Mark Cieliebak

Attribut-effizientes Lernen in Klassen von symmetrischen Funktionen und Threshold-Funktionen

- Diplomarbeit -

Attribut-effizientes Lernen in Klassen von symmetrischen Funktionen und Threshold-Funktionen

Mark Cieliebak

Diplomarbeit
am Fachbereich Informatik
der Universität Dortmund

25. Februar 1999

Betreuer:
Prof. Dr. Ingo Wegener

Titelbild:
Mosaic II von M. C. Escher

Für Snut und Hinrich

Vorsicht Vorwort!

An dieser Stelle möchte ich mich ganz herzlich bei allen bedanken, die direkt oder indirekt zum Entstehen dieser Arbeit beigetragen haben. Insbesondere sind dies:

> Meine Eltern, die mich in meinem gesamten Studium, besonders aber im letzten Jahr, bedingungslos unterstützt haben und immer für mich da waren. Deswegen habe ich ihnen diese Arbeit gewidmet.
>
> Kai, der sich jede meiner Ideen angehört (und viele davon vernichtet) hat und immer Zeit hatte, mir zu helfen oder „mal eben“ was nachzurechnen. Ohne ihn hätte ich diese Arbeit sicher nie geschafft.
>
> Ingo, der mir dieses spannende Thema gegeben hat, stets Zeit für mich hatte und einfach ein toller Betreuer war.
>
> Carsten und Tordis, die als gute Freunde immer ein offenes Ohr hatten und auf die ich wirklich jederzeit zählen konnte.
>
> Jan und Jens, die viele Stunden mit Korrekturlesen verbracht haben.
>
> Suny, die sich meine Ideen angehört hat und nie sauer war, wenn ich stundenlang mit „ihrem“ Kai diskutiert habe.

Außerdem danke ich meiner WG für die moralische und verpflegungstechnische Unterstützung, meinem Miniseminar für die spannenden Vorträge, dem LS2 für einige interessante Anregungen, den Programmierern von Linux, TeX und LaTeX und der Stadt Zürich, in der ich immer Zuflucht fand und mich erholen und auf die Arbeit konzentrieren konnte.

Ich wünsche allen viel Spaß beim Lesen!

Dortmund, im Februar 1999 — Mark Cieliebak

Inhaltsverzeichnis

Kapitel 1

Einleitung

In einer Einleitung wird normalerweise erklärt, worum es im danach folgenden Text im Wesentlichen geht. Das soll hier natürlich auch geschehen. Inspiriert von der Arbeit von Christine Ritzmann [22][1] habe ich mich jedoch entschlossen, zwei Einleitungen zu schreiben, die sich an verschiedene Lesergruppen wenden und unabhängig voneinander sind. Das soll natürlich niemanden daran hindern, beide Teile zu lesen.
Der erste Teil ist für Leute gedacht, die (fast) keine Kenntnisse in Informatik oder Mathematik haben, aber trotzdem wissen wollen, worum es in meiner Arbeit geht. Ich versuche, dies an einem anschaulichen Beispiel zu erklären.
Die zweite Einleitung ist „klassisch-wissenschaftlich", d. h. hier findet die Leserin[2] eine Einführung in die Problemstellung, grundlegende Definitionen und Notationen und eine kurze Zusammenfassung der wesentlichen Resultate.

[1] Eine mathematische Diplomarbeit, die sehr spannend zu lesen ist. Sie wurde mit dem „Preis zur Förderung der sprachlichen Qualität wissenschaftlicher Arbeiten" der GHF ausgezeichnet.

[2] Ich halte den Ausdruck „die Leserin bzw. der Leser" für umständlich und „die/der LeserIn" für unlesbar. Deswegen werde ich in der Arbeit abwechselnd die männliche und weibliche Form verwenden.

1.1 Einleitung für Jedermann und Jedefrau

Bernd: *Hey, was ist denn das für ein Buch?*

Sonja: *Bitte? Ach so, mal sehen: Viele Formeln, fast keine Abbildungen, ein Titelbild von Escher ... das kann eigentlich nur die Diplomarbeit von Mark sein. Ja genau, steht ja auch vorne drauf: „Attribut-effizientes Lernen in Klassen von symmetrischen Funktionen und Threshold-Funktionen“ von Mark Cieliebak.*

Bernd: *Ach, das ist so schrecklicher Informatik-Kram, oder? Unmengen an Theorie, die kein Mensch versteht! Er hat mal versucht, mir zu erklären, worum es geht, aber ich hab' kein Wort verstanden.*

Sonja: *Naja, stimmt schon ein bißchen, aber ganz so schlimm ist es nun auch wieder nicht. Paß auf, ich erklär's dir.*

Bernd: *Ach vergiß es, ich hab' doch keine Ahnung von Mathe!*

Sonja: *Brauchst du auch erstmal nicht, wir kommen fast ohne Mathe aus. Stell dir vor, du arbeitest in einer Fabrik, die Gasleitungen und Schläuche herstellt. Eines Tages kommt dein Chef zu dir und sagt:*

> *„Es hat gestern bei den Schläuchen einen Produktionsfehler gegeben. Von der Tagesproduktion, das waren genau 16384 Schläuche, sind einige defekt. Aus unseren Produktionsdaten wissen wir, daß genau 16 Schläuche undicht sind. Natürlich können wir die so nicht ausliefern. Leider wissen wir aber nicht, welches die defekten sind. Gehen Sie ins Lager, überprüfen Sie alle Schläuche und finden Sie die fehlerhaften.“*

Bernd: *Was ist das denn für ein komisches Beispiel? Jede normale Firma würde sich einen Teufel darum scheren, wenn da 16 kaputte Schläuche ausgeliefert würden.*

Sonja: *Und was ist, wenn das Teile für die Raumfahrt sind? Dann ist es entscheidend, daß alle Schläuche absolut dicht sind.*

Bernd: *Und woher weiß mein Chef, daß genau 16 Schläuche kaputt sind?*

Sonja: *Nimm einfach an, daß eine der Maschinen für genau eine Stunde falsch eingestellt war, so daß das Material nicht ganz in Ordnung war. Dann weiß dein Chef natürlich, wieviele undichte Schläuche in dieser Zeit produziert worden sind. Wenn die aber schon im Lager gelandet und mit den intakten ver-*

mischt worden sind, hast du die Situation aus meinem Beispiel.[3]

Bernd: *Hmmm.*

Sonja: *Zugegeben, das Ganze ist etwas konstruiert, aber das ist leider manchmal so, wenn man einfache Beispiele für Ergebnisse aus der Theorie sucht. Und ich will dir an dem Beispiel ja auch nur anschaulich erklären, worum es in der Arbeit geht. Ich behaupte nicht, daß das jeden Tag genau so angewendet wird.*

Bernd: *Ist ja schon gut. Also nehmen wir an, ich forsche nach 16 undichten Leitungen.*

Sonja: *Als erstes überlegst du dir, wie du einen defekten Schlauch erkennen kannst. Dafür brauchst du ein Dichtigkeits-Testgerät. Das kann zum Beispiel eine Pumpe sein, mit der in dem Schlauch ein relativ hoher Druck erzeugt wird, indem viel Luft hineingepumpt wird. Das Gerät mißt den Druck im Schlauch, wartet einige Minuten und bestimmt anschließend erneut den Druck. Wenn dieser sich nicht verändert hat, ist der Schlauch dicht, andernfalls undicht.*[4]

Bernd: *Wow, da wäre ich ja* **nie** *drauf gekommen! Sowas machen also Theoretische Informatiker?*

Sonja: *Jaja, schon gut. Nehmen wir mal an, du hast so ein komisches Gerät. Diesen kompletten Vorgang mit Luft pumpen, messen, warten und messen will ich mal „testen“ nennen, um nicht immer alles aufzählen zu müssen. Außerdem soll das Gerät eine Lampe haben, die nach einem Test leuchtet, wenn der Schlauch dicht ist.*

Bernd: *Das heißt, um einen Schlauch zu überprüfen, schließe ich das Gerät an und schaue, ob die Lampe leuchtet. Bleibt sie aus, ist der Schlauch Schrott.*

Sonja: *Ja genau. Du kannst jetzt natürlich ins Lager gehen und anfangen, die Schläuche einzeln zu testen. Wenn du viel Glück hast, findest du direkt mit den ersten Tests genau die 16 undichten Leitungen, d. h. bei den ersten 16 Tests bleibt die Lampe aus. Dann brauchst du auch genau 16 Tests.*

Bernd: *Das ist irgendwie klar, oder?*

Sonja: *Ja, aber was passiert, wenn du Pech hast?*

Bernd: *Dann muß ich halt etwas öfter testen. Ist aber doch nicht schlimm, soviel wird's schon nicht sein.*

---

[3] Die meisten der folgenden Überlegungen funktionieren übrigens auch dann, wenn man die Anzahl der undichten Teile nicht genau kennt.

[4] Es gibt natürlich noch viele weitere Möglichkeiten, einen Schlauch zu testen.

Sonja: *Laß uns mal schaun: Da liegen 16384 Schläuche vor dir, von denen genau 16 undicht sind. Die anderen* $16384 - 16 = 16368$ *Leitungen sind also intakt. Jetzt nehmen wir mal an, die ersten 15 Schläuche, die du testest, sind alle kaputt.*

Bernd: *Hast du nicht gesagt, ich hätte Pech? Wenn ich 15 undichte Schläuche gefunden habe, bin ich doch schon fast fertig!*

Sonja: *Ja, aber nicht ganz. Du mußt ja noch die 16. defekte Leitung finden. Jetzt kann es aber sein, daß die nächsten 16367 Schläuche, die du testest, alle in Ordnung sind. Dann liegen noch zwei Schläuche vor dir, die du noch nicht überprüft hast, und du weißt, daß einer davon undicht und der andere dicht ist. Also mußt du noch einen von beiden testen, um zu entscheiden, welcher in Ordnung ist.*

Bernd: *Hey, das ist aber irgendwie extrem unwahrscheinlich, daß ich soviele dichte Leitungen nacheinander finde, oder? Außerdem hab' ich meistens Glück.*

Sonja: *Ja gut, aber es* **kann** *dir passieren. Insgesamt hast du dann* $15 + 16367 + 1 = 16383$ *Tests gemacht. Wenn dann das Gerät noch für jeden Test mehrere Minuten braucht ...*

Bernd: *Uups, das ist viel. Bis dahin hätte ich sicher schon gekündigt. Aber wie gesagt, das ist auch extrem unwahrscheinlich.*

Sonja: *Ja, aber möglich ist es trotzdem. Und pessimistisch, wie die Welt nun mal ist, interessiert man sich sehr oft nur dafür, was dir im schlimmsten Fall passieren kann. Und genau das werden wir auch tun. Das heißt, ich werd' dir gleich ein Verfahren erklären, mit dem du relativ schnell die undichten Leitungen finden kannst. So ein Verfahren nennt man auch Suchstrategie. Eine Suchstrategie sagt dir einfach, welche Schläuche du in welcher Reihenfolge testen mußt.*

Bernd: *Es gibt also verschiedene Suchstrategien?*

Sonja: *Ja klar, und die möchte man natürlich miteinander vergleichen. Dazu schaut man sich an, wieviele Tests du mit einer Suchstrategie machen mußt, wenn du extremes Pech hast. Das ist die Laufzeit der Strategie im schlimmsten Fall. Im Fach-Chinesisch heißt das „worst-case-Laufzeit“. Wenn du nun verschiedene Suchstrategien mit ihren Laufzeiten kennst, kannst du sie vergleichen und entscheiden, welche die beste ist.*

Bernd: *Also ist für unsere Strategie diese komische Laufzeit genau 16383 Tests.*

Sonja: *Ja genau.*

Bernd: *Okay, ich glaube, das ist klar. Und jetzt? Jetzt lese ich die*

Diplomarbeit hier und finde eine bessere Strategie?

Sonja: *Nicht so hastig, du kannst ja erstmal selber nachdenken.*

Bernd: *Hmmm ...*

Sonja: *Ich helf' dir mal ein bißchen. Was passiert, wenn du mehrere Schläuche verbindest und dann mit deinem Gerät testest?*

Bernd: *Keine Ahnung, das hängt davon ab, wie ich die verbinde.*

Sonja: *Nehmen wir mal an, die haben so luftdichte Kupplungen. Und außerdem soll das Testgerät in der Lage sein, beliebig viele Schläuche gleichzeitig zu testen.*[5]

Bernd: *Dann leuchtet die Lampe, wenn alle getesteten Schläuche dicht sind.*

Sonja: *Richtig. Umgekehrt bleibt die Birne dunkel, wenn mindestens einer der verbundenen Schläuche undicht ist. Und genau das ist der Trick. So kannst du nämlich relativ viele Schläuche gleichzeitig testen.*

Bernd: *Also wenn die Lampe leuchtet, sind alle Leitungen, die ich gerade teste, in Ordnung. Die kann ich dann einfach weglegen. Aber wenn sie nicht leuchtet, dann hab' ich doch nichts davon, oder?*

Sonja: *Doch. Nehmen wir mal an, du hast 32 Leitungen verbunden und die Lampe bleibt dunkel. Dann ist also mindestens einer der Schläuche undicht, möglicherweise auch mehr. Wie kannst du nun einen von diesen defekten Schläuchen finden?*

Bernd: *Keine Ahnung.*

Sonja: *Dazu unterbrichst du die Leitungsreihe genau in der Mitte, also nach dem 16. Schlauch, und hast danach eine linke und eine rechte Hälfte mit jeweils 16 Schläuchen. Du testest mit deinem Gerät die linke Hälfte der Leitungen. Falls die Lampe dunkel bleibt, muß mindestens ein undichter Schlauch in dieser Hälfte sein. Falls die Lampe leuchtet, sind alle Leitungen links in Ordnung. Dann weißt du aber, daß mindestens ein undichter Schlauch in der rechten Hälfte liegen muß, da ja von allen 32 Leitungen mindestens eine undicht war. Nehmen wir mal an, die Lampe leuchtet für die linke Hälfte. Dann betrachtest du nur noch die 16 Schläuche in der rechten Hälfte, von denen mindestens einer kaputt ist. Die anderen Leitungen vergißt du sofort. Du unterbrichst die 16 Schläuche wieder in der Mitte und testest die linken 8 Leitungen. Nehmen wir an, die Lampe leuchtet nicht. Dann ist unter den*

[5] Praktische Probleme wie z. B. Meßungenauigkeiten scheinen Sonja nicht zu interessieren. Diese sind aber auch nicht wichtig für das Verständnis der Suchstrategie.

linken 8 Schläuchen mindestens ein kaputter. Wieder untersuchst du nur diese 8 Leitungen weiter, halbierst diese, findest 4 Schläuche mit mindestens einem defekten, halbierst diese, findest 2 Schläuche mit mindestens einem defekten, halbierst diese, findest einen Schlauch mit mindestens einem defekten, und bist fertig.

Bernd: *Ah ja?*

Sonja: *Klar, der letzte Schlauch, der noch vor dir liegt, muß defekt sein.*

Bernd: *Sooo klar find' ich das nicht.*

Sonja: *Warte, ich mal dir mal ein Bildchen.*

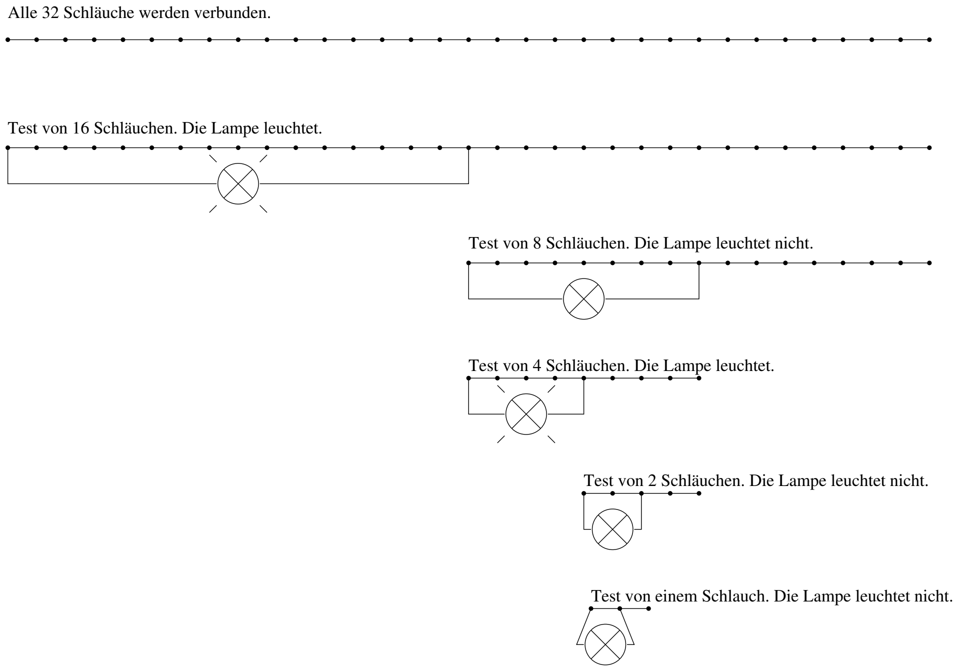

Abbildung 1.1: Halbierungssuche nach einem undichten Schlauch

Am Anfang hast du 32 Schläuche, von denen mindestens einer defekt ist. Dann wird jeweils eine Hälfte der Schläuche getestet und in dem Teil weitergesucht, der mindestens eine defekte Leitung enthält. Sobald nur noch ein Schlauch übrig ist, hörst du natürlich auf mit dem Halbieren. Der letzte Schlauch muß dann defekt sein.

Bernd: *Das ist aber kompliziert!*

Sonja: *Auf den ersten Blick schon, aber wenn du das mal selber versucht hast, ist es total einfach. Fang vielleicht einfach mit*

64 Schläuchen an und nimm an, die 17. Leitung und die 34. sind defekt. Mit dem fortgesetzten Halbieren solltest du nach ein paar Tests genau das 17. Kabel übrigbehalten. Versuch's ruhig mal.

Bernd: *Ach laß mal, das glaub' ich dir auch so. Aber so finde ich doch nur einen undichten Schlauch, und nicht alle, oder?*

Sonja: *Ja, aber der Rest ist einfach. Du fängst damit an, daß du alle 16384 Schläuche verbindest.*

Bernd: *Und das soll einfach sein? Da hab ich ja ewig zu tun.*

Sonja: *Laß uns das gleich diskutieren, okay? Also, unter diesen 16384 Schläuchen sind natürlich auch alle 16 undichten. Du führst unsere Halbierungssuche durch, mit der du einen undichten Schlauch findest. Den schmeißt du einfach weg und ersetzt ihn durch einen neuen, der dicht ist.*[6] *Danach hast du wieder 16384 Schläuche, aber darin sind nur noch 15 undichte. Mit der Halbierungssuche findest du wieder einen defekten und ersetzt diesen sofort. Danach sind es nur noch 14 undichte Leitungen.*

Bernd: *Ach so, und das mache ich solange, bis ich alle undichten Leitungen gefunden habe.*

Sonja: *Ja genau. Du mußt also 16 Halbierungssuchen durchführen, um alle defekten Schläuche zu finden.*

Bernd: *Das dauert doch ewig! Am Anfang muß ich alle Schläuche miteinander verbinden, dann startet jede der Halbierungssuchen immer mit 16384 Leitungen und anschließend muß ich alle Schläuche wieder verbinden.*

Sonja: *Das hängt davon ab, was du mit „ewig“ meinst. Natürlich ist es ein bißchen Arbeit, die ganzen Schläuche immer korrekt miteinander zu verbinden. Aber wenn dieses Testgerät extrem lange für einen Test braucht - wir hatten mehrere Minuten gesagt - dann ist der Aufwand für das Verbinden der Schläuche eher gering im Vergleich zum Testen. Und genau das ist die wesentliche Eigenschaft bei dem Modell, um das es in dieser Arbeit geht: Es gibt irgendwelche Tests, die extrem aufwendig im Vergleich zu allem anderen sind. Deswegen sucht man Strategien, die mit sehr wenigen Tests auskommen. Ob dafür andere Arbeiten, wie bei uns das Verbinden der Schläuche, notwendig sind, interessiert nicht. Es werden* **nur** *die Tests gezählt, die für eine Strategie notwendig sind.*

[6] Eigentlich muß der undichte Schlauch nicht ersetzt werden, aber Sonja hat scheinbar keine Lust zu erklären, wie man die Halbierungssuche für eine ungerade Anzahl Schläuche durchführt. Das geht aber im Wesentlichen genauso.

Bernd: *Das ist aber extrem realitätsfremd, oder?*

Sonja: *Nicht so schlimm, wie es im Moment klingt. Es gibt tatsächlich Beispiele aus der Realität, in denen das zutrifft. Nimm mal an, du hast sehr viele Blutproben vor dir stehen und willst prüfen, welche davon Antikörper enthalten. Du kannst natürlich jede Probe einzeln testen. Du kannst aber auch ein Gemisch aus mehreren Proben erstellen und dann darin nach Antikörpern suchen. Dabei entpricht eine infizierte Blutprobe einem undichten Schlauch in unserem Beispiel, das Zusammenkippen der Blutproben ist genau das Verbinden der Schläuche und eine Blutuntersuchung entspricht einem Test mit unserem Dichtigkeits-Testgerät. Das Zusammenkippen mehrerer Blutproben ist natürlich sehr einfach im Vergleich zu einer Blutuntersuchung.*[7]

Bernd: *Hmm, aber wenn ich nur eine infizierte Blutprobe habe und die mit ganz vielen anderen vermische, dann kann ich irgendwann die Antikörper nicht mehr finden, weil es viel zu wenig sind.*

Sonja: *Ja gut, auch da gibt es noch Probleme in der Praxis, aber es ist zumindest eine echte Anwendung.*

Bernd: *Okay, schon gut. Aber trotzdem muß ich doch mit diesen Halbierungssuchen ewig viele Tests machen, oder?*

Sonja: *Eben nicht! Für eine Halbierungssuche brauchst du halt nur sehr wenig Tests. Laß uns mal ausrechnen, wieviele das sind.*

Bernd: *Ha, das riecht nach Mathe!*

Sonja: *Ja, aber nur ganz wenig. Also, bei den 32 Schläuchen, wie oft mußtest du die halbieren und eine Hälfte testen, bis nur noch ein Kabel übrigblieb?*

Bernd: *Mal sehen. Das war*

$$\begin{aligned} 32 &= 2*16 \\ 16 &= 2*8 \\ 8 &= 2*4 \\ 4 &= 2*2 \\ 2 &= 2*1 \end{aligned}$$

Das sind genau 5 Halbierungen gewesen.

