13])
Angenommen f hat Dimension n und eine Block-Sensitivität von m. Dann existieren m disjunkte Blöcke $B_1, \ldots, B_m$ für die f jeweils sensitiv ist. Das bedeutet, dass bei dem Kippen aller Bits, dessen Indizes in einem Block vorkommen, auch der Wert für f kippen würde. Ein deterministischer Decision Tree T müsste aus jedem dieser Blöcke mindestens eine Variable abfragen. Damit wäre innerhalb jedes Blockes der Wert einer Variablen fest bestimmt. Dadurch dürften diese Blöcke nicht mehr als Ganzes kippen, da dies sonst ein Widerspruch zum Ergebnis der Anfrage an T wäre. Sobald kein Block mehr existiert, der kippen könnte, kann somit auch f nicht mehr kippen. Deshalb ist auch der Wert von f erst danach fest bestimmbar. Also muss der Decision Tree mindestens so viele Anfragen machen, wie es sensitive Blöcke gibt. □

Interessanterweise wurde vor Kurzem in [19] ein noch engerer Zusammenhang zwischen Block-Sensitivität und dem Grad des repräsentierenden Polynoms beschrieben.

Nach der im Beweis dieser Arbeit vorgestellten Argumentation würde sich also auch die Separation von Block-Sensitivität und Sensitivität verbessern. Laut [19] gelte somit die Gleichung

$$bs(f) \leq \sqrt{2/3}\, s(f)^4 + 1,$$

welche aber hier nicht weiter aufgearbeitet wird. Abgesehen von alldem gibt es noch deutlich mehr Komplexitätsmaße, die in dieser Arbeit nicht genauer aufgeführt werden. Nennenswert wäre aber zum Beispiel der Grad des approximierenden multilinearen Polynoms (genannt $\widetilde{deg}(f)$) [14]. Außerdem sind Decision Tree-Modelle die quantenmechanische oder randomisierte Techniken nutzen ebenfalls Teil dieser Familie [2].

4.2 Einflüsse auf die Quantum Complexity

Das Komplexitätsmaß $D(f)$ haben wir, nicht zuletzt durch Kapitel 2.3.1, mit Decision Trees und deren Berechnungen verknüpft. Anstatt uns dafür immer Bäume zu konstruieren können wir die Decision Tree Complexity auch als deterministische Query Complexity verstehen. Bei dieser wird sich anstatt eines Baumes immer ein 'Orakel' vorgestellt. Dieses Orakel gibt bei Anfrage (engl. query) eines bestimmten Bits den dazugehörigen Wahrheitswert aus. Letztendlich ist dieser Vorgang identisch zu den Anfragen an den Decision Tree.

Gemessen wird die Anzahl der Anfragen an ein Orakel, die zur Berechnung einer Funktion benötigt werden. Sprechen wir also von deterministischer Query Complexity, dann ist dies komplett identisch zur Decision Tree Complexity und beides wird mit $D(f)$ bezeichnet.

Außerdem hilft das Denken mit Orakeln dabei ein quantenmechanisches Modell zu verstehen, in dem an Stelle von Bits nun Qubits angefragt werden. Auch wenn Modelle von Quanten-Decision Trees existieren, würde die Herleitung dieser hier zu ausschweifend sein. Somit wird eine genaue Einführung der quantenmechanischen Hintergründe hier nicht gegeben, ist aber zum Beispiel in [12] nachzulesen. Wichtig ist lediglich zu wissen, dass wir uns die Anzahl angefragter Bits (Qubits) merken, die benötigt werden, um eine Funktion f zu berechnen. Diese Anzahl nennen wir dann die (Quantum) Query Complexity.

Definition 4.3. *Wir bezeichnen mit $Q_E(f)$ die Anzahl an Anfragen von Qubits an ein Orakel, die benötigt werden, um f mit Wahrscheinlichkeit 1 richtig zu berechnen. $Q_E(f)$ bezeichnet somit die* Quantum Query Complexity*.*

Eine komplett analoge Definition existiert ebenfalls für ein Komplexitätsmaß, welches die Boole'sche Funktion nur mit einer niedrigeren Wahrscheinlichkeit berechnet. Hierbei spricht man von einem beschränktem Fehler (engl. bounded error) bei der Berechnung von f.

Definition 4.4. *Wir bezeichnen mit $Q_2(f)$ die Anzahl an Anfragen von Qubits an ein Orakel, die benötigt werden, um f mit beschränktem Fehler, also einer Wahrscheinlichkeit von 2/3, richtig zu berechnen. $Q_2(f)$ bezeichnet somit die* Bounded Error Quantum Query Complexity.

Auch diese Modelle der Komplexität wurden untersucht und insbesondere mit ihrem deterministischem Pendant $D(f)$ in Relation gesetzt. Im Rahmen dessen wurde gezeigt, dass $D(f)$ von den beiden neuen Maßen $Q_E(f)$ und $Q_2(f)$ polynomiell nach oben hin beschränkt wird. Demnach benötigt man also oft für die selbe Funktion mit Quanten-Algorithmen weniger Anfragen zur Auswertung als es mit herkömmlichen Algorithmen nötig ist.

Satz 4.5. *[1] Für jede beliebige Boole'sche Funktion $f : \{0,1\}^n \longrightarrow \{0,1\}$ gilt, dass*

$$D(f) \in \mathcal{O}(Q_E(f)^4)$$

und

$$D(f) \in \mathcal{O}(Q_2(f)^4).$$

Insbesondere die zweite Beziehung ist Ergebnis einer Forschungsarbeit, die von den Ergebnissen von Huang rund um die Sensitivity Conjecture inspiriert ist [1]. Es wird sich eine ähnliche Beweistechnik zu Nutze gemacht, wie sie auch Huang selbst nutzte. Außerdem wird im Rahmen dessen noch ein weiteres Komplexitätsmaß definiert, welches ebenfalls eng mit der von Huang benutzten Beweismethode verknüpft ist.

Für dieses Komplexitätsmaß werden Boole'schen Funktion als eine Art von Subgraphen des Hypercubes interpretiert. Denn ein Hypercube besitzt als Knotenmenge alle möglichen Boole'schen Eingaben $\{0,1\}^n$. Daraus lässt sich die 'spektrale Sensitivity' ableiten, ein Maß welches sehr an die Betrachtungen von Huang erinnert. Eine Konstruktion des dazugehörigen 'Sensitivity Graphen' lässt sich sehr gut mit einem Beispiel nachvollziehen.

Beispiel 4.6. Sei f unsere Boole'sche Funktion der Dimension $n = 3$. Für jede Eingabe $x = (x_1x_2x_3) \in \{0,1\}^3$ definieren wir, dass $f(x) = 1$ gilt, genau dann, wenn $(x_1 \wedge x_2) \vee x_3$ als wahr evaluiert wird.

Wir stellen den Graphen des 3-dimensionalen Hypercubes erst einmal ohne Kanten auf und markieren alle Knoten die zu einer erfüllenden Belegung führen (für die also $f(x) = 1$ gelten würde). Alle unmarkierten Knoten führen dementsprechend zu einer nicht-erfüllenden Belegung und damit zu $f(x) = 0$. Als erfüllend kommen nur die Eingaben $(001), (011), (101), (110)$ und (111) in Frage.

Kanten werden nun wieder zwischen Knoten eingesetzt, wenn diese einen unterschiedlichen Wert für $f(x)$ besitzen. Jeder 0-Knoten erhält nur Kanten des Hypercubes, die auch zu einem 1-Knoten führen. So ergibt sich unser Sensitivity Graph, bezeichnet mit G_f, wie in Abbildung 4.1.