Sonja: *Genau. Jetzt stell dir vor, du startest mit 1024 Leitungen.*

[7] Dieses Beispiel stammt tatsächlich aus der Realität. Im Zweiten Weltkrieg wurden in den USA alle Soldaten auf Syphilis untersucht und hierbei entsprechende Strategien entwickelt (vgl. [9]). Warum Sonja dieses Beispiel nicht verwendet, ist unklar. Vielleicht ist es ihr zu „blutrünstig“?

Bernd: *Dann muß ich 512, 256, 128, 64, 32, 16, 8, 4, 2 und 1 Schläuche testen. Das sind genau 10 Tests. Irgendwie kommen mir die Zahlen bekannt vor.*

Sonja: *Das sind die Potenzen von 2. Das ist aber hier nicht so entscheidend. Wichtig ist, daß es in der Mathe die Bezeichnung* $\log_2(n)$ *gibt, den* Logarithmus zur Basis 2, *der genau angibt, wie oft ich die Zahl* n *halbieren muß, um auf die Eins zu kommen. Zum Beispiel ist* $\log_2(32) = 5$

Bernd: *Ach so, und dann ist* $\log_2(1024)$ *genau 10, weil ich 10 mal halbieren muß. Das ist ja einfach. Ich glaub', das kenne ich noch aus der Schule.*

Sonja: *Kann gut sein.*

Bernd: *Aber was ist, wenn die Zahl* n *ungerade ist? Dann kann ich doch nicht genau halbieren, oder?*

Sonja: *Stimmt, dann muß man etwas genauer rechnen, aber das ist im Moment nicht so wichtig. Es macht fast keinen Unterschied, und in meinem Beispiel sind die Zahlen immer gerade.*

Bernd: *Na, so ein Zufall!*

Sonja: *Wie viele Tests mußt du denn jetzt mit der Halbierungssuche machen, um alle 16 defekten Schläuche zu finden?*

Bernd: *Mal schauen - ich muß genau 16 Halbierungssuchen durchführen. Jede von diesen Suchen braucht* $\log_2(16384)$ *Tests. Also sind das insgesamt* $16 \cdot \log_2(16384)$ *Tests.*

Hmmm, $8192, 4096, 2048, \ldots, 1$ *- also ist* $\log_2(16384)$ *gleich 14. Insgesamt sind das dann* $16 \cdot 14 = 224$ *Tests. Das ist aber echt wenig.*

Sonja: *Ja, und das ist schon die Laufzeit im schlimmsten Fall, denn du findest ja mit jeder Halbierungssuche genau einen undichten Schlauch. Hier kommt es also nicht auf Glück oder Pech an.*[8] *Wenn du für das Verbinden der Schläuche nur sehr wenig Zeit im Vergleich zu einem einzelnen Test brauchst, dann sind diese 224 Tests doch entschieden besser als die 16383 von der einfachen Strategie, oder?*

Bernd: *Zugegeben, das ist schon echt gut.*

Sonja: *Ja, „gut" schon, aber noch nicht optimal. Es geht sogar noch besser.*

Bernd: *Echt? Wie denn das?*

Sonja: *Dafür braucht man zwei Ideen. Erstens kann man am Anfang aus allen Schläuchen 16 verschiedene Haufen mit jeweils ge-*

[8] Allerdings kann man bei dieser Strategie kein Glück haben. Man braucht immer 224 Tests.

nau 16384 : 16 = 1024 Leitungen machen und danach jeden Haufen einzeln betrachten. Dann starten die Halbierungssuchen nicht immer mit so vielen Schläuchen. Außerdem kannst man bei den Halbierungssuchen etwas geschickter sein. Was ist, wenn für eine Leitungsreihe in beiden Hälften ein defekter Schlauch ist? Dann testest du im Moment nur die linke Hälfte und suchst da weiter. Wenn du aber direkt beide Hälften testest, weißt du sofort, daß auch in der rechten Hälfte noch ein kaputter Schlauch ist, den du noch finden mußt.

Bernd: *Das hab' ich jetzt irgendwie nicht verstanden.*

Sonja: *Das ist auch nicht so wichtig. Das Entscheidende ist, daß man für diese etwas verbesserte Strategie beweisen kann, daß sie fast optimal ist. Es kann also keine wesentlich besseren Strategien geben.*

Bernd: *Wie macht man das denn?*

Sonja: *Nun, dafür braucht man ein bißchen mehr Mathe ...*

Bernd: *Halt, das gerade war schon genug!*

Sonja: *Na gut, dann erkläre ich nur die grundsätzliche Idee: Nimm an, du hast irgendeine Strategie. Dann interessiert dich, wieviele Tests du mit dieser Strategie im schlimmsten Fall machen mußt, um alle defekten Schläuche zu finden. Bei unserer ersten Strategie waren das zum Beispiel 16383, bei unserer zweiten nur noch 224 Tests. Mit ein wenig Rechnen kann man nun sogenannte „untere Schranken" bestimmen. Die geben an, wieviele Tests* jede *Strategie im schlimmsten Fall mindestens machen muß.*

Bernd: *Häh? Was ist das?*

Sonja: *Ich erklär's nochmal anders: Eine untere Schranke für ein Problem hängt nicht von einer speziellen Suchstrategie ab, sondern nur von der Aufgabe, die du lösen willst. Das ist sozusagen ein Maß dafür, wie schwierig die Aufgabe überhaupt ist. Ich versuch's mal mit einem ganz anderen Beispiel. Stell dir vor, du suchst eine möglichst kurze Fahrtroute von Dortmund nach Zürich. Dann kannst du dir einen Atlas nehmen, verschiedene Strecken raussuchen und deren Länge bestimmen. Eine untere Schranke für diese Längen ist aber ganz bestimmt der Luftlinienabstand zwischen den beiden Städten, das heißt du wirst keine noch so geschickte Fahrtroute finden, die kürzer ist als die Luftlinie.*

Bernd: *Die verschiedenen Fahrtrouten sind also die unterschiedlichen Strategien, die Schläuche zu testen, und die Luftlinie ist diese komische untere Schranke?*

Sonja: *Ja genau. Die untere Schranke sagt dir also, wie lang* **jede** *Strecke mindestens ist, oder in unserem Beispiel wieviele Tests* **jede** *Strategie im schlimmsten Fall stellen muß. Für unser Beispiel ist eine untere Schranke von* $16 \cdot \log_2(\frac{16384}{16}) = 160$ *Tests bekannt. Das bedeutet, daß jede Strategie, egal wie geschickt und toll sie ist, im schlimmsten Fall mindestens 160 Tests machen muß, um alle undichten Schläuche zu identifizieren.*

Bernd: *Dann sind unsere 224 Tests doch gar nicht so schlecht, oder?*

Sonja: *Das stimmt zwar, aber die Informatiker möchten halt besser als „nicht so schlecht" sein. Die wollen immer optimale Lösungen.*

Bernd: *Und die gibt es immer?*

Sonja: *Natürlich, es gibt für jede Aufgabe - zumindest in der Informatik - eine optimale Lösung.*[9] *Es gibt ja auch eine kürzeste Fahrtroute von Dortmund nach Zürich. Aber es gibt auch immer zwei Probleme: Erstens mußt du die optimale Lösung, also in unserem Fall die kürzeste Strecke, finden. Und zweitens mußt du erkennen und danach beweisen können, daß die Lösung optimal ist.*

Bernd: *Das hört sich sehr schwierig an.*

Sonja: *Ist es leider auch. Du mußt dafür ja zeigen, daß niemand jemals eine so gute Idee haben kann, daß seine Strategie besser ist als deine. Dafür kann man gerade die unteren Schranken verwenden. Wenn die Laufzeit deiner Strategie sehr nah bei einer unteren Schranke liegt, dann ist sie auch sehr gut. Leider sind die Beweise oft sehr kompliziert und theoretisch, und meistens kann man nur für sehr eingeschränkte Probleme gute untere Schranken zeigen. Zum Beispiel gibt es Sortierverfahren, von denen man zeigen kann, daß sie annähernd optimal sind. Dazu muß man sich überlegen, daß man jedes Suchverf...*

Bernd: *Hey, nicht noch ein neues Beispiel! Du wolltest mir doch eigentlich nur erklären, was in der Diplomarbeit steht, und nicht eine Vorlesung über Informatik halten. Oder hat das was mit der Arbeit zu tun?*

Sonja: *Ach so, nein, eigentlich nicht direkt. Sorry. Also gut, nochmal zu unserem Beispiel. Laß mich nochmal etwas abstrakter erklären, worum es bei den Schläuchen geht. Es sind* n *Objekte vorhanden, von denen bekannt ist, daß genau* k *von diesen*

[9] Das ist natürlich eine sehr gewagte These, die stark davon abhängt, was Sonja unter „optimal" versteht.

Objekten defekt sind.

Bernd: *Was sind denn n und k?*

Sonja: *Das sind Variablen. So eine Variable n steht quasi für irgendeine beliebige Zahl. Du kannst dir vorstellen, daß da einfach die echten Zahlen stehen, wenn du willst. Mit den Variablen kann man aber einfacher rechnen und allgemeine Formeln herleiten, in die man später die konkreten Zahlen einsetzt.*

Bernd: *Dann sind in unserem Beispiel $n = 16384$ Schläuche vorhanden und darunter $k = 16$ undichte.*

Sonja: *Genau. Die Aufgabe besteht darin, diese k defekten Objekte zu finden. Dazu darf man Tests durchführen, die ein positives oder negatives Resultat haben können.*

Bernd: *Und ein Test ist das Verbinden von mehreren Leitungen mit unserem Testgerät.*

Sonja: *Ja. Wie so ein Test aussieht und wovon das Testergebnis abhängt, kann nun variieren. In unserem Beispiel mit den Schläuchen war das Ergebnis genau dann positiv, wenn mindestens ein kaputtes Kabel in dem Test vorkam. Man kann sich nun noch Tausende weitere Testformen ausdenken. In der Arbeit von Mark kommen drei Tests vor:*

- *Ein Test ist genau dann positiv, wenn mindestens τ defekte Objekte in der Testmenge sind. Der Wert τ ist dabei irgendeine feste Zahl und wird als Schwellwert oder Threshold bezeichnet.*
- *Ein Test ist genau dann positiv, wenn sich die Anzahl der defekten Objekte nicht durch eine Zahl q teilen läßt. Die Zahl q wird dann als Modulus bezeichnet.*
- *Ein Test ist genau dann positiv, wenn genau ρ defekte Objekte in der Testmenge sind. Dabei ist ρ irgendeine beliebige Anzahl.*

Bernd: *Das sind aber komische Tests. Kannst du die etwas erklären?*

Sonja: *Eigentlich nicht, da kenne ich leider keine vernünftigen Beispiele.*

Bernd: *Wie, gar nicht?*

Sonja: *Ich kann vielleicht versuchen, die Tests zu veranschaulichen, aber das hat dann* **nichts** *mit realen Anwendungen zu tun. Das soll dir nur helfen, besser zu verstehen, was mit den Tests gemeint ist.*

Wenn du bei der ersten Testart den Wert τ auf Eins setzt, hast du genau unser Schlauchbeispiel. Das heißt, ein Test ist *positiv, wenn mindestens ein undichter Schlauch mitgetestet wird. Wenn das τ jetzt größer als Eins ist, zum Beispiel 50, kannst du dir vorstellen, daß das Testgerät so ungenau ist, daß schon aus mindestens 50 Schläuchen Luft entweichen muß, damit das Gerät den Druckabfall feststellt. Dann leuchtet die Lampe nicht und du weißt, daß es mindestens 50 defekte Schläuche gibt.*

Bernd: *Na gut, prinzipiell verstehe ich das, aber das Beispiel ist wirklich schlecht. Wenn das Gerät so ungenau ist, kann es doch passieren, daß die Lampe nicht leuchtet und trotzdem sind nur 49 undichte Schläuche in der Reihe, oder?*

Sonja: *Stimmt. Wie gesagt, das ist kein wirklich gutes Beispiel.*

Bernd: *Allerdings!*

Sonja: *Bei der zweiten Testart, die mit dem Modulus, kann ich das vielleicht erstmal für $q = 2$ erklären. Aber ich muß dich warnen, dieses Beispiel ist auch nicht viel besser!*

Bernd: *Hmmm.*

Sonja: *Der zweite Test ist für $q = 2$ genau die Frage, ob die Anzahl der defekten Objekte, die getestet werden, gerade ist oder nicht. Nimm mal an, du hast ein großes Netz von Computern und willst darin Nachrichten verschicken. Du kannst für deine Nachricht angeben, von wo aus sie über welche Computer wohin verschickt werden soll. Was passiert nun, wenn einige Computer defekt sind[10] und die Nachricht immer rückwärts weitergeben, das heißt die Reihenfolge der Buchstaben einfach umdrehen? Wenn so ein kaputter Computer die Nachricht „Nietzsche ist tot." erhält, schickt er „.tot tsi ehcszteiN" weiter. Deine Aufgabe ist es nun, die defekten Computer zu finden. Du kannst dafür Nachrichten über einige Zwischencomputer an dich selber schicken und prüfen, wie sie ankommen. Wenn ein Text korrekt ankommt, war entweder kein defekter Computer dazwischen oder deren Anzahl war gerade. Wenn die Nachricht aber rückwärts ankommt, war die Anzahl der defekten Computer auf dem Weg ungerade. Ein Test ist hier also das Verschicken einer Nachricht über bestimmte Computer. Ein Test ist positiv, wenn die Nachricht korrekt ankommt, also die Anzahl der defekten Computer gerade und somit ein Vielfaches von $q = 2$ war.*

Bernd: *Das heißt ich kann nur sehen, ob die Anzahl der defekten Computer gerade war oder nicht und muß so alle kaputten*

[10] Hier meint Sonja wohl eher „schlecht programmiert", was ja gelegentlich vorkommen soll.

Geräte finden. Das ist knifflig.

Sonja: *Stimmt. Wenn q nun größer als 2 wird, geht es darum, daß sich nicht genau 2, sondern halt q defekte Objekte gegenseitig neutralisieren. Dafür gibt's vielleicht in der Chemie oder Biologie Beispiele, aber wie gesagt, da weiß ich keine konkreten Anwendungen.*

Bernd: *Und die dritte Klasse? Laß mich raten, auch da gibt's keine guten Beispiele?*

Sonja: *Nein, und da versuche ich's auch gar nicht erst. Das ist einfach das Problem, daß du ein Testgerät hast, das nur bei genau ρ defekten Objekten reagiert, also nicht bei weniger und nicht bei mehr. Dafür fällt mir nun wirklich nichts ein.*

Bernd: *Na gut, ich glaub', das verstehe ich jetzt auch so. Und was macht Mark jetzt mit diesen Tests?*

Sonja: *Naja, in der Arbeit geht es im Wesentlichen darum, für diese drei Arten von Testformen optimale oder zumindest gute Strategien anzugeben, um die defekten Objekte zu identifizieren. Man kann die drei Tests übrigens etwas mathematischer als Threshold-, Modulo- und Anzahl-Funktionen bezeichnen.*

Bernd: *Und für diese komischen Funktionen gibt Mark in seiner Arbeit Strategien an.*

Sonja: *Ja genau. Für die Threshold-Funktionen kann er beweisen, daß die Strategien, die er angibt, immer optimal*[11] *sind.*

Für die Modulo-Funktionen schafft er das nicht ganz, da stimmt das nur, wenn der Modulus q nicht allzu groß wird.

Bernd: *Und die Anzahl-Funktionen?*

Sonja: *Tja, das ist schwer zu sagen. Bei den Anzahl-Funktionen zeigt er, daß es dafür eine schnelle Strategie gibt, wenn es ein Lottosystem mit sehr wenig Tippreihen gibt.*

Bernd: *Was für ein System?*

Sonja: *Es geht zum Beispiel darum, wie viele Reihen man tippen muß, um garantiert genau vier Richtige beim Lotto zu haben. Wenn man so ein System kennt, dann hat man auch direkt eine Strategie für die Anzahl-Funktionen.*

Bernd: *Und solche Lottosysteme gibt es?*

Sonja: *Natürlich gibt es solche Systeme. Die Frage ist, mit wie wenig Tippreihen man auskommt. Und da hat Mark leider nichts gefunden. Es gibt zwar viele wissenschaftliche Artikel und auch einige schöne Spielsysteme, aber er hat keins gefunden,*

[11] Hier übertreibt Sonja leider ein wenig. Die Strategien sind nur asymptotisch optimal, d. h. sie treffen die unteren Schranken bis auf konstante Faktoren.

das genau seinen Anforderungen entspricht. Er konnte allerdings auch nicht sehr intensiv suchen, da er diese Beziehung zwischen seiner Arbeit und den Lottosystemen erst kurz vor der Abgabe untersucht hat.

Bernd: *Schade eigentlich, oder?*

Sonja: *Ja, aber vielleicht liest ja mal jemand die Arbeit, der Ahnung von Kombinatorik hat.*

Bernd: *Wenn überhaupt jemand die Arbeit liest. Wieso hat er eigentlich gerade diese drei Tests ausgewählt?*

Sonja: *Oh, das hatte wohl mehrere Gründe. Einerseits beruht die ganze Arbeit auf einem Artikel von Uehara, Tsuchida und Wegener von 1997 mit dem Titel* „Attribute-efficient learning of disjunction, parity, and threshold functions“. *Darin werden verschiedene Arten von Tests untersucht, unter anderem die Schlauchtests und die Modulo-Funktionen, die genau zwischen gerade und ungerade entscheiden können. Die Autoren geben dort für verschiedene Testarten optimale Strategien an.*

Bernd: *Das heißt sie können beweisen, daß die Strategien optimal sind?*

Sonja: *Ja genau. Mark hat in seiner Arbeit einige von diesen Ideen aufgegriffen und verallgemeinert.*

Sonja: *Und die anderen Gründe?*

Sonja: *Gründe? Ach so, ja. Ist dir an unserem Beispiel mit den Schläuchen etwas aufgefallen?*

Bernd: *Eigentlich nicht. Was denn?*

Sonja: *Die Tests sind unabhängig von der Reihenfolge. Du kannst die Schläuche verbinden, wie du willst, die Lampe bleibt aus, wenn mindestens ein undichter Schlauch in der Reihe ist, egal an welcher Stelle.*

Bernd: *Stimmt.*

Sonja: *Und das ist bei den anderen Tests genauso. Das Ergebnis hängt nie von der Anordnung der defekten und intakten Testobjekte ab, sondern nur, wieviele es von der jeweiligen Sorte gerade gibt. Sowas nennt man in der Mathe „symmetrische“ Funktionen. Ein Ziel der Forschung ist es zu erkennen, welche symmetrischen Funktionen einfach und welche schwierig sind. Und so ein bißchen soll die Diplomarbeit wohl dazu beitragen.*

Bernd: *Ach so. Warte mal, dieses „symmetrisch“ hab' ich doch schon irgendwo gesehen ... ah ja, da im Titel:*

„Attribut-effizientes Lernen in Klassen von symme-

trischen Funktionen und Threshold-Funktionen".

Sonja: *Genau. Das müßtest du jetzt eigentlich verstehen. „Attribut-effizientes Lernen" heißt, daß man Strategien finden will, die die defekten von den intakten Objekten trennen und dabei sehr nah an den unteren Schranken sind. Wenn man das „übersetzt", geht es also darum, für Funktionen oder Tests, deren Ergebnisse nicht von der Reihenfolge der getesteten Objekte abhängen, möglichst beweisbar gute Strategien zu finden, die die defekten von den intakten Objekten trennen.*

Bernd: *Und warum steht das nicht genau so vorne drauf, damit man das versteht?*

Sonja: *Das klingt halt nicht so schön wissenschaftlich, oder?*

Bernd: *Das stimmt. Apropos Titel, was hat eigentlich dieses komische Bild auf dem Umschlag zu bedeuten?*

Sonja: *Das Bild ist von M. C. Escher und heißt „Mosaic II". Mark zeigt im Kapitel über Anzahl-Funktionen, daß eine sehr spezielle Art von Anzahl-Funktionen mit Hilfe von sogenannten „Pflasterungen" untersucht werden kann. Du kannst dir das so vorstellen, daß eine Pflasterung gerade ein Mosaic-Steinchen ist. Dann möchte man gerne Pflasterungen finden, die so gut zusammenpassen, daß keine Lücken entstehen. Genau das ist aber auch das Thema des Bildes, nur mit etwas merkwürdige Mosaic-Steinchen.*

Bernd: *Puh, ich glaub', mir reicht's erstmal.*

Sonja: *Ja, das war auch sehr viel. Hast du denn jetzt 'ne Idee, worum es geht?*

Bernd: *Ja, ich glaub schon.*

Sonja: *Und, hast du Lust, mal ein bißchen drin zu lesen?*

Bernd: *Ich weiß nicht, ob ich das verstehe. Das ist doch eigentlich nur Mathe, oder?*

Sonja: *Das stimmt leider.*

Bernd: *Na mal schaun. Vielen Dank erstmal, daß du mir erklärt hast, worum es überhaupt geht.*

Sonja: *Kein Problem.*

1.2 Wissenschaftliche Einleitung

Notation

Natürlich versuche ich, mich in dieser Arbeit an allgemein übliche Notationen zu halten. Trotzdem möchte ich einige Schreibweisen explizit angeben, um Unklarheiten zu vermeiden. Im Anhang findet sich ein Verzeichnis der Symbole, die ich verwende. Außerdem steht dort eine Liste von Variablen, deren Bedeutung sich im Lauf der Arbeit nicht ändert. Ich verwende einige Aussagen und Definitionen, die aus der Literatur stammen. Diese sind durch entsprechende Referenzen am Rand gekennzeichnet.
Soweit nicht explizit eine Basis angegeben wird, bezeichnet $\log a$ den Logarithmus von a zur Basis 2 und $\ln a$ den natürlichen Logarithmus von a.
Für mehrfache Indizierung verwende ich eine Indexliste, d. h. ich schreibe $f_{a,b}$ statt $(f_a)_b$. Dies wird insbesondere bei Funktionen häufig vorkommen.
Für einen Vektor $x \in \{0,1\}^n$ gibt x_i die i-te Koordinate von x an. Mit $|x|_1$ wird die Anzahl Einsen und mit $|x|_0$ die Anzahl Nullen in x bezeichnet.
Für $x, y \in \mathbb{R}^n$ bezeichnet $<x, y>$ das Skalarprodukt,

$$<x, y> := \sum_{i=1}^{n} x_i \ y_i$$

Mit $<x, y>_r$ wird das Skalarprodukt der ersten r Koordinaten bezeichnet, also

$$<x, y>_r := \sum_{i=1}^{r} x_i \ y_i$$

Wenn A eine Teilmenge von B ist, dann bezeichnet $\bar{A}$ das Komplement von A in B, d. h. $\bar{A} := B \backslash A$. Dabei ergibt sich die Menge B normalerweise aus dem Zusammenhang, andernfalls werde ich sie explizit benennen.
Für zwei Mengen A und B bezeichnet $A \oplus B := (A \backslash B) \cup (B \backslash A)$ die symmetrische Mengendifferenz, das sind alle Objekte, die in genau einer der beiden Mengen liegen.
Für eine Menge A bezeichnet $\mathcal{P}(A)$ die Potenzmenge von A, d. h. die Menge aller Teilmengen von A.
Wenn $M \subseteq \mathbb{R}$ eine beliebige Menge ist und $a \in \mathbb{R}$ ein Skalar, dann ist $a + M$ die Verschiebung der Menge M um a, d. h. $a + M := \{a + x \mid x \in M\}$. Es ist klar, daß dann $a+(b+M) = (a+b)+M$ und $a+(M \cup N) = (a+M) \cup (a+N)$ für beliebiges $a, b \in \mathbb{R}$ und $M, N \subseteq \mathbb{R}$ gilt.
Um Formeln verkürzen zu können, verwende ich $[n]$, um die natürlichen Zahlen von 1 bis n zu bezeichnen. Es gilt also $[n] := \{1, \ldots, n\}$. Ein abgeschlossenes Intervall in $\mathbb{R}$ werde ich mit $[u, v] := \{x \in \mathbb{R} \mid u \leq x \leq v\}$ bezeichnen, ein offenes entsprechend mit $]u, v[$.
Um die Laufzeit von Algorithmen anzugeben, verwende ich die übliche O-Notation. Ich gebe hier nur die formale Definition an, eine ausführliche Diskussion findet sich z. B. in [14].

[14, S. 10] **Definition 1.1: O-Notation**

Seien $f, g : \mathbb{N} \to \mathbb{R}^+$.

- $f = O(g) \;:\Longleftrightarrow\; \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+ : \forall n \geq n_0 \; f(n) \leq c \cdot g(n)$
- $f = \Omega(g) \;:\Longleftrightarrow\; g = O(f)$.
- $f = \Theta(g) \;:\Longleftrightarrow\; f = O(g)$ *und* $g = O(f)$.

Eine Variablenbelegung $x \in \{0,1\}^n$ kann auch als Menge A_x beschrieben werden, die genau die Indizes i enthält, für die die Variable x_i mit Eins belegt ist. Dann gilt $A_x := \{i \in [n] | \; x_i = 1\}$. Ich werde überwiegend diese Schreibweise verwenden, da sie den Vorteil hat, daß für Variablenbelegungen auch Mengenoperationen zulässig sind. Die Mengen $E_i = \{i\}$ entsprechen den Variablenbelegungen, in denen die i-te Variable auf Eins gesetzt ist.

Attribut-effizientes Lernen mit Queries

In dieser Arbeit untersuche ich, wie Klassen boolescher Funktionen mit Anfragen (*Queries*[12]) gelernt werden können. Bei diesem Lernmodell ist eine Klasse $F \subseteq \{f : \{0,1\}^n \to \{0,1\}\}$ von booleschen Funktionen gegeben, die sowohl einem Lehrer als auch einem Schüler bekannt ist. Der Lehrer wählt eine Funktion $f_0 \in F$ aus. Der Schüler hat nun die Aufgabe, die Funktion f_0 zu lernen, indem er Fragen stellt.[13]

Eine Frage ist eine Belegung der Variablen $x_1, \ldots, x_n$ mit den Werten Null oder Eins. Für eine solche Frage x antwortet der Lehrer mit $f_0(x)$, d. h. er verrät dem Schüler den Funktionswert, den die gesuchte Funktion für diese Variablenbelegung liefert. Der Schüler versucht, die Funktion mit möglichst wenig Fragen zu identifizieren, d. h. er stellt solange Anfragen, bis nur noch eine einzige Funktion aus F mit allen Fragen konsistent ist. Dies muß dann die gesuchte Funktion f_0 sein. Der Schüler darf seine Fragen adaptiv stellen, d. h. jede neue Frage darf von den bisher erhaltenen Antworten abhängen.

Da der Schüler die Klasse F kennt, kann er sich bereits im voraus eine Strategie von Fragen überlegen, mit der er jede Funktion aus F lernen kann. Eine Strategie ist dabei um so besser, je weniger Fragen der Schüler im schlimmsten Fall stellen muß. Das heißt insbesondere, daß *nur* die Anzahl der Fragen untersucht wird, die gestellt werden. Alle anderen Berechnungen, die notwendig sind, um die Strategie durchzuführen, werden nicht betrachtet.[14]

[12] Ich werde die Begriffe *Frage, Anfrage, Test* und *Query* synonym verwenden.

[13] Eigentlich identifiziert der Schüler die Funktion f_0 eindeutig, was ich nicht unbedingt als „Lernen" bezeichne. Trotzdem werde ich diesen Begriff verwenden, da er sich in der Literatur etabliert hat.

[14] Natürlich ist dies eine starke Einschränkung, da die „normalen" Berechnungen möglicherweise sehr umfangreich sind. Es läßt sich aber dadurch rechtfertigen, daß die Fragen in der Praxis sehr aufwendige Tests (z. B. Bluttests oder chemische Untersuchungen) sein können. Dann ist die Anzahl dieser Tests das entscheidende Kriterium für den zeitlichen Aufwand.

Eine untere Schranke für eine Funktionenklasse F gibt an, wieviele Fragen mindestens notwendig sind, um jede Funktion in F lernen zu können. Für jede Strategie muß also die Anzahl der Anfragen im worst-case mindestens so groß sein wie jede untere Schranke für die Funktionenklasse. Eine Strategie ist dann asymptotisch optimal, wenn sie bis auf einen konstanten Faktor mit einer unteren Schranke übereinstimmt.[15]

Bevor ich eine allgemeine untere Schranke für beliebige Funktionenklassen angebe, zeige ich an einem kleinen Beispiel, wie eine Strategie aussehen kann. Sei $F = \{x_1 \vee x_2, x_1 \wedge x_2, \neg x_1\}$ eine Menge von drei Funktionen auf zwei Variablen. Wie kann eine Strategie für diese Klasse aussehen? Der Schüler kann z. B. als erstes die Anfrage $(1, 0)$ stellen, d. h. die erste Variable wird auf Eins und die zweite auf Null gesetzt. Wenn die Antwort eine Eins ist, muß der Lehrer die Funktion $x_1 \vee x_2$ ausgewählt haben, da diese als einzige Funktion aus F für diese Belegung eine Eins liefert. Dann hat der Schüler die Funktion bereits identifiziert. Falls die Antwort eine Null ist, so hat der Lehrer eine der beiden anderen Funktionen ausgewählt. Der Schüler stellt nun zum Beispiel die Anfrage $(1, 1)$. Falls die Antwort eine Eins ist, muß $x_1 \wedge x_2$ die gesuchte Funktion sein, andernfalls $\neg x_1$. Mit dieser Strategie kann der Schüler also mit höchstens zwei Anfragen die Funktion identifizieren.

Eine Strategie kann als binärer Entscheidungsbaum veranschaulicht werden. An den inneren Knoten stehen genau die Anfragen, die der Schüler stellt. Die beiden Söhne eines Knotens entsprechen den möglichen Antworten Null und Eins des Lehrers. An einem Blatt steht genau die einzige Funktion aus der Funktionenklasse, die mit allen Antworten, die auf dem Pfad von der Wurzel bis zu diesem Blatt gegeben wurden, konsistent ist. Abbildung 1.2 zeigt den Baum für die Strategie aus dem letzten Absatz.

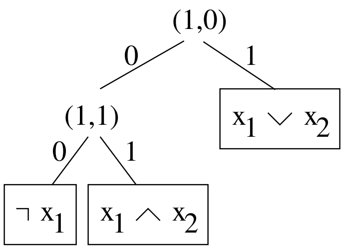

Abbildung 1.2: Binärbaum für eine einfache Strategie

Wird eine Strategie als binärer Entscheidungsbaum dargestellt, so gibt die Tiefe dieses Baumes gerade die Anzahl der Anfragen an, die die Strategie im worst-case stellt. Andererseits muß eine Strategie jede Funktion, die in der

[15] Wenn momentan keine guten unteren Schranken bekannt sind, kann es sein, daß eine asymptotisch optimale Strategie im Augenblick nicht als solche erkannt wird. Trotzdem ist sie natürlich asymptotisch optimal.

betrachteten Funktionenklasse liegt, identifizieren können. Daher muß für jede Funktion aus F mindestens ein Blatt in dem Baum existieren. Hieraus folgt direkt eine allgemeine untere Schranke für beliebige Funktionenklassen.

[24, S. 4]

Satz 1.2: Untere Schranke für allgemeine Funktionenklassen

> *Für eine Funktionenklasse F braucht jede Lernstrategie mindestens $\lceil \log |F| \rceil$ Anfragen im worst-case, um F zu lernen.*

Beweis: Eine Lernstrategie wird als binärer Entscheidungsbaum dargestellt. Dann hat dieser Baum mindestens $|F|$ Blätter. Daraus folgt die Behauptung. □

Mit diesem Ergebnis kann für jede Funktionenklasse unmittelbar eine untere Schranke angegeben werden. Dies ist für einige der Klassen, die ich betrachten werde, bereits die beste untere Schranke, die ich finden konnte. Andererseits gibt es auch Klassen, bei denen Strategien existieren, deren Laufzeit genau dieser unteren Schranke entspricht. In unserem kleinen Beispiel sind mindestens $\lceil \log 3 \rceil = 2$ Anfragen notwendig, und unsere Strategie, die im worst-case mit 2 Anfragen auskommt, ist bereits optimal.
Anfragen $a \in \{0,1\}^n$ können mit einer Anfragemenge $A_a = \{i | \, a_i = 1\}$ identifiziert werden. Dies erlaubt es, auf Anfragen Mengenoperationen durchzuführen. Der Variablenbelegung $a = (1,1,0,0,1,0,0)$ entspricht also die Menge $A_a = \{1,2,5\}$.

Für die Funktionenklassen, die ich untersuchen werde, wird der Begriff der Subfunktion benötigt. Für eine Grundfunktion $g : \{0,1\}^n \to \{0,1\}$ und eine Auswahl von Variablen $S \subseteq [n]$ entsteht die Subfunktion von g bzgl. S dadurch, daß alle nicht in S liegenden Variablen durch eine Konstante (in dieser Arbeit immer Null) ersetzt werden.

Definition 1.3: Subfunktion

> *Sei $g : \{0,1\}^n \to \{0,1\}$ ein boolesche Funktion und $S \subseteq [n]$ eine Auswahl von Variablen. Sei $c \in \{0,1\}^n$. Sei $g_{S,c}(x) := g(x')$ mit*
> $$x'_j = \begin{cases} x_j & \text{für } j \in S \\ c_j & \text{für } j \notin S \end{cases}$$
> *Dann ist $g_{S,c}$ die* **Subfunktion von g bzgl. S und c**.

Falls der Konstantenvektor c durch den Kontext klar ist, kann statt $g_{S,c}$ auch g_S geschrieben werden. Für die Grundfunktion $g(x) = x_1 \vee x_2 \vee \ldots \vee x_8$ und die Menge $S = \{1,3,4\}$ und den Vektor $c = (0,\ldots,0)$ ergibt sich

$$\begin{aligned} g_{S,c}(x) &= x_1 \vee 0 \vee x_3 \vee x_4 \vee 0 \vee 0 \vee 0 \vee 0 \\ &= x_1 \vee x_3 \vee x_4 \end{aligned}$$

Die Variablen in S heißen „wesentlich“ (*essential*), da sie das Verhalten der Funktion g_S entscheidend bestimmen. Für eine Grundfunktion g kann

die Klasse $G(k) := \{g_S | S \subseteq [n], |S| = k\}$ betrachtet werden. Diese Klasse enthält alle Subfunktionen von g mit genau k wesentlichen Variablen. Für diese Klasse kann die obige Lernaufgabe anders beschrieben werden. Die Auswahl einer Funktion aus $G(k)$ entspricht genau der Auswahl von k wesentlichen Variablen. Die Aufgabe des Schülers ist es dann, genau diese k wesentlichen Variablen zu identifizieren.[16] Dies wird als Attribut-Lernen bezeichnet, da die wesentlichen Variablen bzw. Attribute identifiziert werden sollen. Eine Strategie wird „effizient" genannt, wenn die Laufzeit bis auf einen konstanten Faktor mit einer unteren Schranke für das Lernen der Funktionenklasse übereinstimmt.[17]
Sowohl das Gebiet des allgemeinen Lernens mit Queries als auch das Gebiet des Attribut-effizienten Lernens sind nicht in dieser Arbeit erfunden worden. Es gibt zahlreiche Literatur zu beiden Themen, die ich aber hier nicht aufzählen werde.[18] Stattdessen werde ich die relevanten Literaturstellen direkt in den einzelnen Kapiteln angeben.

Ein kurzer Überblick

Was erwartet die Leserin in den einzelnen Kapiteln?
In **Kapitel 2** untersuche ich Funktionenklassen, die auf symmetrischen Funktionen auf n Variablen beruhen. Eine Funktion f ist symmetrisch, wenn der Funktionswert $f(x)$ nur von der Anzahl der Einsen in x, nicht aber von der Reihenfolge der Einsen und Nullen abhängt. Solche Funktionen lassen sich sehr kompakt durch einen Wertevektor beschreiben, der in der i-ten Koordinate genau die Ausgabe der Funktion bei i Einsen in der Eingabe enthält.
Zunächst untersuche ich den Fall, daß die Funktionenklasse auf einer einzigen symmetrischen Funktion basiert, für die der Wertevektor v und die Anzahl der wesentlichen Variablen k bekannt sind. Dann sind (in den nicht-trivialen Fällen) mindestens $\lceil \log \binom{n}{k} \rceil$ Anfragen notwendig, um diese Klasse zu lernen. Ich gebe einen Algorithmus an, dessen Laufzeit linear in der Anzahl aller Variablen ist. Ein zweiter Algorithmus mit Laufzeit $O(\frac{n}{r} + \log \binom{n}{k})$ kann für $n - k < \lfloor \frac{n}{r} \rfloor$ durchgeführt werden. Dabei gibt r die Position des ersten Wechsels zwischen Null und Eins im Wertevektor an. Hieraus und mit einem weiteren Algorithmus aus [8] ergibt sich, daß für sehr kleines oder sehr großes k und für k nahe bei $\frac{n}{2}$ bereits asymptotisch optimale Algorithmen bekannt sind.

[16] Vorausgesetzt, verschiedene Auswahlen von wesentlichen Variablen erzeugen auch verschiedene Funktionen. Dies ist aber fast immer der Fall.

[17] Das heißt aber nicht, daß eine effiziente Strategie auch ein „schneller Algorithmus" im herkömmlichen Sinne ist. Für die Klasse $F = \{f : \{0,1\}^n \to \{0,1\}\}$ aller booleschen Funktionen ist die untere Schranke 2^n (vgl. Satz 1.2). Der triviale Algorithmus, der immer alle 2^n Variablenbelegungen testet, ist also „effizient" (sogar optimal), aber gewiß nicht schnell.

[18] Eine aktuelle Zusammenstellung findet sich z. B. in [8].

Was ändert sich, wenn k oder v nicht bekannt sind? Auch für die verbleibenden drei Fälle bestimme ich untere Schranken und gebe jeweils einen in n linearen Algorithmus an. Falls beide Werte k und v unbekannt sind, ist der lineare Algorithmus bereits asymptotisch optimal.

In den nächsten drei Kapiteln betrachte ich drei spezielle Formen für den Wertevektor v. Diese Kapitel sind weitgehend unabhängig voneinander. Natürlich übertragen sich alle Ergebnisse für beliebige symmetrische Funktionen auf diese speziellen Klassen. Für die Threshold-Funktionen beginnt der Wertevektor mit t Nullen, gefolgt von $n - t + 1$ Einsen. Bei den Modulo-Funktionen hat der Wertevektor die Gestalt

$$v = (0, 1, \ldots, 1, 0, 1, \ldots, 1, 0, 1, \ldots)$$

wobei sich zwischen zwei Nullen immer gleich viele Einsen befinden. Für die Anzahl-Funktionen enthält der Wertevektor fast ausschließlich Nullen. Nur an genau einer Stelle steht eine Eins. Ich habe mich für diese drei Klassen entschieden, da sich durch die stark eingeschränkte Form der Wertevektoren Algorithmen und Sätze leicht beschreiben lassen. Andererseits sind die drei Wertevektoren so unterschiedlich (genau ein Wechsel zwischen Null und Eins, periodische Wechsel und genau ein einzelner Peak), daß sich drei vollkommen verschiedene Klassen ergeben. Vielleicht können die Ergebnisse später für eine Klassifikation genutzt werden, welche symmetrischen Funktionen einfach bzw. schwierig zu lernen sind.

In **Kapitel 3** untersuche ich die Klasse der Threshold-Funktionen mit bekanntem Thresholdwert τ. Diese geben genau dann eine Eins aus, wenn mindestens τ Einsen in der Eingabe sind. Für diese Klassen werden in [24] von Uehara, Tsuchida und Wegener für kleines k bereits asymptotisch optimale Algorithmen angegeben. Für $k \geq \frac{n}{2}$ wird dort nur der Spezialfall $\tau = \frac{n}{2}$ betrachtet. Ich verallgemeinere die dortigen Algorithmen und zeige, wie Threshold-Funktionen für beliebiges k und τ asymptotisch optimal gelernt werden können.

Die Modulo-Funktionen, die in **Kapitel 4** betrachtet werden, liefern genau dann eine Null, wenn die Anzahl der Einsen in der Eingabe ein Vielfaches des Modulus q sind. Ebenfalls in [24] werden für Parity-Funktionen, die der Spezialfall der Modulo-Funktionen mit $q = 2$ sind, asymptotisch optimale Algorithmen angegeben. Ich zeige, daß sich die unteren Schranken und Algorithmen für größeren Modulus verallgemeinern lassen. Für nicht zu großen Modulus ($q = O(\log \binom{n}{k})$) gebe ich einen asymptotisch optimalen Algorithmus an.
Für sehr großes q vermute ich, daß Modulo-Funktionen im Wesentlichen so schwierig zu lernen sind wie Anzahl-Funktionen, da sich die Wertevektoren sehr stark ähneln.

Die letzte Klasse von symmetrischen Funktionen, die ich in **Kapitel 5** untersuche, sind die Anzahl-Funktionen. Eine Anzahl-Funktion der Ordnung ρ liefert genau dann eine Eins, wenn genau ρ Einsen in der Eingabe sind. Die untere Schranke $\lceil \log \binom{n}{k} \rceil$ sowie die Algorithmen für allgemeine symmetrische Funktionen gelten auch hier. Ich gebe eine weitere untere Schranke sowie für einige Spezialfälle Algorithmen an. Außerdem zeige ich, daß das Lernen von Anzahl-Funktionen im Wesentlichen äquivalent dazu ist, ein System von (verallgemeinerten) Lottoscheinen anzugeben, mit dem man für jede mögliche Ziehung auf mindestens einem Schein genau ρ Richtige hat. Lottoprobleme werden bereits seit langem in der Mathematik untersucht, trotzdem habe ich (aus Zeitgründen) keine geeigneten Resultate gefunden. Für den Fall $\rho = 1$ gebe ich einen Algorithmus mit $O(k\binom{\log n}{\log k} + \log \binom{n}{k})$ Anfragen an (für $k \leq \frac{n}{2}$).

In **Kapitel 6** beschreibe ich einige Ideen zur Untersuchung von gewichteten Threshold-Funktionen mit Gewichten 1 und 2 und Thresholdwert t. Diese Funktionen sind nicht symmetrisch. Ich gebe untere Schranken für alle möglichen Parameterkombinationen sowie einen in n linearen Algorithmus an. Einen weiteren Algorithmus beschreibe ich für den Fall, daß der Thresholdwert t kleiner als die Anzahl der wesentlichen Variablen k ist. Dieser Algorithmus ist für einige Parameter bereits asymptotisch optimal.
Ich vermute, daß sich die Algorithmen für „normale“ Threshold-Funktionen auf den gewichteten Fall übertragen lassen.

Im **Anhang** befindet sich ein Verzeichnis der verwendeten Symbole, eine Aufstellung der Variablen, deren Bedeutung in allen Kapiteln identisch ist, eine Zusammenstellung einiger Formeln und Beweise sowie natürlich ein Literaturverzeichnis.

Kapitel 2

Symmetrische Funktionen

Mit Queries können prinzipiell beliebige Klassen boolescher Funktionen gelernt werden. Ich werde mich in weiten Teilen dieser Arbeit auf Klassen von symmetrischen Funktionen beschränken. Diese lassen sich durch einen Wertevektor beschreiben.
In diesem Kapitel werde ich zunächst einige Eigenschaften von symmetrischen Funktionen zusammenstellen. Dann untersuche ich, wie symmetrische Funktionen gelernt werden können, wenn der Wertevektor v und die Anzahl der wesentlichen Variablen k bekannt sind. Anschließend werden diese Ergebnisse für den Fall, daß v oder k unbekannt sind, verallgemeinert. Für jede dieser vier Klassen gebe ich untere Schranken und einen Algorithmus an, dessen Laufzeit linear in der Anzahl aller Variablen ist.