Die spektrale Sensitivity, genannt $\lambda(f)$, lässt sich nun aus dem Sensitivity Graphen G_f ableiten. Hierfür benötigt man die Adjazenzmatrix des Sensitivity Graphen. In

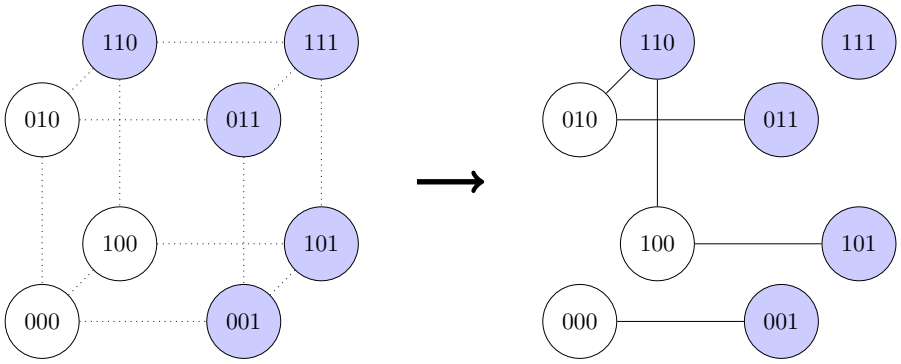

Abbildung 4.1: Konstruktion des Sensitivity Graphen G_f für f

der Originalquelle wird die spektrale Sensitivity mithilfe von Matrixnormen definiert. Um die Verbindung zu Huangs Beweis besser zu wahren, kann man sich die spektrale Sensitivity aber auch einfach als den größten Eigenwert der Adjazenzmatrix von G_f vorstellen. Insbesondere kann man an dem Sensitivity Graphen die Sensitivität $s(f)$ ablesen. Die Knoten bilden mögliche Eingaben und die Kanten stehen für sensitive Bits. Demnach erhalten wir die Formeln

$$s(f) = \Delta(G_f) \text{ und } \lambda(f) = \lambda_1(G_f),$$

und sehen damit wie sich die Konstruktion von G_f lohnen kann. Es wird sehr gut mit ihnen veranschaulicht, wo genau die Funktion noch sensitiv auf die Eingaben reagiert, und wo nicht.

Des Weiteren stellt sich heraus, dass auch die spektrale Sensitivity ein Komplexitätsmaß ist, welches mit allen bereits genannten verwandt ist. Es ist also gut möglich, dass dieses Komplexitätsmaß als eine erste Anlaufstelle genutzt werden kann, wenn Beweise ähnlich dem von Huang geführt werden können.

Außerdem trug die spektrale Sensitivity im großen Maße dazu bei, die Ergebnisse aus Satz 4.5 zu belegen [1]. Diese sind weiter so zu interpretieren, dass es möglich ist, durch quantenmechanische Berechnungsmodelle polynomiell schnellere Berechnungen Boole'scher Funktionen durchzuführen. Zwar klingt dies zunächst ernüchternd, jedoch müssen wir uns klar machen, dass wir bisher immer mit *totalen* Boole'schen Funktionen gearbeitet haben. Das heißt, wir hatten immer für jede mögliche Eingabe auch eine definierte Ausgabe. Formell schrieben wir, dass

$$f : \{0,1\}^n \to \{0,1\}$$

eine Boole'sche Funktion sei. Partielle Funktionen arbeiten hingegen auf Teilmengen von $\{0,1\}^n$. In einer kleinen Bemerkung aus [1] wird erwähnt, dass der mögliche Speedup durch Quantenberechnungen für diese partiellen Funktionen theoretisch deutlich höher und zum Beispiel auch exponentiell sein kann. Nichtsdestotrotz wurden durch spektrale Betrachtungen Boole'scher Funktionen neue Einsichten in die Berechnung eben dieser Funktionen mit quantenmechanischen Methoden gewonnen.