Definition 2.1: Symmetrische Funktion

> *Eine Funktion $f : \{0,1\}^n \to \{0,1\}$ auf $n \in \mathbb{N}$ Variablen heißt* **symmetrisch**, *wenn für $x \in \{0,1\}^n$ der Wert $f(x)$ nur von der Anzahl der Einsen in x abhängt, nicht aber von der Anordnung der Einsen und Nullen in x.*

Etwas formaler kann man auch sagen, daß eine Funktion f genau dann symmetrisch ist, wenn für jede Permutation π auf $[n]$ und für jedes $x \in \{0,1\}^n$ die Gleichung $f(x) = f(\pi(x))$ gilt.
Eine symmetrische Funktion läßt sich durch einen Wertevektor v (*value vector*) der Länge $(n+1)$ beschreiben. Dieser gibt in der i-ten Position an, was die Funktion bei i Einsen in der Eingabe ausgibt. Es gilt also $f(x) = v_{|x|_1}$.
Jede symmetrische Funktion ist durch ihren Wertevektor bereits eindeutig bestimmt. Wenn zu zwei symmetrischen Funktionen die Wertevektoren übereinstimmen, so sind die Funktionen identisch. Ich möchte hier für einige Beispiele von symmetrischen Funktionen die Wertevektoren beschreiben.
Für die Disjunktion beginnt der Wertevektor mit einer Null, gefolgt von n Einsen. Bei der Konjunktion beginnt der Vektor mit n Nullen und der letzte Eintrag ist eine Eins. Der Wertevektor der Majority-Funktion besteht aus

$\left\lfloor\frac{n+1}{2}\right\rfloor$ Nullen, gefolgt von $\left\lceil\frac{n+1}{2}\right\rceil$ Einsen. Für die Parity-Funktion enthält der Wertevektor abwechselnd Einsen und Nullen, beginnend mit einer Null. Eine Verallgemeinerung sind die Modulo-Funktionen mit Modulus q, bei denen zwischen zwei Nullen jeweils $q-1$ Einsen stehen. Bei Threshold-Funktionen mit Thresholdwert τ beginnt der Wertevektor mit τ Nullen, gefolgt von $n-\tau+1$ Einsen.
Anhand der Wertevektoren läßt sich leicht zeigen, daß es genau 2^{n+1} verschiedene symmetrische Funktionen auf n Variablen gibt.
Wenn f und g symmetrische Funktionen mit Wertevektoren v und w sind, dann sind auch die Funktionen $\neg f$, $f \vee g$ und $f \wedge g$ symmetrisch. Die Wertevektoren lauten

$$\begin{aligned} v_{\neg f} &= (\neg v_0, \ldots, \neg v_n) \\ v_{f \vee g} &= (v_0 \vee w_0, \ldots, v_n \vee w_n) \\ v_{f \wedge g} &= (v_0 \wedge w_0, \ldots, v_n \wedge w_n) \end{aligned}$$

2.1 Bekannte Parameter v und k

Zu einer symmetrischen Funktion f_v kann für eine Auswahl von wesentlichen Variablen $K \subseteq [n]$ die Subfunktion $f_{v,K}$ definiert werden. Die Menge aller Subfunktionen mit genau k wesentlichen Variablen bildet die Funktionenklasse, die hier untersucht werden soll.

Definition 2.2: $SYM_v^n(k)$

> *Seien $n \in \mathbb{N}$, $k \in \mathbb{N}_0$ mit $0 \leq k \leq n$. Sei $v \in \{0,1\}^{n+1}$ ein Wertevektor. Sei f_v die durch den Wertevektor v definierte symmetrische Funktion.*
>
> *Dann besteht die Klasse $SYM_v^n(k)$ aus allen Subfunktionen von f_v mit k wesentlichen von n Variablen. Unwesentliche Variablen werden durch Null ersetzt.*

$$\mathbf{SYM_v^n(k)} := \{f_{v,K} \mid K \subseteq [n], |K| = k\}$$

Wenn n aus dem Zusammenhang eindeutig ist, schreibe ich auch einfach $SYM_v(k)$.[1] Da Funktionen in $SYM_v(k)$ nur k wesentliche Variablen enthalten, sind von v nur die ersten $k+1$ Koordinaten relevant. Daher kann ich

[1] Die Funktionen, die in $SYM_v(k)$ liegen, sind übrigens *nicht* symmetrisch. Der Funktionswert von $f_{v,K}(x)$ hängt davon ab, wieviele wesentliche Variablen in x mit Eins belegt sind. Die Position dieser wesentlichen Variablen wird durch K festgelegt, so daß die Einsen und Nullen in x nicht beliebig vertauscht werden können, ohne den Funktionswert zu ändern.

im Folgenden stets $v \in \{0,1\}^{k+1}$ voraussetzen. Wenn $k = 0$ oder $k = n$ gilt, so ist die Lernaufgabe trivial, da es entweder nur unwesentliche oder nur wesentliche Variablen gibt. In beiden Fällen kann also $SYM_v(k)$ ohne Anfrage gelernt werden.
Ich setze im Folgenden stets

$$\boxed{n \in \mathbb{N}, 1 \leq k \leq n-1 \text{ und } v \in \{0,1\}^{k+1}}$$

voraus und gebe dies nicht mehr explizit in jedem Satz an.

Ich bestimme nun die Größe der Klasse $SYM_v(k)$. Es gibt $\binom{n}{k}$ verschiedene Möglichkeiten, die wesentlichen Variablen auszuwählen. Welche Auswahlen von Variablen verschiedene Funktionen definieren, wird im folgenden Lemma untersucht.

Lemma 2.3: Größe von $SYM_v(k)$

> *Falls $v_0 = \ldots = v_k$ gilt, so enthält die Klasse $SYM_v(k)$ genau eine Funktion.*
>
> *Andernfalls liegen genau $\binom{n}{k}$ Funktionen in $SYM_v(k)$.*

Beweis: Falls $v_0 = \ldots = v_k$ gilt, so ist die durch v definierte Funktion konstant. Die wesentlichen und unwesentlichen Variablen können nicht unterschieden werden, da sie keinen Einfluß auf den Funktionswert haben. Also enthält $SYM_v(k)$ genau eine Funktion.
Andernfalls existiert ein $r \in \{1, \ldots k\}$, so daß $v_0 = \ldots = v_{r-1} \neq v_r$. Sei o. B. d. A. $v_{r-1} = 0, v_r = 1$. Sei f die symmetrische Funktion mit Wertevektor v. Seien $A = \{i_1, \ldots, i_k\}$ und $B = \{j_1, \ldots, j_k\}$ zwei beliebige verschiedene Auswahlen von wesentlichen Variablen. Ich zeige, daß die beiden durch die Mengen A und B bestimmten Funktionen f_A und f_B unterschiedlich sind, d. h. daß es mindestens einen Vektor $x \in \{0,1\}^n$ gibt, so daß $f_A(x) \neq f_B(x)$ gilt. Sei dazu s minimal so gewählt, daß $i_1 = j_1, \ldots i_{s-1} = j_{s-1}$ und $i_s \neq j_s$ gilt.

- 1. Fall: $r \leq s$. Setze $x_{i_1}, \ldots, x_{i_{r-1}} = 1$, $x_{i_s} = 1$ und alle anderen Variablen auf Null. Dann sind in x genau r Variablen aus A auf Eins gesetzt, also ist $f_A(x) = 1$. Andererseits sind genau $r-1$ Variablen aus B auf Eins gesetzt, also ist $f_B(x) = 0$.

- 2. Fall: $r > s$. Setze $x_{i_1}, \ldots, x_{i_s}$ und weitere $r-s$ beliebige Variablen $i_j \in A$ auf Eins, alle anderen Variablen auf Null. Dann sind in x genau r Variablen aus A auf Eins gesetzt, also ist $f_A(x) = 1$. Andererseits sind höchstens $r-1$ Variablen aus B auf Eins gesetzt, da $i_s \notin B$, also ist $f_B(x) = 0$.

Also sind die Funktionen f_A und f_B unterschiedlich. Da es $\binom{n}{k}$ Möglichkeiten der Variablenwahl gibt, enthält die Klasse $SYM_v(k)$ also $\binom{n}{k}$ verschiedene Funktionen. □

Aus der unteren Schranke für allgemeine Funktionenklassen ergibt sich direkt eine untere Schranke für die Klasse $SYM_v(k)$.

Satz 2.4: Untere Schranke für $SYM_v(k)$

> *Falls $v_0 = \ldots = v_k$ gilt, so kann $SYM_v(k)$ ohne Anfrage gelernt werden.*
>
> *Andernfalls sind mindestens $\lceil \log \binom{n}{k} \rceil$ Anfragen notwendig, um $SYM_v(k)$ zu lernen. Es gilt $\lceil \log \binom{n}{k} \rceil \geq k \log \frac{n}{k}$.*

Beweis: Falls $v_0 = \ldots = v_k$ gilt, so enthält $SYM_v(k)$ genau die konstante Funktion $f \equiv v_0$, wesentliche und unwesentliche Variablen können nicht unterschieden werden.
Im zweiten Fall ergibt sich die untere Schranke direkt aus der Größe der Klasse $SYM_v(k)$ in Verbindung mit Satz 1.2. Die Abschätzung folgt aus Lemma B.1. □

In Abbildung 2.1 ist die Funktion $f(x) = x \log \frac{n}{x}$ für $n = 2^{15}$ skizziert. Es gilt $f'(x) = \log \frac{n}{x} - \log e$. Die Funktion f nimmt in $\frac{n}{e}$ ihr Maximum an. Es gilt $f(\frac{n}{4}) = f(\frac{n}{2}) = \frac{n}{2}$.
Ich gebe für die Klasse der symmetrischen Funktionen einen Algorithmus an, dessen Laufzeit linear in n ist. Die wesentliche Idee ist, zunächst mit linearer Suche eine wesentliche Variable sowie eine Variablenmenge A zu finden, die bei Hinzunahme einer wesentlichen Variable eine andere Antwort liefert als ohne. Dann werden die restlichen Variablen durch Vergrößerung bzw. Verkleinerung der Menge A um jeweils eine Variable identifiziert.

Satz 2.5: Algorithmus für $SYM_v(k)$

- *Falls $v_0 = \ldots = v_k$ gilt, so enthält $SYM_v(k)$ nur eine konstante Funktion und kann ohne Anfrage gelernt werden.*
- *Falls $v_0 \neq v_1$, so kann die gesuchte Funktion mit höchstens n Anfragen gelernt werden.*
- *Falls ein $r \in \{2, \ldots, k\}$ existiert mit $v_0 = \ldots = v_{r-1} \neq v_r$, so können die wesentlichen Variablen mit maximal $2n - r$ Anfragen identifiziert werden.*

Beweis:

- 1. Fall: $v_0 = \ldots = v_k$. Die gesuchte Funktion ist $f(x) \equiv v_0$, die Variablen können nicht unterschieden werden. Es ist keine Anfrage notwendig, da v bekannt ist.

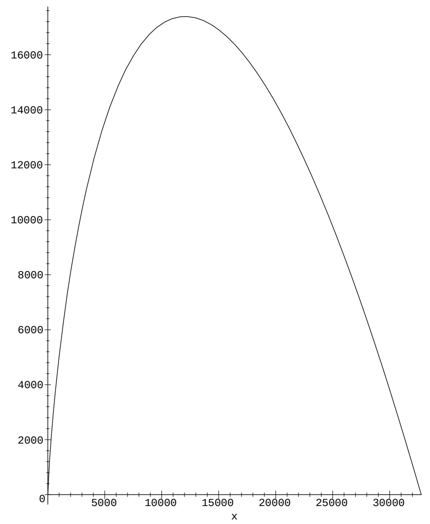

Abbildung 2.1: Die Funktion $f(x) = x \log \frac{n}{x}$ für $n = 2^{15}$.

- 2. Fall: $v_0 \neq v_1$. Sei o. B. d. A. $v_0 = 0, v_1 = 1$. Ein Algorithmus stellt die einelementigen Mengen $\{x_i\}$ mit $1 \leq i \leq n$ als Anfragen. Jede Eins als Antwort identifiziert eine wesentliche Variable. Sobald k wesentliche Variablen identifiziert sind, stoppt der Algorithmus.

- 3. Fall: $v_0 = \ldots = v_{r-1} \neq v_r$ mit $2 \leq r \leq k$. Sei o. B. d. A. $v_0 = \ldots = v_{r-1} = 0, v_r = 1$. Der folgende Algorithmus lernt die wesentlichen Variablen.

  Algorithmus 2.6:

  1. Teste solange $\{x_1, \ldots, x_j\}$ mit $r \leq j \leq n$, bis für $l = j$ eine Eins geantwortet wird. Dann ist x_l wesentliche Variable und $\{x_1, \ldots, x_l\}$ enthält genau r wesentliche Variablen.
  2. Falls $k = r = l$ gilt, so sind bereits alle wesentlichen Variablen identifiziert, STOP.
  3. Teste $\{x_1, \ldots, x_l\} \backslash \{x_i\}$ für $1 \leq i \leq l-1$. Jede Antwort Null identifiziert eine wesentliche Variable. Teste solange, bis insgesamt r wesentliche Variablen gefunden sind.
  4. Falls $k = r$ gilt, so sind bereits alle wesentlichen Variablen identifiziert, STOP.

5. Teste $\{x_1, \ldots, x_{l-1}\} \cup \{x_j\}$ für $l+1 \leq j \leq n$. Jede Antwort Eins identifiziert eine wesentliche Variable. Teste höchstens solange, bis insgesamt k wesentliche Variablen gefunden sind.

Die Anzahl Anfragen beträgt höchstens

$$\begin{aligned}(l-r+1)+(l-1)+(n-l) &= n+l-r \\ &\leq 2n-r\end{aligned}$$

□

Man kann die Algorithmen für den zweiten und dritten Fall noch verbessern. Wenn bereits eine Menge A bekannt ist, die genau r wesentliche Variablen enthält, so kann auch ein Algorithmus für die Klasse der Konjunktionen $AND(r)$ verwendet werden, um in A alle Variablen zu identifizieren. Hierfür wird in [24] ein asymptotisch optimaler Algorithmus angegeben. In beiden Algorithmen lassen sich manchmal weitere Anfragen sparen. Wenn z. B. beim ersten Algorithmus nach $n-p$ Anfragen erst $k-p$ wesentliche Variablen gefunden wurden, so müssen die restlichen Variablen $x_{n-p+1}, \ldots, x_n$ alle wesentlich sein. Ähnliches gilt beim zweiten Algorithmus.
Solche Verbesserungen sind sicher sinnvoll und notwendig, wenn ein Algorithmus tatsächlich benutzt werden soll. Leider läßt sich die worst-case-Laufzeit dieser Algorithmen, die uns hier interessiert, scheinbar hierdurch nicht wesentlich verbessern. Das will ich an einem Beispiel für den zweiten Algorithmus verdeutlichen. Angenommen, die wesentlichen Variablen sind genau die k Variablen mit den größten Indizes, also $x_{n-k+1}, \ldots, x_n$. Im ersten Schritt fragt der Algorithmus die Mengen $\{x_1, \ldots, x_j\}$ für $r \leq j \leq n$. Dies sind genau $n-r+1$ Fragen. Für nicht zu großes r ist dies bereits linear in n.
Im zweiten Algorithmus wird nicht verwendet, daß die Anzahl k der wesentlichen Variablen bekannt ist. Hiermit ließen sich einige Anfragen sparen. Andererseits kann der Algorithmus später für den Fall, daß k unbekannt ist, unverändert verwendet werden (vgl. Satz 2.12).
In Satz 2.4 wird gezeigt, daß mindestens $k \log \frac{n}{k}$ Anfragen notwendig sind, um die Klasse $SYM_v(k)$ zu lernen. Wenn $k = \frac{n}{c}$ für $c > 0$ gilt, so folgt $k \log \frac{n}{k} = \frac{n}{c} \log c$. Dieser Term ist für konstantes c linear in n, so daß in diesem Fall der Algorithmus aus Satz 2.5 bereits asymptotisch optimal ist. Für sehr kleines oder sehr großes k ist der Unterschied zwischen der bekannten unteren Schranke und der Laufzeit des linearen Algorithmus jedoch sehr groß.

In [8] wird ein Algorithmus für beliebige Klassen boolescher Funktionen angegeben, der diese mit $O(k2^k \log n)$ Anfragen lernt. Wenn die Anzahl der wesentlichen Variablen konstant ist, ist dieser Algorithmus asymptotisch optimal, da sich für $k = O(1)$ die untere Schranke $\log n$ ergibt.

Ich betrachte nun die Klasse $SYM_v(n-k')$. Die Variable k' bezeichnet dabei die Anzahl der unwesentlichen Variablen. Ich untersuche den Fall, daß k' nicht zu groß ist.

Satz 2.7: Algorithmus für $SYM_v(n-k')$ mit $k' < \lfloor \frac{n}{r} \rfloor$

> *Sei $r \leq \frac{n}{2}$ mit $v_0 = \ldots = v_{r-1} \neq v_r$. Sei $k = n - k'$ mit $k' < \lfloor \frac{n}{r} \rfloor$. Dann kann die Klasse $SYM_v(n-k')$ mit $O(\frac{n}{r}+k' \log \frac{n}{k'})$ Anfragen gelernt werden.*

Beweis: Sei o. B. d. A. $v_0 = \ldots = v_{r-1} = 0, v_r = 1$. Der folgende Algorithmus lernt alle unwesentlichen Variablen.

Algorithmus 2.8:

1. Bilde $\lfloor \frac{n}{r} \rfloor$ disjunkte Mengen der Größe r und frage diese Mengen. Dies sind mehr als k' Anfragen. Da nur k' unwesentliche Variablen existieren, muß mindestens eine Anfrage nur aus wesentlichen Variablen bestehen und eine Eins liefern. Diese enthält genau r wesentliche Variablen $w_1, \ldots, w_r$. Definiere Dummymengen $D_j := \{w_1, \ldots, w_{r-j}\}$ für $1 \leq j \leq r-1$.

   Jede Anfrage mit Antwort Null wird eine Gewinnerin.

2. Bilde die Menge A, die alle Variablen enthält, die im ersten Schritt in keiner der angefragten Mengen lagen. Frage $A \cup D_{|A|}$. Falls diese Anfrage eine Null liefert, wird A eine weitere Gewinnerin.

3. Sei R eine Gewinnerin. Falls R einelementig ist, so ist die in R liegende Variable unwesentlich. Andernfalls wird R in zwei (fast) gleichgroße Mengen U und V aufgeteilt. Frage $U \cup D_{|U|}$ und $V \cup D_{|V|}$. Jede Null identifiziert eine neue Gewinnerin, die weiter untersucht wird.

Korrektheit:
Im ersten Schritt werden mehr als k' disjunkte Anfragen gestellt, daher muß mindestens eine der Anfragen nur aus wesentlichen Variablen bestehen. Hiermit können die Dummymengen D_j definiert werden. Eine Dummymenge D_j enthält genau $r-j$ wesentliche Variablen. Für eine beliebige Menge S mit $|S| \leq r$ liefert $S \cup D_{|S|}$ genau dann eine Null, wenn S mindestens eine unwesentliche Variable enthält.
Jede Anfrage, die eine Gewinnerin wird, muß mindestens eine unwesentliche Variable enthalten. Im zweiten Schritt werden ggf. die Variablen untersucht, die im ersten Schritt noch nicht betrachtet wurden. Nach diesem Schritt muß jede Variable in genau einer der angefragten Mengen gelegen haben. Dies gilt insbesondere für die k' unwesentlichen Variablen. Diese liegen in den Gewinnerinnen.
Im dritten Schritt werden mit den binären Suchen alle unwesentlichen Variablen identifiziert.

Laufzeit:
Im ersten Schritt werden $\lfloor \frac{n}{r} \rfloor$ Anfragen gestellt, im zweiten Schritt ggf. eine.
Wie viele Mengen werden bei den binären Suchen im dritten Schritt angefragt? Dazu ordne ich jeder Menge eine Tiefe zu, die der Anzahl der vorhergehenden Halbierungen entspricht. Die Gewinnerinnen, die im ersten und zweiten Schritt entstehen, haben Tiefe Null. Die Mengen U und V, die durch Halbierung einer Gewinnerin R mit Tiefe d entstehen, haben Tiefe $d + 1$.

Jede Gewinnerin, die im ersten Schritt gefunden wird, enthält genau r Elemente. Also ist die Tiefe durch $\lceil \log r \rceil$ beschränkt.

Jede Gewinnerin enthält mindestens eine unwesentliche Variable. Da es genau k' unwesentliche Variablen gibt und alle Mengen der gleichen Tiefe d disjunkt sind, kann es höchstens k' Gewinnerinnen der Tiefe d geben. Für jede Gewinnerin werden höchstens 2 Anfragen gestellt. Also werden im zweiten Schritt insgesamt höchstens $O(k' \log r)$ Anfragen gestellt.

Mit $k' < \lfloor \frac{n}{r} \rfloor$ folgt die Behauptung. □

Wenn r so groß ist, daß $\frac{n}{r} = O(\log \binom{n}{k})$ gilt, und die Anzahl der unwesentlichen Variablen klein genug ist, ist dies ein asymptotisch optimaler Algorithmus.

2.2 Unbekannte Parameter v oder k

Bisher habe ich vorausgesetzt, daß die Anzahl der wesentlichen Variablen k sowie der Wertevektor v der symmetrischen Funktion, die untersucht wird, bekannt sind.

Was ändert sich, wenn die Anzahl der unwesentlichen Variablen unbekannt ist? Und was passiert, wenn ich nur weiß, daß die gesuchte Funktion symmetrisch ist, aber den Wertevektor nicht kenne?[2]

Diese Fragen werde ich in diesem Abschnitt untersuchen. Wenn der Wertevektor v unbekannt ist, handelt es sich dabei um eine Erweiterung des Attribut-Lernens, da außer der Menge der wesentlichen Variablen auch der Wertevektor der ausgewählten Funktion gefunden werden muß, um die Funktion zu lernen.

Ich werde für jeden der drei offenen Fälle die entsprechende Funktionenklasse definieren, eine untere Schranke bestimmen und einen linearen Algorithmus angeben. Dafür werden keine neuen Ideen verwendet, sondern die Sätze und Beweise aus dem letzten Abschnitt geeignet verallgemeinert.

[2]Zumindest der erste Fall ist auch in der Praxis interessant, wenn z. B. die wesentlichen Variablen für defekte Produkte stehen, die gefunden werden sollen. Dann ist es plausibel anzunehmen, daß die Anzahl dieser defekten Teile nicht bekannt ist. Für den Fall, daß der Wertevektor unbekannt ist, ist mir leider keine praktische Anwendung eingefallen.

Bekanntes v und unbekanntes k

Definition 2.9: SYM_v^n

> *Sei $n \in \mathbb{N}$ und $v \in \{0,1\}^{n+1}$ ein Wertevektor. Sei f_v die durch den Wertevektor v definierte symmetrische Funktion.*
>
> *Dann besteht die Klasse SYM_v^n aus allen Subfunktionen von f_v mit beliebig vielen wesentlichen von n Variablen. Unwesentliche Variablen werden durch Null ersetzt.*

$$\mathbf{SYM_v^n} := \{f_{v,K} \mid K \subseteq [n]\}$$

Statt SYM_v^n werde ich SYM_v schreiben, wenn n aus dem Zusammenhang bekannt ist. Es gilt

$$SYM_v = \bigcup_{k=0}^{n} SYM_v(k) \tag{2.1}$$

Im Folgenden setze ich voraus, daß v nicht der konstante Null- oder Eins-Vektor ist, da in diesen beiden Fällen die Variablen nicht unterschieden werden können und die Lernaufgabe trivial ist.
Mit Satz 1.2 läßt sich aus der Größe einer Funktionenklasse direkt eine untere Schranke berechnen. Um die Größe von SYM_v zu bestimmen, zeige ich zunächst, daß zwei unterschiedlich große Auswahlen von wesentlichen Variablen, die nicht zu klein sind, stets unterschiedliche Funktionen erzeugen. Mit Gleichung 2.1 ergibt sich das folgende Lemma.