4.3 PRAM-Modell und die Sensitivity

Um die Komplexität eines Algorithmus zu charakterisieren, benötigt man ein theoretisches Modell eines Rechners, der diesen ausführen soll. Traditionell sind dafür zum Beispiel Turingmaschinen geeignet. Jedoch gibt es ein weiteres Rechnermodell, anhand dessen man seine Betrachtungen durchführen kann. Bei einer Random Access Machine (kurz RAM), beziehungsweise der allgemeineren Variante der Parallel Random Access Machine (kurz PRAM) handelt es sich ebenfalls um ein abstraktes Rechnermodell, geeignet für komplexitätstheoretische Betrachtungen.

Weil die Arbeit sich nicht fundamental mit Konzepten der PRAMs beschäftigt, werden sie nur kurz eingeführt. Alternativ ist eine Definition auch in [3] zu finden. Wichtig ist zu wissen, dass eine PRAM eine beliebige Menge Prozessoren $P_1, P_2, \ldots$ besitzt. Jeder Prozessor P_i besitzt rein theoretisch eine beliebige Anzahl an Registern welche für interne Rechnungen nutzbar sind. Außerdem gibt es eine Menge globaler Register $C_1, C_2, \ldots$, auf die ein jeder Prozessor zugreifen darf. In diesen muss auch am Ende der Berechnung ein Ergebnis anliegen. Die Variante mit mehreren Prozessoren ist insbesondere nützlich für die Charakterisierung von parallelen Algorithmen.

Die Arbeitsweise einer PRAM lässt sich in Zyklen bestehend aus drei simplen Schritten einteilen, nachdem die Eingabe auf die globalen Register verteilt worden sind.

1. Jeder Prozessor P_i darf ein globales Register lesen.
2. Jeder Prozessor P_i darf Berechnungen auf seinen verfügbaren Daten durchführen.
3. Jeder Prozessor P_i darf in ein globales Register schreiben.

Offensichtlich ist es möglich, dass bei paralleler Verarbeitung von Daten Konflikte auftreten können. Um das Verhalten bei dem gleichzeitigen Lesen oder Schreiben mehrerer Prozessoren zu definieren, betrachtet man unterschiedliche Versionen von PRAMs.

- **CREW:** Concurrent Read. Exclusive Write.
  - Gleichzeitiges Lesen. Exklusives Schreiben.
- **EREW:** Exclusive Read. Exclusive Write.
  - Exklusives Lesen. Exklusives Schreiben.
- **CRCW:** Concurrent Read. Concurrent Write.
  - Gleichzeitiges Lesen. Gleichzeitiges Schreiben.

Folgend werden lediglich PRAMs betrachtet, die mit der CREW-Variante arbeiten.

Definition 4.7. *Sei f eine beliebige Boole'sche Funktion. Wir bezeichen mit*

$$\mathrm{CREW}(f)$$

die Anzahl an Zyklen, die ein CREW-PRAM benötigt um f zu berechnen. Dabei besitzt dieser CREW-PRAM eine beliebige Anzahl Prozessoren und Registerzellen.

Insbesondere im Kontext der PRAMs wurde auch die Sensitivität von Boole'schen Funktionen weitgehend betrachtet. Hierfür wurde diese noch oft als die *'critical complexity'* (dt. kritische Komplexität) bezeichnet. Dabei wurde in [3] gezeigt, dass die Sensitivität ein Maß dafür sein kann, wie viele Schritte ein PRAM zur Berechnung einer Funktion mindestens benötigt.

Satz 4.8. *[3] Sei f eine beliebige Boole'sche Funktion. Dann gilt für die Konstante $\alpha = \frac{1}{2}(5 + \sqrt{21})$, dass*

$$\mathrm{CREW}(f) \geq \log_\alpha s(f).$$

Dieses Resultat festigt die Sensitivität weiter als ein sehr nützliches Komplexitätsmaß. Weitere Resultate aus [13] zeigen außerdem auch Verbindungen zwischen den CREW PRAMS und Maßen wie der Block-Sensitivität oder der Decision Tree Tiefe auf.