Lemma 2.10: Größe von SYM_v

> *Sei $v \in \{0,1\}^{n+1}$. Sei $r \in [n]$ mit $v_0 = \ldots = v_{r-1} \neq v_r$.*
>
> *Dann enthält die Klasse SYM_v genau*
>
> $$1 + \sum_{k=r}^{n} \binom{n}{k}$$
>
> *Funktionen.*

Beweis: Sei o. B. d. A. $v_0 = \ldots = v_{r-1} = 0, v_r = 1$. Sei f die durch den Wertevektor v definierte Funktion. Seien $A = \{i_1, \ldots, i_k\}$ und $B = \{j_1, \ldots, j_l\}$ zwei beliebige unterschiedlich große Auswahlen von wesentlichen Variablen. Sollte eine der Variablenauswahlen weniger als r Variablen enthalten, so ist die zugehörende Funktion die konstante Funktion $f \equiv v_0$. Dann stimmen die beiden Funktionen genau dann überein, wenn auch die andere Variablenauswahl weniger als r Variablen enthält.

Sei im Folgenden $k, l \geq r$ und o. B. d. A. $k > l$. Zu zeigen ist, daß die beiden durch die Mengen A und B bestimmten Funktionen f_A und f_B unterschiedlich sind. Sei i_s der kleinste Index aus A, der nicht in B vorkommt. Sei o. B. d. A. $i_1 = j_1, \ldots, i_{s-1} = j_{s-1}$.

- 1. Fall: $r \leq s$. Setze $x_{i_1}, \ldots, x_{i_{r-1}} = 1$, $x_{i_s} = 1$ und alle anderen Variablen auf Null. Dann sind in x genau r Variablen aus A auf Eins gesetzt, also ist $f_A(x) = 1$. Andererseits sind genau $r-1$ Variablen aus B auf Eins gesetzt, also gilt $f_B(x) = 0$.
- 2. Fall: $r > s$. Setze $x_{i_1}, \ldots, x_{i_s}$ sowie weitere $r-s$ Variablen $x_{i_j} \in A$ auf Eins, alle anderen Variablen auf Null. Dann sind in x genau r Variablen aus A auf Eins gesetzt, also ist $f_A(x) = 1$. Andererseits sind höchstens $r-1$ Variablen aus B auf Eins gesetzt, da $i_s \notin B$, also ist $f_B(x) = 0$.

Also sind für $k \neq l$ und $k, l \geq r$ alle Funktionen in $SYM_v(k)$ und $SYM_v(l)$ verschieden. Es gilt

$$SYM_v = \bigcup_{k=0}^{n} SYM_v(k) \tag{2.2}$$

Die Klassen $SYM_v(0), SYM_v(1), \ldots, SYM_v(r-1)$ enthalten genau die konstante Funktion $\{f \equiv v_0\}$. Daher gibt es in SYM_v genau

$$1 + \sum_{k=r}^{n} |SYM_v(k)|$$

verschiedene Funktionen. Mit Lemma 2.3 folgt die Behauptung. □

Satz 2.11: Untere Schranke für SYM_v

Sei $v \in \{0,1\}^{n+1}$. Sei $r \in [n]$ mit $v_0 = \ldots = v_{r-1} \neq v_r$.

Dann sind mindestens

$$\left\lceil \log \sum_{k=r}^{n} \binom{n}{k} \right\rceil \tag{2.3}$$

Anfragen notwendig, um die Klasse SYM_v zu lernen.

Beweis: Die Behauptung ergibt sich direkt aus der Größe der Funktionenklasse in Verbindung mit der allgemeinen unteren Schranke aus Satz 1.2. □

Leider ist mir keine geschlossene Form für den Ausdruck 2.3 bekannt.[3]

[3] Und ich bin scheinbar nicht der einzige, vgl. [12, S. 165].

Satz 2.12: Algorithmus für SYM_v

Sei $v \in \{0,1\}^{n+1}$.

Falls $v_0 \neq v_1$ *gilt, so kann die gesuchte Funktion mit genau* n *Anfragen gelernt werden.*

Falls ein $r \in \{2,\ldots,n\}$ *existiert mit* $v_0 = \ldots = v_{r-1} \neq v_r$, *so kann die gesuchte Funktion mit höchstens* $2n-r$ *Anfragen gelernt werden.*

Beweis:

- 1. Fall: $v_0 \neq v_1$. Sei o. B. d. A. $v_0 = 0, v_1 = 1$. Es werden die einelementigen Mengen $\{x_i\}$ mit $i \in [n]$ gefragt. Jede Eins als Antwort identifiziert eine wesentliche Variable. Es werden genau n Anfragen gestellt, um alle Variablen zu identifizieren.
- 2. Fall: $v_0 = \ldots = v_{r-1} \neq v_r$ mit $2 \leq r \leq n$. Dann können die wesentlichen Variablen mit Algorithmus 2.6 gelernt werden, da in diesem Algorithmus nicht verwendet wird, daß k bekannt ist.

$\square$

Falls r nicht zu groß ist (z. B. $r \leq \frac{n}{2}$), so ist die untere Schranke bereits linear in n. Für solche Wertevektoren ist dieser lineare Algorithmus also bereits asymptotisch optimal.

Unbekanntes v und bekanntes k

Das Attribut-Lernen beruht darauf, daß es genau eine Grundfunktion g gibt, aus der eine Klasse von Funktionen gebildet wird, indem einige Variablen durch Konstanten ersetzt werden. Die folgenden beiden Funktionenklassen beruhen aber auf *allen* symmetrischen Funktionen. Daher besteht die Lernaufgabe hier nicht nur in der Identifikation der wesentlichen Variablen, sondern auch in der Bestimmung des Wertevektors v der ausgewählten Funktion. Ich möchte die beiden Klassen trotzdem hier betrachten, da sie natürliche Verallgemeinerungen von $SYM_v(k)$ sind, und beginne mit dem Fall, daß die Anzahl der wesentlichen Variablen bekannt ist.

Definition 2.13: $SYM^n(k)$

Sei $n \in \mathbb{N}$ *und* $0 \leq k \leq n$.

Dann besteht die Klasse $SYM^n(k)$ *aus allen Subfunktionen von beliebigen symmetrischen Funktionen* f_v *mit einem Wertevektor* $v \in \{0,1\}^{n+1}$ *auf* k *wesentlichen von* n *Variablen. Unwesentliche Variablen werden durch Null ersetzt.*

$$\mathbf{SYM^n(k)} := \{f_{v,K} \mid K \subseteq [n], |K| = k, v \in \{0,1\}^{n+1}\}$$

Wie üblich schreibe ich $SYM(k)$ statt $SYM^n(k)$. Es gilt

$$SYM(k) = \bigcup_{v \in \{0,1\}^{n+1}} SYM_v(k) \tag{2.4}$$

Da die Fälle $k = 0$ und $k = n$ trivial sind, betrachte ich nur $1 \le k \le n-1$. In v sind nur die ersten $k+1$ Koordinaten relevant, da es nur k wesentliche Variablen gibt. Daher wird im Folgenden stets $v \in \{0,1\}^{k+1}$ vorausgesetzt.

Lemma 2.14: Größe von $SYM(k)$

> *Sei* $1 \le k \le n-1$.
>
> *Dann enthält die Klasse* $SYM(k)$ *genau* $2 + (2^{k+1} - 2)\binom{n}{k}$ *Funktionen.*

Beweis: Ich verwende Gleichung 2.4, um zu zeigen, daß die Funktionen in $SYM_v(k)$ und $SYM_w(k)$ für zwei verschiedene Wertevektoren v und w verschieden sind, wenn k genügend groß ist. Verschiedene Vektoren v und w können bei gleicher Variablenwahl höchstens dann unterschiedliche Funktionen in $SYM(k)$ beschreiben, wenn sie sich bereits in einer der ersten $k+1$ Koordinaten unterscheiden. Es gibt genau 2^{k+1} verschiedene Vektoren, die sich paarweise in einer der ersten $k+1$ Stellen unterscheiden. Die beiden konstanten Vektoren $v_0 = \ldots = v_k = 0$ und $v_0 = \ldots = v_k = 1$ bestimmen jeweils genau eine Funktion, da wesentliche und unwesentliche Variablen nicht unterschieden werden können. Es bleibt zu zeigen, daß für zwei nichtkonstante Wertevektoren v und w der Länge $k+1$ und für zwei Auswahlen von Variablen $A = \{i_1, \ldots, i_k\}$ und $B = \{j_1, \ldots, j_k\}$ mit $v \ne w$ oder $A \ne B$ auch die Funktionen $f_{v,A}$ und $g_{w,B}$ verschieden sind. Dabei sind f_v und g_w die durch v bzw. w definierten Funktionen.

- 1. Fall: $A = B$, $v \ne w$. Sei l die erste Koordinate, so daß $v_l \ne w_l$ gilt. O.B.d.A. sei $v_l = 1$, $w_l = 0$. Setze $x_{i_1} = \ldots = x_{i_l} = 1$, alle anderen Variablen werden auf Null gesetzt. Dann ist $f_{v,A}(x) = 1$ und $g_{w,B}(x) = 0$.

- 2. Fall: $A \ne B$, $v = w$. Verfahre wie im Beweis von Lemma 2.3.

- 3. Fall: $A \ne B$, $v \ne w$. Sei l der kleinste Index, so daß $v_l \ne w_l$ gilt. O.B.d.A. sei $v_l = 1$, $w_l = 0$. Sei s die Anzahl der Variablen, die in $A \cap B$ liegen. O.B.d.A. sei $A \cap B = \{x_{i_1}, \ldots x_{i_s}\}$. Falls $l \le s$, so können analog zum ersten Fall $x_{i_1}, \ldots, x_{i_l}$ auf Eins und alle anderen Variablen auf Null gesetzt werden. Dann ist $f_{v,A}(x) = 1$ und $g_{w,B}(x) = 0$. Andernfalls gilt $l > s$. Die Variablen $x_{i_1}, \ldots, x_{i_s}, x_{i_{s+1}}, \ldots, x_{i_l}$ sowie $x_{j_{s+1}}, \ldots, x_{j_l}$ werden auf Eins und alle verbleibenden Variablen auf Null gesetzt. Dann sind l Variablen aus A und l Variablen aus B auf Eins gesetzt. Es gilt $f_{v,A}(x) = 1$ und $g_{w,B}(x) = 0$.

Da es 2^{k+1} verschiedene Wertevektoren der Länge $k+1$ gibt und für jeden Wertevektor außer den beiden konstanten Vektoren genau $\binom{n}{k}$ Möglichkeiten für die wesentlichen Variablen existieren, ist die Anzahl verschiedener Funktionen in $SYM(k)$ genau $2+(2^{k+1}-2)\binom{n}{k}$. □

Satz 2.15: Untere Schranke für $SYM(k)$

> *Es sind mindestens* $\left\lceil \log(2+(2^{k+1}-2)\binom{n}{k})\right\rceil$ *Anfragen nötig, um* $SYM(k)$ *zu lernen. Es gilt*
>
> $$\left\lceil \log(2+(2^{k+1}-2)\binom{n}{k})\right\rceil \geq k + \left\lceil \log\binom{n}{k}\right\rceil$$

Beweis: Die untere Schranke ergibt sich aus der Größe der Konzeptklasse in Verbindung mit Satz 1.2. Die Abschätzung ergibt sich aus folgenden einfachen Rechnungen.
Für $k=1$ gilt $\left\lceil \log(2+(2^{k+1}-2)\binom{n}{k})\right\rceil = \lceil \log(2+2\binom{n}{1})\rceil \geq 1+\lceil \log n\rceil$.
Für $k \geq 2$ gilt

$$\begin{aligned}
\left\lceil \log\left(2+(2^{k+1}-2)\binom{n}{k}\right)\right\rceil &= \left\lceil \log 2 + \log\left(1+(2^k-1)\binom{n}{k}\right)\right\rceil \\
&\geq 1+\left\lceil \log(2^k-1)\binom{n}{k}\right\rceil \\
&= 1+\left\lceil \log(2^k-1)+\log\binom{n}{k}\right\rceil \\
&\geq 1+\left\lceil \log(2^k-1)\right\rceil + \left\lceil \log\binom{n}{k}\right\rceil - 1 \\
&= k+\left\lceil \log\binom{n}{k}\right\rceil
\end{aligned}$$

□

Satz 2.16: Algorithmus für $SYM(k)$

> *Mit maximal* $2n$ *Anfragen kann jede Funktion aus der Klasse* $SYM(k)$ *gelernt werden. Ist die gesuchte Funktion konstant, so genügen* $n+1$ *Anfragen, um dies festzustellen.*

Beweis: Um die unbekannte Funktion $f \in SYM(k)$ zu lernen, müssen alle Variablen identifiziert und der Vektor $v_0,\ldots,v_k$ gefunden werden, wobei k die Anzahl der wesentlichen Variablen bezeichnet. Diese Aufgabe erfüllt der folgende Algorithmus:

Algorithmus 2.17:

1. Teste $\{x_1, \ldots, x_j\}$ mit $0 \leq j \leq n$ höchstens solange, bis für $l = j$ etwas anderes geantwortet wird als für $l-1$. Wenn dieser Fall nie eintritt, ist die gesuchte Funktion konstant, die Variablen können nicht unterschieden werden. Falls die Antworten stets Eins waren, so gilt $v_0 = \ldots = v_k = 1$, sonst $v_0 = \ldots = v_k = 0$. STOP.

   Andernfalls werde o. B. d. A. für $\{x_1, \ldots, x_{l-1}\}$ die Antwort 0 und für $\{x_1, \ldots, x_l\}$ die Antwort 1 gegeben. Dann ist x_l wesentliche Variable.

2. Teste $\{x_1, \ldots, x_l\} \backslash \{x_i\}$ für $1 \leq i \leq l-1$. Jede Antwort Null identifiziert eine wesentliche, jede Eins eine unwesentliche Variable.

   Sei r die Anzahl der bisher gefunden wesentlichen Variablen. Dann gilt $v_0 = \ldots = v_{r-1} = 0, v_r = 1$.

   * Falls $r = k$ gilt, so sind alle wesentlichen Variablen gefunden und $v_0, \ldots, v_k$ ist bekannt. STOP.

3. Teste $\{x_1, \ldots, x_{l-1}\} \cup \{x_j\}$ für $l+1 \leq j \leq n$. Jede Eins identifiziert eine wesentliche, jede Null eine unwesentliche Variable.

   Nach diesem Schritt sind alle k wesentlichen Variablen gefunden.

4. Teste Mengen mit j wesentlichen Variablen für $r+1 \leq j \leq k$ und bestimme so die Werte v_j.

Für eine konstante Funktion werden genau $n+1$ Anfragen gestellt. Andernfalls sind $(l+1)+(l-1)+(n-l)+(k-r) = n+l+(k-r)$ Anfragen notwendig. Es gilt $k - r \leq n - l$, da nach dem zweiten Schritt in den noch nicht untersuchten $n-l$ Variablen genau $k-r$ wesentliche Variablen enthalten sind. Daher kann die gesuchte Funktion mit höchstens $2n$ Anfragen gelernt werden. □

Der Algorithmus ist für $k = \Omega(n)$ bereits asymptotisch optimal, da die untere Schranke aus Satz 2.15 dann linear in n ist.

Unbekanntes v und unbekanntes k

In der letzten Klasse von allgemeinen symmetrischen Funktionen, die ich in diesem Abschnitt betrachte, sind sowohl der Wertevektor als auch die Anzahl der wesentlichen Variablen unbekannt. In diesem Fall ist die Funktionenklasse so groß, daß ein linearer Algorithmus bereits asymptotisch optimal ist.

Definition 2.18: *SYM^n*

Sei $n \in \mathbb{N}$.

Dann besteht die Klasse SYM^n aus allen Subfunktionen von beliebigen symmetrischen Funktionen f_v mit einem Wertevektor $v \in \{0,1\}^{n+1}$ mit beliebig vielen wesentlichen von n Variablen. Unwesentliche Variablen werden durch Null ersetzt.

$$\mathbf{SYM^n} := \{f_{v,K} |\ K \subseteq [n], v \in \{0,1\}^{n+1}\}$$

Statt SYM^n werde ich SYM verwenden. Es gilt

$$SYM = \bigcup_{k=0}^{n} SYM(k) \tag{2.5}$$

Hiermit läßt sich wie in den vorherigen Abschnitten die Größe der Funktionenklasse bestimmen.

Lemma 2.19: Größe von *SYM*

Die Klasse SYM enthält genau $2 + 2 \cdot 3^n - 2 \cdot 2^n$ Funktionen.

Beweis: Der Beweis ergibt sich aus Gleichung 2.5 und der Größe der Klassen $SYM(k)$. Ich zeige zunächst, daß für verschiedene k und l in $SYM(k)$ und $SYM(l)$ nur die beiden konstanten Funktionen gemeinsam enthalten sind. Seien dazu $f \in SYM(k)$, $g \in SYM(l)$, f und g nicht konstant und o. B. d. A. $k < l$. Die Wertevektoren seien v und w, die gewählten wesentlichen Variablen seien $A = \{i_1, \ldots, i_k\}$ und $B = \{j_1, \ldots, j_l\}$. Sei o. B. d. A. $j_l \notin A$. Sei s so gewählt, daß $v_0 = \ldots = v_{s-1} \neq v_s$ und o. B. d. A. sei $v_{s-1} = 0, v_s = 1$.

- 1. Fall: Die beiden Wertevektoren unterscheiden sich bereits in einer der ersten k Koordinaten, d. h. es existiert ein r mit $0 < r \leq k$, so daß $v_0 = w_0, \ldots, v_{r-1} = w_{r-1}, v_r \neq w_r$ gilt.

  1. Unterfall: $s < r$: Es werden $s-1$ Variablen aus B ausgewählt und auf Eins gesetzt, ebenso die Variable x_{j_l}. Alle anderen Variablen in x erhalten den Wert Null. Dann sind genau s Variablen aus B auf Eins gesetzt und daher gilt $g_{w,B}(x) = 1$. Andererseits sind höchstens $s-1$ Variablen aus A mit Eins belegt, also gilt $f_{v,A}(x) = 0$.

  2. Unterfall: $s > r$: Dann gilt $v_0 = \ldots = v_{r-1} = v_r = 0$ und $w_0 = \ldots = w_{r-1} = 0, w_r = 1$. Es werden $r-1$ Variablen aus B ausgewählt und auf Eins gesetzt, außerdem die Variable x_{j_l}. Alle anderen Variablen in x erhalten den Wert Null. Dann sind genau r Variablen aus B auf Eins gesetzt und es gilt $g_{w,B}(x) = 1$. Andererseits sind höchstens $r-1$ Variablen aus A mit Eins belegt, daher folgt $f_{v,A}(x) = 0$.

3. Unterfall: $s = r$: Aus A werden genau s Variablen mit Eins belegt, alle anderen Variablen werden auf Null gesetzt. Dann gilt $f_{v,A}(x) = 1$ und $g_{w,B}(x) = 0$.

- 2. Fall: $v_0 \neq w_0$: Für $x = (0, \ldots, 0)$ gilt $f_{v,A}(x) = 1$ und $g_{w,B}(x) = 0$.
- 3. Fall: Es gilt $v_j = w_j$ für jedes $0 \leq j \leq k$. Dann gilt insbesondere $w_{s-1} = 0$ und $w_s = 1$. Es werden $s - 1$ Variablen aus B auf Eins gesetzt, außerdem die Variable x_{j_l}. Alle anderen Variablen erhalten den Wert Null. Dann gilt $f_{v,A}(x) = 0$ und $g_{w,B}(x) = 1$.

Also sind je zwei Funktionen aus $SYM(k)$ und $SYM(l)$ verschieden, sofern sie nicht konstant sind. Mit der Größe der Klasse $SYM(k)$ aus Lemma 2.14 folgt

$$\begin{aligned}
|SYM| &= |\bigcup_{k=0}^{n} SYM(k)| \\
&= 2 + \sum_{k=0}^{n} (2^{k+1} - 2)\binom{n}{k} \\
&= 2 + 2\sum_{k=0}^{n} 2^k \binom{n}{k} - \sum_{k=0}^{n} 2\binom{n}{k} \\
&= 2 + 2 \cdot 3^n - 2 \cdot 2^n
\end{aligned}$$

□

Satz 2.20: Untere Schranke für SYM

Es sind mindestens $\lceil \log(2 + 2 \cdot 3^n - 2 \cdot 2^n) \rceil$ Anfragen notwendig, um die Klasse SYM zu lernen. Dies sind mindestens $\lceil 1,58n \rceil$ Anfragen.

Beweis: Die untere Schranke ergibt sich aus der Größe der Klasse SYM in Verbindung der allgemeinen unteren Schranke aus Satz 1.2.
Für $n = 1$ gilt $\lceil \log(2 + 2 \cdot 3^n - 2 \cdot 2^n) \rceil = 2 \geq 1,58$.
Für $n \geq 2$ gilt $3^n \geq 2^{n+1}$. Hieraus folgt

$$\begin{aligned}
\lceil \log(2 + 2 \cdot 3^n - 2 \cdot 2^n) \rceil &\geq \left\lceil \log(3^n + 3^n - 2^{n+1}) \right\rceil \\
&\geq \lceil \log(2 \cdot 3^n - 3^n) \rceil \\
&\geq \lceil \log 3^n \rceil \\
&\geq 1,58n
\end{aligned}$$

□

Satz 2.21: Algorithmus für *SYM*

> *Mit maximal* $2n$ *Anfragen kann jede Funktion aus* SYM *gelernt werden. Ist die gesuchte Funktion konstant, so genügen* $n+1$ *Anfragen, um dies festzustellen.*

Beweis: Wenn in Algorithmus 2.17 der mit * gekennzeichnete Schritt nicht ausgeführt wird, lernt dieser Algorithmus alle wesentlichen Variablen, ohne deren Anzahl zu kennen. Im dritten Schritt wird dann k als die Anzahl der bis dahin gefundenen wesentlichen Variablen festgelegt. Die Analyse und die Laufzeit ergeben sich direkt aus dem Beweis von Satz 2.16. □

Da die untere Schranke für die Klasse SYM linear in n ist, ist dieser Algorithmus asymptotisch optimal.

Kapitel 3

Threshold-Funktionen

Eine Threshold-Funktion mit Thresholdwert τ gibt genau dann eine Eins aus, wenn in der Eingabe mindestens τ Einsen enthalten sind. Der Wertevektor einer solchen Funktion hat die Form $v = (0, \ldots, 0, 1, \ldots 1)$, wobei genau an der τ-ten Position der Wechsel zwischen Null und Eins geschieht. Ich untersuche die Klassen $THR^n_\tau(k)$ von Threshold-Funktionen mit Threshold-Wert τ auf n Variablen mit k wesentlichen Variablen.
In der Arbeit von Uehara, Tsuchida und Wegener [24] werden für $k \leq \frac{n}{2}$ und beliebiges $0 \leq \tau \leq k$ asymptotisch optimale Algorithmen angegeben. Für $k > \frac{n}{2}$ werden dort für den Spezialfall $\tau = \frac{n}{2}$ ebenfalls asymptotisch optimale Algorithmen angegeben. Hegedüs und Idyk untersuchen in [16], wie Disjunktionen von Threshold-Funktionen gelernt werden können. Sie geben für Disjunktionen von wenigen Funktionen Algorithmen an, die mit polynomiell vielen Anfragen (membership und equivalence) auskommen.

Ich werde in diesem Abschnitt die Algorithmen aus [24] für $k \geq \frac{n}{2}$ und beliebiges $0 \leq \tau \leq k$ verallgemeinern und eine weitere untere Schranke angeben. Anschließend wird gezeigt, daß mit diesen Algorithmen jede Funktion aus $THR^n_\tau(k)$ mit asymptotisch optimal vielen Anfragen gelernt werden kann.

Definition 3.1: Threshold-Funktion

> *Seien $n \in \mathbb{N}$ und $\tau \in \mathbb{N}_0$. Eine Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt* **Threshold-Funktion mit Thresholdwert** τ*, wenn $f(x) = 1$ genau dann gilt, wenn $|x|_1 \geq \tau$, d. h. die Anzahl Einsen in x mindestens τ ist.*
>
> *Man schreibt $f(x) = thr_\tau(x)$.*

Der Wertevektor der Funktion thr_τ beginnt mit τ Nullen, gefolgt von $n-\tau+1$ Einsen.

[24], S.4

Definition 3.2: $THR^n_\tau(k)$

Seien $n \in \mathbb{N}$ und $k, \tau \in \mathbb{N}_0$ mit $0 \leq k \leq n$.

Dann besteht die Klasse $THR^n_\tau(k)$ aus allen Threshold-Funktionen mit Threshold-Wert τ auf n Variablen mit k wesentlichen Variablen. Unwesentliche Variablen werden durch Null ersetzt.

$$THR^{\mathbf{n}}_\tau(\mathbf{k}) := \{thr_{\tau,K} \mid K \subseteq [n], |K| = k\}$$

Ich schreibe auch $THR_\tau(k)$, wenn n aus dem Zusammenhang eindeutig hervorgeht.
Für $\tau = 0$ und $\tau > k$ besteht $THR^n_\tau(k)$ nur aus der konstanten Eins- bzw. Null-Funktion. Für $k = 0$ oder $k = n$ ist die Lernaufgabe trivial.
Es gilt $THR^n_1(k) = OR^n(k)$ und $THR^n_k(k) = AND^n(k)$. Hierfür werden in [24] asymptotisch optimale Algorithmen beschrieben. Außerdem wird dort für $THR_\tau(k)$ mit $k \leq \frac{n}{2}$ ein asymptotisch optimaler Algorithmus angegeben.

In [24] wird gezeigt, daß mindestens $\lceil \log \binom{n}{k} \rceil$ Anfragen notwendig sind, um $THR_\tau(k)$ zu lernen. Ich gebe eine weitere untere Schranke, indem ich eine Strategie eines Gegenspielers (*adversary*) beschreibe, der in jeder Anfrage höchstens τ Variablen identifiziert.

Satz 3.3: Untere Schranke für $THR_\tau(k)$

Sei $n, k \in \mathbb{N}$ mit $1 \leq k \leq n$. Sei $\tau \in \mathbb{N}$ mit $1 \leq \tau \leq k - 1$.

Dann sind mindestens $\left\lceil \frac{k}{\tau} \right\rceil$ Anfragen notwendig, um die Klasse $THR_\tau(k)$ zu lernen.

Beweis: Ich nehme an, daß Anfragen mindestens τ Variablen enthalten, da sonst die Antwort Null bereits vorher feststeht.
Ein Gegenspieler kann jeden beliebigen Algorithmus zwingen, mindestens $\left\lceil \frac{k}{\tau} \right\rceil$ Anfragen zu stellen.

Dazu antwortet der Gegenspieler auf die ersten $\left\lceil \frac{k}{\tau} \right\rceil - 1$ Anfragen eines Algorithmus stets eine Eins und identifiziert zusätzlich in jeder Anfrage genau τ Variablen als wesentlich. Dabei werden zunächst solche Variablen benannt, die bereits in früheren Schritten als wesentlich bekannt gegeben wurden. Der Algorithmus erhält keine Informationen über die weiteren Variablen. Nach diesen Anfragen sind höchstens $k-1$ Variablen identifiziert und der Algorithmus muß noch mindestens eine weitere Anfrage stellen, um alle Variablen zu lernen. □

Also sind für $THR_\tau(k)$ die unteren Schranken $\lceil \log \binom{n}{k} \rceil$ und $\left\lceil \frac{k}{\tau} \right\rceil$ bekannt. Für kleines τ und großes k ist die erste Schranke der zweiten unterlegen.
Ich gebe nun mehrere Algorithmen an, um Threshold-Funktionen zu lernen. Diese sind Verallgemeinerungen aus [24], die dort für $\tau = \frac{n}{2}$ angegeben

werden. Dabei bezeichne ich mit k die Anzahl der wesentlichen und mit $k' = n - k$ die Anzahl der unwesentlichen Variablen. In [24] wird gezeigt, daß $THR_\tau^n(n-k')$ genauso schwierig zu lernen ist wie $THR_{n-k'-\tau+1}^n(n-k')$. Deswegen werde ich mich im Folgenden auf

$$\boxed{n, k, \tau \in \mathbb{N}, 1 \leq k' \leq \tfrac{n}{2} \text{ und } 2 \leq \tau \leq \tfrac{n-k'}{2}}$$

beschränken, da alle anderen Fälle entweder trivial oder bereits untersucht sind. Diese Voraussetzungen gelten generell und werden in den Sätzen teilweise verschärft.

Satz 3.4: Algorithmus für $THR_\tau(n-k')$ mit $k' < \lfloor \frac{n}{\tau} \rfloor$

Sei $k' < \lfloor \frac{n}{\tau} \rfloor$. Dann kann $THR_\tau(n-k')$ mit $O(\frac{n-k'}{\tau} + k' \log \frac{n}{k'})$ Anfragen gelernt werden.

Beweis: Der Wertevektor v einer Threshold-Funktion hat die Gestalt $v = (0, \ldots, 0, 1, \ldots, 1)$. Mit dem Algorithmus 2.8 aus Satz 2.7 folgt direkt die Behauptung. $\square$

In [24] wird ein Algorithmus für $k' = \Omega(\sqrt{\frac{n}{\log n}})$ und $\tau = \frac{n}{2}$ angegeben, der die Klasse $THR_\tau(n-k')$ mit $O(k' \log \frac{n}{k'})$ Anfragen lernt. Ich werde diesen Ansatz für $\left\lfloor \frac{n}{k'+1} \right\rfloor \leq \tau \leq \frac{n}{2}$ verallgemeinern.
Dazu werden Dummymengen D_j für $1 \leq j \leq \left\lfloor \frac{n}{k'+1} \right\rfloor$ verwendet, so daß D_j genau $\tau - j$ wesentliche Variablen enthält. Für eine Menge A mit höchstens $\left\lfloor \frac{n}{k'+1} \right\rfloor$ Elementen liefert die Anfrage $A \cup D_{|A|}$ genau dann eine Eins, wenn A nur wesentliche Variablen enthält. Umgekehrt liefert $A \cup D_{|A|}$ eine Null, wenn A mindestens eine unwesentliche Variable enthält. Mit einer binären Suche können alle unwesentlichen Variablen gefunden werden.
Um die Dummymengen zu bestimmen, werden aus einer Menge S^*, die genau τ wesentliche Variablen enthält, solange Variablen entfernt, bis $\left\lfloor \frac{n}{k'+1} \right\rfloor$ wesentliche Variablen bekannt sind. Dann können alle Dummymengen definiert werden.

Satz 3.5: Algorithmus für $THR_\tau(n-k')$ mit $k' = \Omega(\sqrt{\frac{n}{\log n}})$, $\tau \geq \left\lfloor \frac{n}{k'+1} \right\rfloor$

Sei $k' \leq \frac{n}{8}, \tau \geq \left\lfloor \frac{n}{k'+1} \right\rfloor$ und $k' = \Omega(\sqrt{\frac{n}{\log n}})$.

Dann kann die Klasse $THR_\tau(n-k')$ mit $O(k' \log \frac{n}{k'})$ Anfragen gelernt werden.

Beweis: Ich teile die Variablen in drei disjunkte Mengen B_1, B_2 und B_3 auf mit

$$B_i = \{x_{\lceil \frac{(i-1)n}{3} \rceil + 1}, \ldots, x_{\frac{in}{3}}\}$$

und gebe einen Algorithmus an, der alle unwesentlichen Variablen in B_3 findet. Analog können alle anderen Variablen gelernt werden. In Abbildung 3.1 werden die verwendeten Mengen veranschaulicht.

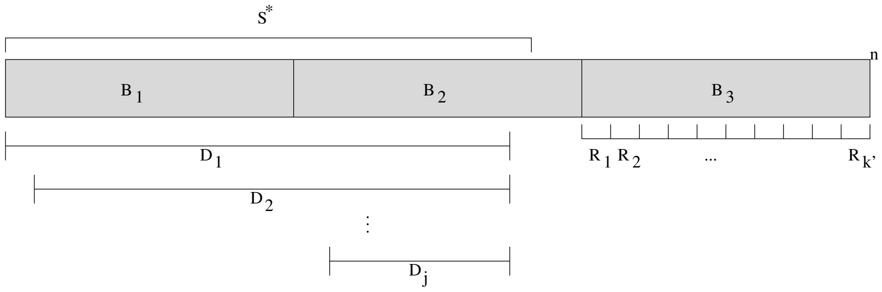

Abbildung 3.1: In Algorithmus 3.6 verwendete Mengen

Algorithmus 3.6:

1. Bestimme ein minimales s, so daß $S^* = \{x_1, \ldots, x_s\}$ eine Eins liefert. Dieses liegt in $\{\tau, \ldots, \tau + k'\}$ und kann mit binärer Suche bestimmt werden.

2. Stelle für $1 \leq i \leq s$ die Anfrage $S^* \backslash \{x_i\}$ solange, bis $\left\lfloor \frac{n}{k'+1} \right\rfloor$ wesentliche Variablen $e_1, \ldots, e_{\left\lfloor \frac{n}{k'+1} \right\rfloor}$ gefunden wurden. Die Variable x_i ist wesentlich, wenn die Anfrage $S^* \backslash \{x_i\}$ eine Null liefert.

3. Für $1 \leq j \leq \left\lfloor \frac{n}{k'+1} \right\rfloor$ wird $D_j := S^* \backslash \{e_1, \ldots, e_j\}$ definiert. Eine Menge D_j enthält genau $\tau - j$ wesentliche Variablen.

4. Teile B_3 in k' disjunkte Mengen $R_1, \ldots, R_{k'}$ mit jeweils höchstens $\left\lfloor \frac{n}{k'+1} \right\rfloor$ Elementen auf.

5. Frage für $1 \leq i \leq k'$ die Mengen $R_i \cup D_{|R_i|}$ an. Jede Menge mit einer Null als Antwort wird eine Gewinnerin. Eine Gewinnerin enthält mindestens eine unwesentliche Variable.

6. Sei R eine Gewinnerin. Falls R einelementig ist, so ist die in R liegende Variable unwesentlich. Andernfalls wird R in zwei (fast) gleichgroße Mengen U und V aufgeteilt. Frage $U \cup D_{|U|}$ und $V \cup D_{|V|}$. Jede Null identifiziert eine neue Gewinnerin, die weiter untersucht wird.

Korrektheit:
Das im ersten Schritt gesuchte s liegt in $\{\tau, \ldots, \tau + k'\}$, da einerseits mindestens τ wesentliche Variablen notwendig sind, damit S^* eine Eins erzeugen kann, und andererseits S^* höchstens k' unwesentliche Variablen enthalten kann.

Im zweiten Schritt werden mindestens $\left\lfloor \frac{n}{k'+1} \right\rfloor$ wesentliche Variablen gefunden, da in S^* genau $\tau \geq \left\lfloor \frac{n}{k'+1} \right\rfloor$ wesentliche Variablen enthalten sind. Daher können im dritten Schritt alle D_j gebildet werden. Es gilt $D_j \subseteq S^* \subseteq \{x_1, \ldots, x_{\tau+k'}\}$. Aus $k' \leq \frac{n}{8}$ und $\tau \leq \frac{n}{2}$ folgt $\tau + k' \leq \left\lceil \frac{2n}{3} \right\rceil$, also gilt $S^* \cap B_3 = \emptyset$.

Ich zeige nun, daß mit den k' Mengen $R_1, \ldots, R_{k'}$, die im vierten Schritt gebildet werden, die Menge B_3 überdeckt werden kann. Dazu weise ich nach, daß die Vereinigung dieser Mengen mindestens $\lceil \frac{n}{3} \rceil$ Variablen enthält, wenn die Mengen alle Größe $\left\lceil \frac{n}{k'+1} \right\rceil$ haben. Im Algorithmus werden einige dieser Mengen gegebenenfalls kleiner gewählt, damit die Überdeckung disjunkt wird. Für $k' \geq 2$ und n groß genug gilt

$$\begin{aligned}
k' \left\lfloor \frac{n}{k'+1} \right\rfloor &\geq k' \left(\frac{n}{k'+1} - 1 \right) \\
&= \frac{k'}{k'+1} n - k' \\
&\geq \frac{n}{2} - k' \\
&\geq \frac{n}{2} - \frac{n}{8} \\
&= \frac{3n}{8} \\
&\geq \frac{n}{3} + 1 \\
&\geq \left\lceil \frac{n}{3} \right\rceil
\end{aligned}$$

Die in den letzten beiden Schritten verwendeten Mengen enthalten jeweils höchstens $\left\lfloor \frac{n}{k'+1} \right\rfloor$ Variablen. Eine Menge R wird genau dann zur Gewinnerin, wenn sie mindestens eine unwesentliche Variable enthält, denn sonst enthielte $R \cup D_{|R|}$ genau τ wesentliche Variablen und würde eine Eins liefern. Mit den binären Suchen werden alle in B_3 enthaltenen unwesentlichen Variablen gefunden.

Laufzeit:

Der erste Schritt benötigt höchstens $\lceil \log(k'+1) \rceil$ Anfragen. Im zweiten Schritt werden höchstens $k' + \left\lfloor \frac{n}{k'+1} \right\rfloor$ Anfragen gestellt, da es nur k' unwesentliche Variablen gibt. Die Analyse der binären Suche im sechsten Schritt geschieht wie im Beweis von Satz 3.4. Insgesamt werden höchstens

$$\begin{aligned}
& \lceil \log(k'+1) \rceil + \left(k' + \left\lfloor \frac{n}{k'+1} \right\rfloor\right) + k' + 2k' \cdot \left\lceil \log \left\lfloor \frac{n}{k'+1} \right\rfloor \right\rceil \\
= \quad & O(\log k') + O(k') + O\left(\frac{n}{k'}\right) + O\left(k' \log \frac{n}{k'}\right) \\
= \quad & O\left(k' \log \frac{n}{k'}\right) + O\left(\frac{n}{k'}\right)
\end{aligned}$$

Anfragen gestellt. Aus Lemma B.4 folgt $\frac{n}{k'} = O(k' \log \frac{n}{k'})$ für $k' = \Omega(\sqrt{\frac{n}{\log n}})$ und somit die Behauptung.

□

Im vorigen Satz konnten in S^* mit linearer Suche $\left\lfloor \frac{n}{k'+1} \right\rfloor$ wesentliche Variablen bestimmt werden, da k' genügend groß war. Für kleines k' ergibt sich eine schlechte Laufzeit. Ich gebe nun einen Algorithmus an, der ebenfalls mit Dummymengen arbeitet. Die Bestimmung dieser Mengen ist jedoch aufwendiger. Deswegen werde ich zunächst die Idee beschreiben, wie die Dummymengen gefunden werden können. Das genaue Verfahren findet sich im Beweis des folgenden Satzes.
Um die Dummymengen definieren zu können, müssen genügend viele wesentliche Variablen bekannt sein. Ich verwende ein geeignetes System von disjunkten Mengen R_i, die jeweils genau r Variablen enthalten (für ein genauer zu bestimmendes r). Es ist bekannt, daß ein Index i_0 existiert, so daß R_{i_0} nur wesentliche Variablen enthält. Dieser Index soll mit Hilfe einer Mengenkette $C_0 \supset C_1 \supset \ldots \supset C_{k'}$ bestimmt werden. Die Kette ist so konstruiert, daß die Anzahl wesentlicher Variablen in C_j für steigendes j monoton fallend ist und es mindestens einen Index j_0 gibt, so daß C_{j_0} genau $\tau - r$ wesentliche Variablen enthält. Die Anfrage $R_{i_0} \cup C_{j_0}$ würde also eine Eins liefern, da R_{i_0} genau r wesentliche Variablen enthält. Der Index j_0 wird nun mit einer binären Suche bestimmt, indem geeignete Mengen C_j mit jedem R_i vereinigt und angefragt werden. Für den Index j_0 gilt dann, daß mindestens eine dieser Anfragen eine Eins liefert und für alle Anfragen mit C_{j_0+1} nur Nullen geantwortet werden. Mit dem Index j_0 kann i_0 anschließend leicht gefunden werden.

Satz 3.7: Algorithmus für $THR_\tau(n-k')$ mit $\frac{n}{3k'} \leq \tau$ und $k' = O(n^{1-\epsilon})$

> *Sei $k' \leq \frac{n}{12}$, $\frac{n}{3k'} \leq \tau$ und $k' = O(n^{1-\epsilon})$ für ein $\epsilon > 0$. Dann kann $THR_\tau(n-k')$ mit $O(k' \log \frac{n}{k'})$ Anfragen gelernt werden.*

Beweis: Ich teile die Variablen in vier disjunkte Mengen $B_1, \ldots, B_4$ auf mit

$$B_i = \{x_{\lceil \frac{(i-1)n}{4} \rceil + 1}, \ldots, x_{\frac{in}{4}}\}$$

und gebe einen Algorithmus an, der alle unwesentlichen Variablen in B_4 findet. Analog können alle anderen Variablen gelernt werden. In Abbildung 3.2

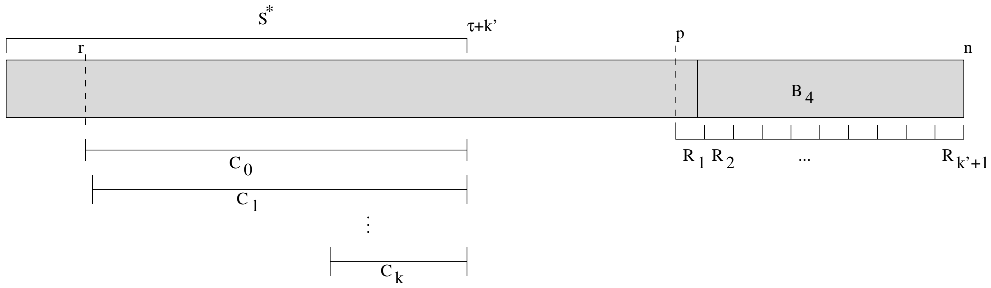

Abbildung 3.2: In Algorithmus 3.8 verwendete Mengen

sind die verwendeten Mengen veranschaulicht. Im Gegensatz zu den anderen Algorithmen in dieser Arbeit gebe ich beim folgenden Algorithmus direkt zu jedem Schritt einige Erklärungen an. Diese stellen einerseits einen Nachweis der Korrektheit dar, sind aber andererseits notwendig, um die weitere Arbeitsweise des Algorithmus zu verstehen.

Algorithmus 3.8:

1. Sei $S^* := \{x_1, \ldots, x_{\tau+k'}\}$. S^* enthält mindestens τ wesentliche Variablen.

2. Sei $r = \left\lceil \frac{n}{4(k'+1)} \right\rceil$ und $p = n - (k'+1)r + 1$. Dann gilt $\tau + k' < p \leq \left\lceil \frac{3n}{4} \right\rceil + 1$ und $r \leq \frac{n}{3k'} \leq \tau$.

Die Aussage $\tau + k' < p$ folgt direkt aus den Voraussetzungen des Satzes, denn mit $k' \leq \frac{n}{12}$ und $\tau \leq \frac{n}{2}$ gilt

$$\begin{aligned} p &= n - (k'+1)\left\lceil \frac{n}{4(k'+1)} \right\rceil + 1 \\ &> n - (k'+1)\left(\frac{n}{4(k'+1)} + 1\right) + 1 \\ &= \frac{3n}{4} - k' \\ &\geq \frac{3n}{4} - \frac{n}{12} \\ &= \frac{8n}{12} \\ &> \frac{n}{2} + \frac{n}{12} \\ &\geq \tau + k' \end{aligned}$$

Die anderen Ungleichungen ergeben sich durch einfaches Nachrechnen.

3. Teile die Menge $\{x_p, \ldots, x_n\}$ in $k'+1$ disjunkte Mengen R_i der Größe r auf. Mindestens ein R_i besteht nur aus wesentlichen Variablen.

Es gilt $(k'+1)r = n-p+1$. In den $k'+1$ disjunkten Mengen können höchstens k' unwesentliche Variablen liegen, also muß mindestens eine Menge nur aus wesentlichen Variablen bestehen. Aus $p \leq \left\lceil \frac{3n}{4} \right\rceil + 1$ folgt

$$B_4 \subseteq \bigcup_{i=1}^{k'+1} R_i$$

4. Für $0 \leq j \leq k'$ werden die Mengen $C_j = S^* \backslash \{x_1, \ldots, x_{r+j}\}$ definiert.

Es gilt $r + j \leq \tau + k'$ und es entsteht eine Mengenkette $C_{k'} \subset C_{k'-1} \subset \ldots \subset C_0 \subseteq S^*$.
C_0 enthält mindestens $\tau - r$ und $C_{k'}$ höchstens $\tau - r$ wesentliche Variablen. Die Anzahl wesentlicher Variablen in C_j ist monoton fallend in j. In einem Schritt kann sich diese Anzahl höchstens um Eins vermindern. Also muß es mindestens ein j_0 geben, so daß C_{j_0} genau $\tau - r$ wesentliche Variablen enthält.
Falls für ein festes j alle Anfragen $C_j \cup R_i$ für $1 \leq i \leq k'+1$ eine Null liefern, so enthält C_j weniger als $\tau - r$ wesentliche Variablen, da mindestens ein R_i genau r wesentliche Variablen enthält.