Satz 4.9. *[13] Sei f eine beliebige Boole'sche Funktion. Dann gilt für die Konstante $\alpha = \frac{1}{2}(5 + \sqrt{21})$, dass*

$$\mathrm{CREW}(f) \geq \log_\alpha bs(f).$$

Außerdem gilt, dass

$$\mathrm{CREW}(f) \leq \log_2 D(f).$$

Demnach können wir mit der (Block)-Sensitivität angeben, wie viele Schritte zur Berechnung mindestens notwendig sind. Mithilfe der Decision Tree Tiefe können wir die maximal benötigte Anzahl an Schritten abschätzen.

Durch die aus Kapitel 4.1 bekannte enge Verwandtschaft zwischen Block-Sensitivität und Decision Tree Tiefe können wir also sagen, dass $\mathrm{CREW}(f)$ sowohl nach oben, als auch nach unten durch $\log\ bs(f)$ und $\log\ D(f)$ beschränkt ist. Dafür lohnt es sich die obere Schranke hinsichtlich der Block-Sensitivität zu berechnen.

Beispiel 4.10. Sei f eine beliebige Boole'sche Funktion. Erneut ist $\alpha = \frac{1}{2}(5 + \sqrt{21})$. Außerdem erinnern wir uns mit unserer Tabelle 4.1 an die Eigenschaft $D(f) \leq bs(f)^3$. Mit dem vorherigen Satz 4.9 erhalten wir die Gleichung

$$\begin{aligned}\mathrm{CREW}(f) \leq \log_2 D(f) &= \frac{\log_\alpha D(f)}{\log_\alpha 2} \\ &\approx 2.26\ \log_\alpha D(f) \leq 2.26\ \log_\alpha bs(f)^3 \\ &= 6.78\ \log_\alpha bs(f).\end{aligned}$$

Damit ist $\mathrm{CREW}(f)$ sowohl nach oben, als auch nach unten hin, durch $log\ bs(f)$ beschränkt. Es gilt also bis auf Rundungen die Gleichung

$$\log_\alpha bs(f) \leq \mathit{CREW}(f) \leq 6.78\ \log_\alpha bs(f).$$

Analog kann dies auch für die Decision Tree Tiefe oder eines der anderen Komplexitäsmaße gezeigt werden.

Bisher war mit Satz 4.8 bekannt, dass CREW(f) durch die Sensitivität nur nach unten hin beschränkt ist. Ähnlich wie in Beispiel 4.10 könnte man versuchen eine obere Schranke in Abhängigkeit von $log\ s(f)$ zu finden. Diese Fragestellung wäre jedoch äquivalent zur Sensitivity Conjecture. Durch den Beweis der Conjecture ist dies also sehr leicht möglich.

Beispiel 4.11. Finden wir also nun die obere Schranke für CREW(f) in Abhängigkeit von log $s(f)$. Erneut ist $\alpha = \frac{1}{2}(5 + \sqrt{21})$ und f eine beliebige Boole'sche Funktion.

$$\begin{aligned} \text{CREW}(f) &\leq 6.78\ \log_\alpha bs(f) \leq 6.78\ \log_\alpha s(f)^4 \\ &= 27.12\ \log_\alpha s(f) \end{aligned}$$

Somit ist CREW(f) ebenfalls nach oben durch die Sensitivität beschränkt und es gilt (bis auf Rundungen) die Gleichung

$$\log_\alpha s(f) \leq \mathit{CREW}(f) \leq 27.12\ \log_\alpha s(f).$$

Ohne die bestätigte Conjecture, also der polynomiellen Verwandtschaft, wäre der Zusammenhang nicht mehr linear in log $s(f)$ gewesen. Damit hat die Sensitivity Conjecture ebenfalls einen Einfluss auf ein Rechnermodell und die Analyse von parallelen Algorithmen gehabt. Außerdem zeigt sich, dass alle unsere Komplexitätsmaße, dank Huang eingeschlossen der Sensitivität, geeignet sind um CREW(f) einer Funktion abzuschätzen.