5. Führe eine binäre Suche auf $\{0, \ldots, k'\}$ durch, um den Index j_0 zu bestimmen. Für einen Suchbereich A wird das in der Mitte von A gelegene j gewählt und es werden die Anfragen $C_j \cup R_i$ für alle $1 \leq i \leq k'+1$ gestellt. Wenn mindestens eine Anfrage eine Eins liefert, so wird in der oberen Hälfte von A weitergesucht, andernfalls in der unteren. Die Suche endet, wenn der Suchbereich einelementig wird. Dieses Element sei j_0.

Wenn für C_j alle Anfragen eine Null liefern, so enthält C_j weniger als $\tau - r$ wesentliche Variablen. Das gesuchte j_0 muß also kleiner als j sein, da die Anzahl wesentlicher Variablen monoton fallend in j ist.

6. Sei $i_0 \in \{1, \ldots, k'+1\}$ ein Index, so daß $C_{j_0} \cup R_{i_0}$ eine Eins liefert. R_{i_0} enthalte die Variablen $\{e_1, \ldots, e_r\}$.

Der Index i_0 kann direkt aus den Antworten des vorigen Schrittes ermittelt werden, da C_{j_0} genau $\tau - r$ wesentliche Variablen enthält und jedes i, für das $C_{j_0} \cup R_i$ eine Eins liefert, geeignet ist. $C_{j_0} \cup R_{i_0}$ hat genau τ wesentliche Variablen.

7. Definiere die Dummymengen $D_l = C_{j_0} \cup R_{i_0} \backslash \{e_1, \ldots, e_l\}$ für $1 \leq l \leq r$.

D_l enthält genau $\tau - l$ wesentliche Variablen und es gilt $D_l \cap R_i = \emptyset$ für alle $i \neq i_0$, da C_j Teilmenge von S^* ist, $\tau + k' < p$ gilt und alle R_i paarweise disjunkt sind.

8. Für $1 \leq i \leq k'+1$ und $i \neq i_0$ wird die Anfrage $R_i \cup D_{|R_i|}$ gestellt. Jede Antwort Null erzeugt einen „Gewinner".

Jeder Gewinner enthält mindestens eine unwesentliche Variable.

9. Sei R ein Gewinner. Falls R einelementig ist, so ist die in R liegende Variable unwesentlich. Andernfalls wird R in zwei (fast) gleichgroße Mengen U und V aufgeteilt. Frage $U \cup D_{|U|}$ und $V \cup D_{|V|}$. Jede Null identifiziert einen neuen Gewinner, der weiter untersucht wird.

Laufzeit:
Nur im fünften und in den letzten beiden Schritten werden Anfragen gestellt. Im fünften Schritt wird eine binäre Suche auf $\{0, \ldots, k'\}$ durchgeführt. Hierbei wird der Suchbereich höchstens $\lceil \log(k'+1) \rceil$-mal halbiert, bis dieser einelementig wird. Nach jeder Halbierung werden maximal $k'+1$ Anfragen gestellt. Insgesamt ergibt dies $O(k' \log k')$ Anfragen.
Die Analyse der letzten beiden Schritte ergibt sich wie im Beweis von Satz 3.4. Insgesamt werden höchstens $O(k' \log \frac{n}{k'}) + O(k' \log k')$ Anfragen gestellt. Mit der Voraussetzung $k' = O(n^{1-\epsilon})$ folgt $k' \log k' = O(k' \log \frac{n}{k'})$ aus Lemma B.5 und somit die Behauptung.

□

Ich zeige nun, daß mit den angegebenen Algorithmen für jedes k' und τ ein asymptotisch optimaler Algorithmus für $THR_\tau(n-k')$ bekannt ist.

Satz 3.9: Asymptotisch optimaler Algorithmus für $THR_\tau(n-k')$

Sei $n \in \mathbb{N}$ *und* $k, \tau \in \mathbb{N}_0$ *mit* $0 \leq k' \leq n$.

Dann ist für jede Parameterkombination ein Algorithmus bekannt, der die Klasse $THR_\tau(n-k')$ *in asymptotisch optimaler Zeit lernt.*

Beweis: Für $\tau = 0$ und $\tau > n - k'$ besteht $THR_\tau(n-k')$ nur aus der konstanten Eins- bzw. Null-Funktion.
Für $k' = 0$ und $k' = n$ ist die Lernaufgabe trivial.
Es gilt $THR_1(n-k') = OR^n(n-k')$ und $THR_{n-k'}(n-k') = AND^n(n-k')$. Hierfür werden in [24] asymptotisch optimale Algorithmen beschrieben.
In [24] wird für $THR_\tau(n-k')$ mit $k' \geq \frac{n}{2}$ ein asymptotisch optimaler Algorithmus angegeben.
In [24] wird gezeigt, wie aus einem Algorithmus für $THR_\tau(n-k')$ direkt ein Algorithmus für $THR_{n-k'-\tau+1}(n-k')$ mit gleicher Laufzeit konstruiert werden kann. Deswegen sei im weiteren $\tau \leq \frac{n-k'}{2}$.

Als untere Schranken sind $k' \log \frac{n}{k'}$ und $\frac{n-k'}{\tau}$ bekannt. Ich gebe nun für $n \in \mathbb{N}, 1 \leq k' \leq \frac{n}{2}$ und $2 \leq \tau \leq \frac{n-k'}{2}$ Algorithmen an, deren Laufzeiten sich höchstens um einen konstanten Faktor von diesen unteren Schranken unterscheiden. Dazu führe ich eine vollständige Fallunterscheidung durch.

- **Fall 1:** $k' \geq \frac{n}{12}$. Dann gilt $k' = \Theta(n)$. In diesem Fall ist der lineare Algorithmus aus Satz 2.7 asymptotisch optimal.
- **Fall 2:** $k' < \frac{n}{12}$.
  - **Fall 2.1:** $k' < \lfloor \frac{n}{\tau} \rfloor$. Im Beweis von Satz 3.4 wird ein Algorithmus mit Laufzeit $O(\frac{n-k'}{\tau} + k' \log \frac{n}{k'})$ angegeben.
  - **Fall 2.2:** $k' \geq \lfloor \frac{n}{\tau} \rfloor$.
    - **Fall 2.2.1:** $k' \geq \sqrt{\frac{n}{\log n}}$. Dann gilt $k' = \Omega(\sqrt{\frac{n}{\log n}})$. Aus $k' \geq \lfloor \frac{n}{\tau} \rfloor$ folgt $\tau \geq \left\lfloor \frac{n}{k'+1} \right\rfloor$. Hierfür wird im Beweis von Satz 3.5 ein Algorithmus mit Laufzeit $O(k' \log \frac{n}{k'})$ angegeben.
    - **Fall 2.2.2:** $k' < \sqrt{\frac{n}{\log n}}$. Dann gilt $k' \leq \sqrt{n} = O(n^{\frac{1}{2}})$ und $\frac{n}{3k'} < \tau$. Somit sind die Voraussetzungen von Satz 3.7 erfüllt, in dessen Beweis ein Algorithmus mit Laufzeit $O(k' \log \frac{n}{k'})$ angegeben wird.

$\square$

Kapitel 4

Modulo-Funktionen

Eine Modulo-Funktion mit Modulus q gibt genau dann eine Eins aus, wenn die Anzahl Einsen in der Eingabe kein Vielfaches von q ist. Der Wertevektor einer Modulo-Funktion hat die Gestalt $v = (0, 1, \ldots, 1, 0, 1, \ldots, 1, 0, 1, \ldots)$, wobei zwischen zwei Nullen jeweils $q - 1$ Einsen stehen.
Ich untersuche in diesem Kapitel Klassen von Modulo-Funktionen mit Modulus q und k wesentlichen Variablen.
Die Ergebnisse aus Kapitel 2 für allgemeine symmetrische Funktionen übertragen sich natürlich auf Modulo-Funktionen. Insbesondere ergeben sich hieraus eine untere Schranke $\lceil \log \binom{n}{k} \rceil$ und ein in n linearer Algorithmus. Für $k \leq \frac{n}{2}$ und beliebigen Modulus $q \leq k$ gebe ich asymptotisch optimale Algorithmen an, indem ich entsprechende Aussagen, die in [24] für Parity-Funktionen gezeigt werden, verallgemeinere. Für $k \geq \frac{2n}{3}$ zeige ich, daß für $q = O(\log \binom{n}{k})$ ebenfalls asymptotisch optimale Algorithmen bekannt sind. Für $q > \frac{2n}{3}$ und $k = q + c$ gebe ich eine weitere untere Schranke an. Außerdem zeige ich, daß ein Algorithmus, der nur Anfragen mit höchstens q Variablen verwendet, für $q > \frac{2n}{3}$ und $n - k > 2 \log n$ mindestens $\frac{n}{2}$ Anfragen benötigt, um $MOD_q(k)$ zu lernen.

Definition 4.1: Modulo-Funktion

> *Sei $n, q \in \mathbb{N}$. Eine Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt* **Modulo-Funktion q-ter Ordnung**, *wenn $f(x) = 0$ genau dann gilt, wenn die Anzahl Einsen in x ein Vielfaches von q ist.*
>
> *Man schreibt $f(x) = \mathrm{mod}_q(x)$.*

Definition 4.2: $MOD_q^n(k)$

> *Seien $n, k, q \in \mathbb{N}$ mit $0 \leq k \leq n$. Dann besteht die Klasse $MOD_q^n(k)$ aus allen Modulo-Funktionen q-ter Ordnung auf n Variablen mit k wesentlichen Variablen. Unwesentliche Variablen werden durch Null ersetzt.*

$$MOD_{\mathbf{q}}^{\mathbf{n}}(\mathbf{k}) := \{mod_{q,K} \mid K \subseteq [n], |K| = k\}$$

Ich schreibe auch $MOD_q(k)$, wenn n aus dem Zusammenhang bekannt ist.
Für $k = 0$ und $k = n$ enthält die Klasse $MOD_q(k)$ genau eine Funktion. Für $q = 1$ besteht $MOD_q(k)$ nur aus der konstanten Null-Funktion. Für $q = 2$ liegt der Spezialfall der Parity-Funktionen vor. Für $q \geq k + 1$ erzeugt eine Anfrage genau dann eine Eins, wenn mindestens eine wesentliche Variable in der Eingabe liegt. Dies entspricht einer Disjunktion. Für die beiden letzten Fälle werden in [24] asymptotisch optimale Algorithmen angegeben.
Ich beschränke mich in den folgenden Betrachtungen auf

$$\boxed{n \in \mathbb{N}, 3 \leq k \leq n - 1 \text{ und } 3 \leq q \leq k}$$

da alle anderen Fälle entweder trivial sind oder bereits untersucht wurden. In den drei folgenden Sätzen gebe ich eine untere Schranke sowie einen in n linearen Algorithmus an. Diese Aussagen ergeben sich direkt aus den entsprechenden Sätzen über allgemeine symmetrische Funktionen aus Kapitel 2 und der Form des Wertevektors für Modulo-Funktionen.

Lemma 4.3: Größe von $MOD_q(k)$

Die Klasse $MOD_q(k)$ enthält genau $\binom{n}{k}$ Funktionen.

Beweis: Sei v der Wertevektor einer Funktion aus $MOD_q(k)$. Dann gilt $v_0 = 0, v_1 = 1$. Daher folgt die Behauptung direkt aus Satz 2.3. □

Satz 4.4: Untere Schranke für $MOD_q(k)$

Es sind mindestens $\lceil \log \binom{n}{k} \rceil$ Anfragen notwendig, um $MOD_q(k)$ zu lernen. Es gilt $\lceil \log \binom{n}{k} \rceil \geq k \log \frac{n}{k}$.

Beweis: Es gilt $|MOD_q(k)| = \binom{n}{k}$. Die erste Behauptung folgt direkt aus der allgemeinen unteren Schranke für Funktionenklassen (vgl. Satz 1.2). Die Abschätzung folgt aus Lemma B.1. □

Satz 4.5: Linearer Algorithmus für $MOD_q(k)$

Die Klasse $MOD_q(k)$ kann mit $n - 1$ Anfragen gelernt werden.

Beweis: Sei v der Wertevektor einer Funktion aus $MOD_q(k)$. Dann gilt $v_0 = 0$ und $v_1 = 1$. Die Behauptung folgt direkt aus Satz 2.5. □

Wenn k nahe bei $\frac{n}{2}$ liegt, ist die untere Schranke aus Satz 4.4 linear in n und der obige lineare Algorithmus ist asymptotisch optimal. Daher werde ich mich im Folgenden darauf beschränken, daß die Anzahl der wesentlichen Variablen klein ($k \leq \frac{n}{3}$) oder groß ($k \geq \frac{3n}{4}$) ist.

4.1 Ein Algorithmus für $MOD_q(k)$ mit $k \leq \frac{n}{3}$

In [24, S. 11] wird für die Klasse der Parity-Funktionen mit k wesentlichen Variablen ein asymptotisch optimaler Algorithmus für $k \leq \frac{n}{3}$ angegeben. Diesen werde ich für beliebigen Modulus $q \leq k$ verallgemeinern. Dazu weise ich für $k \leq \frac{n}{3}$ die Existenz eines nicht zu großen Mengensystems $T_1, \ldots, T_m$ nach, so daß für jede Auswahl von höchstens k wesentlichen Variablen mindestens eine Menge T_i existiert, so daß die Anzahl der wesentlichen Variablen in T_i kein Vielfaches von q ist. Da dieser Satz später auch für unwesentliche Variablen verwendet werden soll, werde ich die Aussage allgemeiner für eine Auswahl S aus κ Objekten formulieren.
Mit Hilfe dieses Mengensystems können mit binärer Suche alle Variablen identifiziert werden. Wenn $k = O(n^{1-\epsilon})$ für ein $\epsilon > 0$ gilt, läßt sich hiermit ein asymptotisch optimaler Algorithmus konstruieren. Für größeres $k \leq \frac{n}{3}$ zeige ich, daß ähnliche Mengensysteme existieren, bei denen zusätzlich die Größe der Mengen T_i beschränkt ist. Hiermit kann für beliebiges $k \leq \frac{n}{3}$ ein asymptotisch optimaler Algorithmus angegeben werden.

Satz 4.6: Existenz eines Mengensystems

Seien $n, \kappa, q \in \mathbb{N}$ mit $3 \leq \kappa \leq \frac{n}{3}$ und $3 \leq q \leq \kappa$. Sei $m = \lceil \log \binom{n}{\kappa} \rceil + 2$.

Dann gibt es ein System von m Mengen $T_i \subseteq [n]$, so daß für jedes $S \subseteq [n]$ mit $1 \leq |S| \leq \kappa$ mindestens ein Index i existiert mit $|S \cap T_i| \not\equiv 0 \bmod q$.

Beweis: Sei $S \subseteq [n]$ mit $1 \leq |S| \leq \kappa$. Eine Menge $R \subseteq [n]$ soll *c-schlecht* heißen, wenn $|S \cap R| \equiv c \bmod q$ gilt. Ein Tupel von m Mengen $(R_1, \ldots, R_m)$ heißt *c-schlecht*, wenn $|S \cap R_i| \equiv c \bmod q$ für jedes $1 \leq i \leq m$ gilt.
Ich zeige zunächst, daß für jedes $0 \leq c \leq q-1$ höchstens die Hälfte aller Mengen $R \subseteq [n]$ c-schlecht ist. Dazu führe ich eine Induktion über die Größe s der Menge S durch.

- IA: $s = 1$. Sei $S = \{x\}$. Das Objekt x liegt genau in der Hälfte aller Mengen $R \subseteq [n]$. Also ist die Hälfte aller Menge 0-schlecht, die andere Hälfte ist 1-schlecht. Für größeres c gibt es keine c-schlechten Mengen.

- IS: $s \to s+1$. Sei $S = V \cup \{x\}$ mit $|V| = s$. Eine Menge R ist c-schlecht bzgl. S, wenn sie entweder $(c-1)$-schlecht bzgl. V ist und x enthält oder c-schlecht bzgl. V ist und x nicht enthält.

  Die Induktionsvoraussetzung besagt, daß höchstens die Hälfte aller Mengen c-schlecht bzgl. V ist, d. h. für höchstens die Hälfte aller Mengen gilt $|R \cap V| \equiv c \bmod q$. Da x nicht in V liegt, enthält genau die Hälfte dieser Mengen R das Objekt x und die andere Hälfte nicht.

Also gilt für höchstens ein Viertel aller möglichen Mengen R, daß sie c-schlecht bzgl. V sind und x nicht enthalten.

Nach Induktionsvoraussetzung gilt ebenfalls, daß höchstens die Hälfte aller Mengen $(c-1)$-schlecht bzgl. V ist. Von diesen enthält genau die Hälfte das Objekt x. Also gilt für höchstens ein Viertel aller möglichen Mengen R, daß sie $c-1$-schlecht bzgl. V ist und x enthält.

Insgesamt sind also höchstens die Hälfte aller Mengen c-schlecht bzgl. S.

Ich zeige nun, daß höchstens ein Anteil von $(\frac{1}{2})^m$ aller möglichen Tupel von m Mengen c-schlecht ist.

Mit $\mathcal{P}([n])$ bezeichne ich die Potenzmenge $[n]$. Dann können die Mengen aus $\mathcal{P}([n])$ so auf einer Achse angeordnet werden, daß alle c-schlechten Mengen in der ersten Hälfte liegen, da höchstens die Hälfte aller Mengen c-schlecht ist. Die Länge dieser Achse sei auf Eins normiert. Jedes Tupel von m Mengen liegt in dem m-dimensionalen Raum $(\mathcal{P}([n]))^m$. Die c-schlechten Tupel liegen alle in einem Quader mit Seitenlänge $\frac{1}{2}$. Das Volumen dieses Quaders beträgt $(\frac{1}{2})^m$.

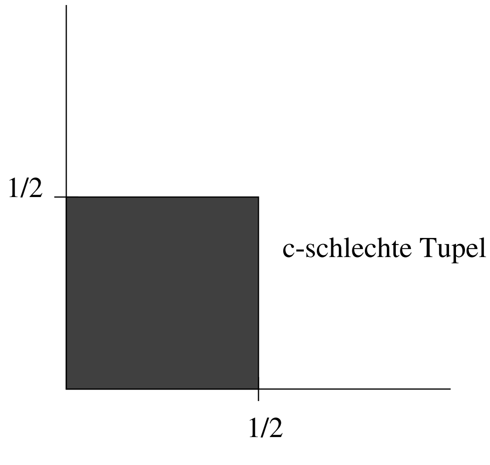

Abbildung 4.1: Der Raum $(\mathcal{P}([n]))^m$ für $m = 2$

Ich betrachte nun den Fall $c = 0$. Die Anzahl aller Mengen $S \subseteq [n]$ mit $1 \leq |S| \leq \kappa$ beträgt $\sum_{i=1}^{\kappa} \binom{n}{i}$. Für $\kappa \leq \frac{n}{3}$ sind dies nach Lemma B.2 höchstens $2\binom{n}{\kappa}$ Mengen. Durch jede dieser Mengen wird maximal ein Anteil von $(\frac{1}{2})^m$

aller möglichen m-Tupel 0-schlecht. Also sind maximal

$$\begin{aligned} 2\binom{n}{\kappa}\cdot\left(\frac{1}{2}\right)^m &= 2\binom{n}{\kappa}\cdot\left(\frac{1}{2}\right)^{\lceil\log\binom{n}{\kappa}\rceil+2} \\ &\leq \frac{1}{2}\binom{n}{\kappa}\cdot\left(\frac{1}{2}\right)^{\log\binom{n}{\kappa}} \\ &\leq \frac{1}{2} \end{aligned}$$

aller m-Tupel 0-schlecht für irgendeine Menge S. Daher muß es mindestens ein Tupel $T_1,\ldots,T_m$ von m Mengen geben, das für keine Menge S mit $1\leq|S|\leq\kappa$ die Eigenschaft 0-schlecht erfüllt. Dies ist die Behauptung.

□

Ich verwende das obige Mengensystem $T_1,\ldots,T_m$, um einen Algorithmus für $MOD_q(k)$ mit $k\leq\frac{n}{3}$ zu beschreiben. Sei $K=S$ die (unbekannte) Menge der wesentlichen Variablen mit $|K|=k=\kappa$. Dann gibt es eine Menge T_j, so daß $|K\cap T_j|\not\equiv 0 \bmod q$ gilt. Die Anfrage T_j liefert also eine Eins. Mit einer binären Suche kann eine wesentliche Variable x in T_j gefunden werden, indem T_j halbiert wird und beide Hälften angefragt werden. Da die Anzahl wesentlicher Variablen in T_j kein Vielfaches von q ist, muß dies auch für mindestens eine der beiden Hälften gelten, so daß mindestens eine der beiden Anfragen eine Eins liefern muß. In dem entsprechenden Teil wird weitergesucht.

Sei $K'=K\setminus\{x\}$ die (unbekannte) Menge der verbliebenen wesentlichen Variablen, nachdem die Variable x als wesentlich identifiziert wurde. Falls $|K'|\geq 1$ gilt, so existiert eine Menge $T_{j'}$, so daß $|K'\cap T_{j'}|\not\equiv 0 \bmod q$. Da x nicht in diesem Schnitt vorkommt, liefert die Anfrage $T_{j'}\setminus\{x\}$ eine Eins. Eine solche Menge $T_{j'}\setminus\{x\}$ kann zum Beispiel gefunden werden, indem x aus allen Mengen T_i entfernt und jede dieser Mengen angefragt wird. In $T_{j'}$ kann mit binärer Suche eine weitere wesentliche Variable identifiziert werden. Auf diese Weise können sukzessive alle wesentlichen Variablen gelernt werden.

Die Laufzeit dieses Algorithmus ist $O(k*(m+\log n))$, da für jede der k wesentlichen Variablen alle (entsprechend verkleinerten) Mengen T_i gefragt werden und jeweils auf einer der Mengen eine binäre Suche durchgeführt wird. Der obige Algorithmus hat den Nachteil, daß für jede wesentliche Variable *alle* Mengen T_i angefragt werden, um herauszufinden, auf welcher Menge eine binäre Suche starten kann.