5 Zusammenfassung und Ausblick

Im Verlaufe der Arbeit wurde die Sensitivity Conjecture als ein lange ungelöstes Problem eingeführt. Die Block-Sensitivität und die Sensitivität sind Teil einer großen und immer weiter wachsenden Familie von Komplexitätsmaßen Boole'scher Funktionen. Eine Reihe der wichtigsten Maße wurden in Kapitel 2 eingeführt.

Das Beschreiben Boole'scher Funktionen lässt uns immer mehr Einblicke in die Welt der Berechenbarkeit und der Effizienz vieler Algorithmen generieren. Demnach hat die Studie Boole'scher Funktionen auch in moderner Zeit nicht an Relevanz verloren. Neue Berechnungsmodelle geben auch heutzutage noch viel Aufschluss über das Mögliche und Unmögliche in der Komplexitätstheorie.

Die zwei Maße rund um die Sensitivität scheinen nur einen kleinen Bruchteil des Gebiets auszumachen. Aber dennoch schlagen sie mit dem Beweis der Sensitivity Conjecture interessante Brücken, die vorher nicht unbedingt bekannt waren.

Als besonders bemerkenwert ist die Beweisführung rund um die Sensitivity Conjecture hervorzuheben, welche in Kapitel 3 aufgearbeitet wurde. Ein derart kurzer und dennoch korrekter Beweis eines solch großen Theorems kommt nur sehr selten vor. Die Idee einer clever gewählten 'Pseudo'-Adjazenzmatrix hat durch den Beweis der Sensitivity Conjecture bei vielen forschenden Informatikern oder Mathematikern Eingang in ihre Beweistechniken gefunden. Insbesondere innerhalb der Kombinatorik und Graphentheorie ist ihr Einfluss erkennbar. So wurde zum Beispiel ein Rahmen für generalisierte 'Pseudo'-Adjazenzmatrizen in [6] geschaffen. Aber auch das Resultat von Huang selbst wurde auf größere Familien von Graphen, wie zum Beispiel den Hamming-Graphen, verallgemeinert [4]. Dies wurde unter anderem auch von Huang selbst im Schlusswort seines Beweises vorgeschlagen.

Außerdem wurden nach Huangs Vorbild auch neue Beweise innerhalb der Komplexitätstheorie geführt. Einige Anwendungen und Folgen die aus seiner Arbeit entspringen werden in Kapitel 4 erarbeitet. Außerdem erwähnt Huang noch weiter, dass die Lücke zwischen Sensitivität und Block-Sensitivität eventuell deutlich kleiner sein könnte, als es hier beweisen wurde. Somit kann man auch weiter auf neue Ergebnisse innerhalb der Informatik gespannt bleiben.

Resümierend lässt sich also sagen, dass mit dem Beweis der Sensitivity Conjecture ein wahrer Durchbruch in der Komplexitätstheorie, aber auch der Graphentheorie geschehen ist. Als solcher wird er vermutlich noch sehr lange viele Wissenschaftler oder auch Studierende beschäftigen. So würde sich als fortführende Arbeit ein umfangreicher Überblick über Boole'sche Komplexitätsmaße lohnen. Schließlich existieren

neben den in dieser Arbeit genannten Maßen noch viele weitere, die alle jeweils ihre Anwendungsbereiche besitzen.

Außerdem wäre es interessant Betrachtungen zu Funktionen durchzuführen, welche engere Verbindungen zwischen den Maßen suggerieren. So wurde zum Beispiel in [16] eine Familie von Funktionen vorgeschlagen, welche die Gleichung

$$bs(f) \leq s(f)^2$$

erfüllen. Offensichtlich ist dies ein engerer Zusammenhang als er von Huang bewiesen werden konnte. Aufbauend auf diesen Funktionen lassen sich mögliche Ansätze für weiterführende Forschungen erarbeiten.

Aber auch das Aufarbeiten einer der zahlreichen anderen Beweise innerhalb dieses Teilbereichs der Komplexitätstheorie kann lohnenswert sein um tiefere Einblicke zu gewinnen.

Literatur

[1] Scott Aaronson u. a. „Quantum Implications of Huang's Sensitivity Theorem“. In: *Electronic Colloquium on Computational Complexity (ECCC)* 27 (2020), S. 66.

[2] Harry Buhrman und Ronald de Wolf. „Complexity measures and decision tree complexity: a survey“. In: *Theoretical Computer Science* 288.1 (2002), S. 21–43.

[3] Stephen Cook, Cynthia Dwork und Rudger Reischuk. „Upper and Lower Time Bounds for Parallel Random Access Machines without Simultaneous Writes“. In: *SIAM J. Comput.* 15.1 (1986), S. 87–97. ISSN: 0097-5397.

[4] Dingding Dong. *On Induced Subgraphs of the Hamming Graph*. 2019. arXiv: `1912.01780 [math.CO]`.

[5] Steve Fisk. „A very short proof of Cauchy's interlace theorem for eigenvalues of Hermitian matrices“. In: *American Mathematical Monthly* 112.2 (2005), S. 118.

[6] Chris Godsil, Maxwell Levit und Olha Silina. *Graph covers with two new eigenvalues*. 2020. arXiv: `2003.01221 [math.CO]`.

[7] Craig Gotsman und Nathan Linial. „The equivalence of two problems on the cube“. In: *Journal of Combinatorial Theory, Series A* 61.1 (1992), S. 142–146.

[8] Pooya Hatami, Raghav Kulkarni und Denis Pankratov. „Variations on the Sensitivity Conjecture“. In: *Theory of Computing* 1.1 (2011), S. 1–27.

[9] Hao Huang. „Induced subgraphs of hypercubes and a proof of the Sensitivity Conjecture“. In: *Annals of Mathematics* 190.3 (2019), S. 949–955.

[10] Stasys Jukna. *Boolean Function Complexity: Advances and Frontiers*. Bd. 27. Algorithms and Combinatorics. Springer Publishing Company, Incorporated, 2012. ISBN: 3642245072.

[11] Gatis Midrijanis. *Exact quantum query complexity for total Boolean functions*. 2004. arXiv: `quant-ph/0403168 [quant-ph]`.

[12] Michael A. Nielsen und Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. 10th. USA: Cambridge University Press, 2011. ISBN: 1107002176.

[13] Noam Nisan. „CREW PRAMS and Decision Trees“. In: *Proceedings of the Twenty-First Annual ACM Symposium on Theory of Computing*. STOC '89. USA: Association for Computing Machinery, 1989, S. 327–335. ISBN: 0897913078.

[14] Noam Nisan und Mario Szegedy. „On the Degree of Boolean Functions as Real Polynomials“. In: *Computational Complexity* 4 (Jan. 1995).

[15] George Ostrouchov. „Parallel computing on a hypercube: an overview of the architecture and some applications“. In: *Proceedings of the 19th Symposium on the Interface of Computer Science and Statistics*. American Statistical Association, März 1987, S. 27–32.

[16] David Rubinstein. „Sensitivity vs. block sensitivity of Boolean functions“. In: *Combinatorica* 15 (1995), S. 297–299.

[17] Gilbert Strang. *Introduction to Linear Algebra*. Fourth. Wellesley, MA: Wellesley-Cambridge Press, 2009.

[18] Avishay Tal. „Properties and Applications of Boolean Function Composition“. In: *Proceedings of the 4th Conference on Innovations in Theoretical Computer Science*. ITCS '13 (2013), S. 441–454.

[19] Jake Wellens. *Relationships between the number of inputs and other complexity measures of Boolean functions*. 2020. arXiv: `2005.00566 [math.CO]`.