Ich zeige nun, daß $O(m+k\log n)$ Anfragen ausreichen, um $MOD_q(k)$ für $k\leq\frac{n}{3}$ zu lernen. Sei A eine Anfrage, die eine Null liefert. Wenn im weiteren Verlauf des Algorithmus g wesentliche Variablen gefunden werden, die in A liegen, so ist ohne erneute Anfrage klar, daß noch weitere $(q-g) \bmod q$ wesentliche Variablen in der Menge A liegen müssen. Im Beweis des folgenden Satzes gebe ich einen Algorithmus an, der dieses Wissen ausnutzt, um stets auf geeigneten Mengen binäre Suchen zu starten, ohne zuvor weitere Anfragen stellen zu müssen.

Satz 4.7: Algorithmus für $MOD_q(k)$ mit $k \leq \frac{n}{3}$

> *Sei $k \leq \frac{n}{3}$. Dann kann $MOD_q(k)$ mit $O(k \log n + \log \binom{n}{k})$ Anfragen gelernt werden.*

Beweis: Der folgende Algorithmus beruht auf dem Mengensystem aus Satz 4.6. Dort wird gezeigt, daß für $\kappa \leq \frac{n}{3}$ und $m = \lceil \log \binom{n}{\kappa} \rceil + 2$ Mengen $T_i \subseteq [n]$ existieren, so daß für jede Menge $S \subseteq [n]$ mit $1 \leq |S| \leq \kappa$ mindestens ein Index i existiert mit $|S \cap T_i| \not\equiv 0 \bmod q$. Ich identifiziere die Menge S mit der Auswahl der wesentlichen Variablen K und κ mit k.
Im folgenden Algorithmus bezeichnet G die Menge der bereits bekannten wesentlichen Variablen. Es werden Mengen M_i verwendet, die aus T_i durch Entfernen von wesentlichen Variablen entstehen. Für einen Index $maxi$ gilt stets, daß für jedes $i < maxi$ die Anfrage M_i eine Null liefern würde und für jedes $j > maxi$ die Menge M_j noch nie gefragt wurde. Der Index $maxi$ wächst im Verlauf des Algorithmus.
Der Algorithmus stoppt, sobald $|G| = k$ gilt, also alle wesentlichen Variablen gefunden sind.

Algorithmus 4.8:

1. $maxi = 1$
   $G = \emptyset$
   $\forall i = 1 \ldots m : M_i = T_i$
2. Falls ein $j < maxi$ existiert mit $|M_j \cap G| \not\equiv 0 \bmod q$:
   - $l = q - (|M_j \cap G| \bmod q)$
   - Bestimme mit binärer Suche eine wesentliche Variable $x \in M_j \setminus G$
     $G = G \cup \{x\}$
   - Wiederhole den letzten Schritt, bis genau l wesentliche Variablen gefunden wurden
   - $M_j = M_j \setminus G$
   - weiter bei Schritt 2
3. $M_{maxi} = M_{maxi} \setminus G$
4. Frage M_{maxi}
   Falls die Antwort Null ist:
   - $maxi = maxi + 1$
   - weiter bei Schritt 3

5. Bestimme mit binärer Suche eine wesentliche Variable $x \in M_{maxi}$

   $G = G \cup \{x\}$

   weiter bei Schritt 2

Bevor ich die Korrektheit und die Laufzeit untersuche, möchte ich die Arbeitsweise des Algorithmus in einem Beispiel veranschaulichen. Sei dazu $q = 3$, $T_1 = \{a, b, c, d, e, f\}, T_2 = \{a, b, c, d, h, i, j, k, l\}$ und $T_3 = \{b, c, d, e\}$. Ich nehme an, daß dies alles wesentliche Variablen sind, da unwesentliche Variablen das Verhalten des Algorithmus nicht beeinflussen. Zu Beginn gilt $maxi = 1$ und $M_i = T_i$ für alle i. Die Menge M_1 wird angefragt und liefert eine Null. Der Zähler $maxi$ wird auf 2 erhöht und M_2 angefragt. Auch hier ist die Antwort Null, $maxi$ wird erneut erhöht. Da die Anfrage M_3 eine Eins liefert, kann der Algorithmus mit einer binären Suche eine wesentliche Variable in M_3 identifizieren, z. B. b, und in G eingefügen. Anschließend bestimmt der Algorithmus, wie groß der Schnitt von G und M_1 ist. Dieser ist einelementig. Da die Anfrage M_1 eine Null liefert, müssen noch mindestens zwei weitere wesentliche Variablen in dieser Menge liegen. Mit binärer Suche werden die Variablen a und c identifiziert und in G eingefügt. Die Variablen a, b, c werden aus M_1 entfernt. Anschließend würde die neue Menge M_1 wieder eine Null als Antwort erzeugen, wenn sie als Anfrage gestellt würde. Jetzt bestimmt der Algorithmus den Durchschnitt von G und M_2. Dieser ist dreielementig, also trifft die Bedingung in Schritt 2 nicht zu. In M_2 können im Moment keine Variablen identifiziert werden.
Der Algorithmus entfernt alle Variablen aus M_3, die in G liegen. Dies sind b und c. Die neue Menge M_3 wird als Anfrage gestellt und liefert eine Eins. Mit einer binären Suche wird d als wesentlich identifiziert und in G eingefügt. Anschließend beginnt erneut die Untersuchung von M_1. Auf diese Weise fährt der Algorithmus fort, bis die Anfrage M_3 eine Null liefert. Dann wird $maxi$ erhöht und M_4 gefragt. Sukzessive werden so alle wesentlichen Variablen gelernt.

Korrektheit:
Die Menge G enthält stets alle bereits gefundenen wesentlichen Variablen. Um die Korrektheit nachzuweisen, zeige ich, daß der Algorithmus eine weitere wesentliche Variable findet, solange $|G| < k$ gilt.
Wenn von einer Menge A bekannt ist, daß sie eine Eins liefert, kann mit einer binären Suche eine wesentliche Variable in A bestimmt werden. Dazu wird A halbiert und beide Hälften werden angefragt. Mindestens eine der beiden Hälften muß eine Eins liefern, da die Anzahl wesentlicher Variablen in A nicht durch q teilbar ist und dies auch für mindestens eine Hälfte gelten muß.
Wenn eine Anfrage B eine Null liefert und g wesentliche Variablen aus B bekannt sind, so können weitere $(q - g) \bmod q$ wesentliche Variablen in B mit binärer Suche gefunden werden.

Ich zeige nun, daß für jedes $i < maxi$ die Anfrage M_i eine Null liefern würde. Am Anfang ist diese Aussage korrekt, da kein $i < maxi$ existiert. Im Verlauf des Algorithmus kann sich der Wahrheitsgehalt dieser Aussage nur dann ändern, wenn $maxi$ steigt oder eine Menge M_j mit $j < maxi$ verändert wird. Der Index $maxi$ steigt nur in Schritt 4 und nur dann, wenn M_{maxi} eine Null liefert. Wenn vorher für alle $i < maxi$ die Anfragen M_i eine Null geliefert haben, so stimmt dies auch nach Schritt 4 noch. Nur in Schritt 2 wird eine Menge M_j mit $j < maxi$ verändert. Wenn M_j eine Null liefert, muß auch das neue $M_j = M_j \setminus G$ eine Null liefern, da nach den binären Suchen die Anzahl der wesentlichen Variablen aus M_j, die in G liegen, ein Vielfaches von q ist.
Wenn in Schritt 2 ein Index $j < maxi$ existiert, so daß $|M_j \cap G| \not\equiv 0 \bmod q$ gilt, so muß die Anfrage $M_j \setminus G$ eine Eins liefern. Das liegt daran, daß M_j als Anfrage eine Null liefern würde. Daher kann die Anzahl wesentlicher Variablen in $M_j \setminus G$ kein Vielfaches von q sein. Also können wesentliche Variablen in M_j gefunden werden und es reicht für die Korrektheit zu zeigen, daß folgende Aussage gilt:

(Ψ) Solange $|G| < k$ gilt, gibt es ein $j_0 < maxi$ mit $|M_{j_0} \cap G| \not\equiv 0 \bmod q$ oder es existiert ein $i_0 \geq maxi$, so daß die Anfrage $M_{i_0} \setminus G$ eine Eins liefert, und der Algorithmus findet ein geeignetes j_0 oder i_0.

Ich zeige zunächst zwei Hilfsaussagen:

(H1) Sei K die (unbekannte) Menge der k wesentlichen Variablen. Solange $|G| < k$ gilt, gibt es mindestens einen Index $\nu \in \{1, \ldots, m\}$, so daß $|(T_\nu \setminus G) \cap K| \not\equiv 0 \bmod q$ gilt, d. h. die Anfrage $T_\nu \setminus G$ würde eine Eins liefern.

Beweis: Es gilt
$$(T_i \setminus G) \cap K = T_i \cap (K \setminus G)$$
für jedes $i \in [m]$. Die Menge $K \setminus G$ enthält genau die noch nicht gefundenen wesentlichen Variablen. Solange $|G| < k$ gilt, ist diese Menge mindestens einelementig, also folgt die Behauptung direkt aus der Eigenschaft des Mengensystems $T_1, \ldots, T_m$ aus Satz 4.6.

(H2) Wenn ein $j < maxi$ existiert, so daß $T_j \setminus G$ eine Eins liefert, so gilt $|M_j \cap G| \not\equiv 0 \bmod q$

Beweis: Sei $j < maxi$ so, daß $T_j \setminus G$ eine Eins liefert. Da M_j aus T_j durch Entfernen der wesentlichen Variablen aus G entsteht, gilt $T_j \setminus G = M_j \setminus G$. Daher liefert auch $M_j \setminus G$ als Anfrage eine Eins. Da $j < maxi$ gilt, liefert die Anfrage M_j eine Null. Also muß $|M_j \cap G| \not\equiv 0 \bmod q$ gelten.

Ich beweise nun die Aussage Ψ. Zu Beginn des Algorithmus gilt $M_i = T_i$ für alle i, $maxi = 1$ und $G = \emptyset$. Wegen der Eigenschaft des Mengensystems $T_1, \dots, T_m$ gibt es mindestens ein i, so daß $M_i \setminus G$ eine Eins liefert. Im Verlauf des Algorithmus wird G ständig vergrößert. Jedesmal, wenn sich G ändert, wird danach Schritt 2 solange ausgeführt, bis $|M_i \cap G| \equiv 0 \bmod q$ für alle $i < maxi$ gilt. Solange also noch ein $j_0 < maxi$ existiert mit $|M_{j_0} \cap G| \not\equiv 0 \bmod q$, wird dieses auch gefunden. Was passiert, wenn kein solches $j_0 < maxi$ existiert? Dann gibt es nach der zweiten Hilfsaussage auch keine Menge T_{j_0} mit $j_0 < maxi$, so daß $T_{j_0} \setminus G$ eine Eins liefert. Also muß wegen der ersten Hilfsaussage ein $i \geq maxi$ existieren, so daß $T_i \setminus G$ eine Eins liefert. Sei $i_0 \geq maxi$ minimal, so daß $T_{i_0} \setminus G$ eine Eins liefert. Da die Bedingung von Schritt 2 aus dem Algorithmus nicht erfüllt ist, wird solange $maxi$ erhöht, bis $maxi = i_0$ gilt. Dann liefert die Anfrage $M_{i_0} \setminus G$ eine Eins.

Laufzeit:

Jede Menge M_i hat höchstens n Variablen, eine binäre Suche benötigt also $O(\log n)$ Anfragen. Mit jeder binären Suche wird eine wesentliche Variable gefunden, insgesamt benötigen diese Suchen also höchstens $O(k \log n)$ Anfragen.

Außer in den binären Suchen werden nur in Schritt 4 Anfragen gestellt. Dort wird M_{maxi} gefragt. Nach jeder Anfrage wird entweder eine wesentliche Variable gefunden oder der Zähler $maxi$ um Eins erhöht. Da es genau k wesentliche Variablen gibt, der Zähler $maxi$ nie vermindert wird und stets $maxi \leq m$ gilt, werden im Schritt 4 höchstens $k + m$ Anfragen gestellt. □

Der obige Algorithmus verwendet die Mengen T_i aus dem Mengensystem $T_1, \dots, T_m$ als Anfragen, um $MOD_q(k)$ für $k \leq \frac{n}{3}$ zu lernen. Aus Satz 4.6 ist bekannt, daß solche Mengen existieren. Solche Systeme können konstruiert werden, ohne Anfragen zu stellen. Allerdings kann der Aufwand für die Konstruktion der Mengen $T_1, \dots, T_m$ sehr groß sein, zumindest ist mir kein schnelles Verfahren hierfür bekannt. Der Aufwand, um diesen Algorithmus tatsächlich durchzuführen, ist also möglicherweise sehr groß, auch wenn die Anzahl der Anfragen, die der Algorithmus stellt, nur klein ist.

Für $k = O(n^{1-\epsilon})$ ist Algorithmus 4.8 bereits asymptotisch optimal (vgl. Lemma B.6). Für k nahe bei $\frac{n}{2}$ ist der in n lineare Algorithmus aus Satz 4.5 asymptotisch optimal. Es bleibt der Fall offen, daß $k \leq \frac{n}{3}$ gilt und k nicht sehr klein ist. Dies kann dadurch ausgedrückt werden, daß $k(n) = \frac{n}{h(n)}$ gilt für eine Funktion h, die nicht zu schnell gegen Unendlich strebt. Dies kann z. B. $h(n) = \log n$ sein. Da die Funktion h von n abhängt, ist auch k somit eine Funktion von n. Ich werde im Folgenden stets annehmen, daß der Term $\frac{n}{h(n)}$ ganzzahlig ist. Dies ist keine wesentliche Einschränkung an die Funktion $h(n)$.

In [24] wird für eine analoge Situation für Parity-Funktionen ein asymptotisch optimaler Algorithmus angegeben. Ich werde diesen Ansatz verallge-

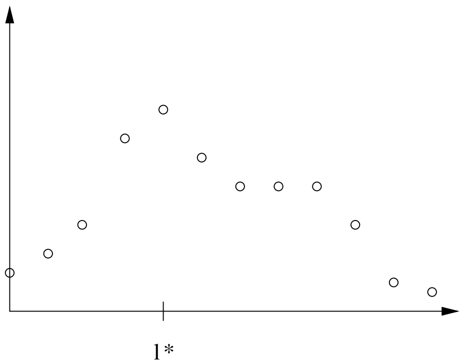

Abbildung 4.2: Beispiel einer unimodalen Funktion

meinern und zeigen, daß es für solche $k(n)$ ein System von „kleinen" Mengen $T_1, \ldots, T_m$ gibt, mit dem ähnlich wie im obigen Algorithmus einige wesentliche Variablen gelernt werden können. Mit mehreren solcher Mengensysteme kann anschließend ein effizienter Algorithmus für $MOD_q(k)$ konstruiert werden.

Sei $k(n) = \frac{n}{h(n)}$ für eine geeignete Funktion h. Ich zeige zunächst, daß für eine zufällige Menge R mit genau $h(n)^2$ Elementen die Wahrscheinlichkeit, daß die Anzahl der wesentlichen Variablen in R ein Vielfaches von q ist, höchstens $\frac{9}{10}$ beträgt. Hiermit kann anschließend die Existenz eines geeigneten Mengensystems nachgewiesen werden. Die folgenden Aussagen gebe ich für beliebige Objekte an, das sie sowohl für wesentliche als auch für unwesentliche Variablen verwendet werden sollen.

In dem Beweis werden unimodale Funktionen verwendet. Diese sind zunächst monoton steigend und anschließend monoton fallend.

Definition 4.9: Unimodale Funktion

> *Sei $m \in \mathbb{N}$, $M = \{0, \ldots, m\}$ und $f : M \to \mathbb{R}$. Die Funktion f heißt* **unimodal**, *wenn es ein $l^* \in M$ gibt, so daß gilt:*
>
> - *Für jedes $l \in M$ mit $l < l^*$ gilt $f(l) \leq f(l+1)$.*
> - *Für jedes $l \in M$ mit $l^* \leq l - 1$ gilt $f(l-1) \geq f(l)$.*

Satz 4.10: Wahrscheinlichkeitsabschätzung

> *Sei $h : \mathbb{N} \to \mathbb{N}$ eine Funktion, die gegen Unendlich strebt. Für jedes $\epsilon > 0$ und jedes $c > 0$ gelte $h(n) < cn^\epsilon$ für $n \in \mathbb{N}$ groß genug.*
>
> *Sei $S \subseteq [n]$ mit $\frac{n}{h(n)^2} \leq |S| \leq \frac{n}{h(n)}$. Sei $R \subseteq [n]$ eine zufällige Menge mit $|R| = h(n)^2$.*

Dann gilt für n groß genug $P(|S \cap R| \equiv 0 \bmod q) \leq \frac{9}{10}$.

Beweis: Ich betrachte im Folgenden stets Aussagen über den Durchschnitt von S und R und schreibe $P(l)$ statt $P(|S \cap R| = l)$, $P(\equiv \lambda)$ statt $P(|S \cap R| \equiv \lambda \bmod q)$ und $P(\not\equiv \lambda)$ entsprechend.
Die Größen von S und R hängen von n ab. Sei $s(n) = |S|$ und $r(n) = |R|$. Für großes n gilt dann

$$r(n) = h(n)^2 \leq \frac{n}{h(n)^2} \leq s(n)$$

da andernfalls $h(n) > n^{\frac{1}{4}}$ gelten würde im Widerspruch zur Voraussetzung.

Ich zeige folgende Aussagen:

(A1) Für $0 \leq l \leq r(n)$ gilt

$$P(l) = \frac{\binom{r(n)}{l}\binom{n-r(n)}{s(n)-l}}{\binom{n}{s(n)}}$$

(A2) Die Verteilung P ist unimodal und nimmt den Maximalwert bei

$$l^*(n) := \left\lfloor \frac{r(n)s(n) + r(n) + s(n) - n - 1}{n+2} \right\rfloor + 1$$

an.

(A3) Es gilt

$$\lim_{n\to\infty} l^*(n) = \left\lfloor \frac{r(n)s(n)}{n} \right\rfloor$$

(A4) Für genügend großes n gilt

$$\frac{P(l^*(n)+1)}{P(l^*(n))} \geq \frac{1}{4}$$

(A5) Für genügend großes n gilt für jedes $l \in \{0, \ldots, r(n)\}$: $P(l) \leq \frac{4}{5}$

(A6) Für genügend großes n gilt

$$P(\equiv 0) \leq \frac{1 + P(l^*(n))}{2} \leq \frac{9}{10}$$

Aus der letzten Aussage folgt direkt die Behauptung.

zu A1:

Die Zufallsvariable l ist hypergeometrisch verteilt (vgl. [23, S. 139]). Für festes $l \in \{0, \ldots, r(n)\}$ gilt

$$P(l) = \frac{\binom{r(n)}{l}\binom{n-r(n)}{s(n)-l}}{\binom{n}{s(n)}}$$

zu A2:
Ich betrachte in diesem Teil einen festen Wert n und schreibe im Folgenden r und s statt $r(n)$ und $s(n)$.

$$\begin{aligned}
& P(l) \leq P(l+1) \\
\Longleftrightarrow\ & \frac{\binom{r}{l}\binom{n-r}{s-l}}{\binom{n}{s}} \leq \frac{\binom{r}{l+1}\binom{n-r}{s-l-1}}{\binom{n}{s}} \\
\Longleftrightarrow\ & \frac{r!(n-r)!}{l!(r-l)!(s-l)!(n-r-s+l)!} \leq \frac{r!(n-r)!}{(l+1)!(r-l+1)!(s-l-1)!(n-r-s+l+1)!} \\
\Longleftrightarrow\ & (l+1)(n-r-s+l+1) \leq (r-l)(s-l) \\
\Longleftrightarrow\ & nl - rl - sl + l^2 + l + n - r - s + l + 1 \leq rs - rl - sl + l^2 \\
\Longleftrightarrow\ & nl + 2l \leq rs + r + s - n - 1 \\
\Longleftrightarrow\ & l \leq \frac{rs + r + s - n - 1}{n+2}
\end{aligned}$$

Analog läßt sich

$$P(l) \geq P(l+1) \iff l \geq \frac{rs + r + s - n - 1}{n+2}$$

zeigen.
Sei

$$l^*(n) := \left\lfloor \frac{r(n)s(n) + r(n) + s(n) - n - 1}{n+2} \right\rfloor + 1$$

Für ganzzahliges $l < l^*(n)$ gilt $l \leq l^*(n) - 1$ und somit $P(l) \leq P(l+1)$.
Für $l \geq l^*(n)$ gilt $P(l) \geq P(l+1)$.
Also ist P unimodal. Es gilt $1 \leq l^*(n) \leq r(n)$.

zu A3:
Für $n \to \infty$ konvergieren $\frac{r(n)}{n+2}$ und $\frac{s(n)}{n+2}$ gegen Null. Hiermit folgt

$$\begin{aligned}
l^*(n) &= \left\lfloor \frac{r(n)s(n) + r(n) + s(n) - n - 1}{n+2} \right\rfloor + 1 \\
&= \left\lfloor \frac{r(n)s(n)}{n+2} + \frac{r(n)}{n+2} + \frac{s(n)}{n+2} - \frac{n}{n+2} - \frac{1}{n+2} \right\rfloor + 1 \\
&\to \left\lfloor \frac{r(n)s(n)}{n} + 0 + 0 - 1 + 0 \right\rfloor + 1 = \left\lfloor \frac{r(n)s(n)}{n} \right\rfloor
\end{aligned}$$

zu A4:
Auch in diesem Teil werde ich für die folgenden Ungleichungen r, s und l^* statt $r(n), s(n)$ bzw. $l^*(n)$ verwenden.

$$
\begin{aligned}
& \frac{P(l^*+1)}{P(l^*)} \geq \frac{1}{4} \\
\Longleftrightarrow\ & \frac{\binom{r}{l^*+1}\binom{n-r}{s-l^*-1}}{\binom{r}{l^*}\binom{n-r}{s-l^*}} \geq \frac{1}{4} \\
\Longleftrightarrow\ & \frac{r!(n-r)!(l^*)!(r-l^*)!(s-l^*)!(n-r-s+l^*)!}{(l^*+1)!(r-l^*-1)!(s-l^*-1)!(n-r-s+l^*+1)!r!(n-r)!} \geq \frac{1}{4} \\
\Longleftrightarrow\ & \frac{(r-l^*)(s-l^*)}{(l^*+1)(n-r-s+l^*+1)} \geq \frac{1}{4} \\
\Longleftrightarrow\ & rs - rl^* - sl^* + (l^*)^2 \geq \frac{1}{4}(nl^* - rl^* - sl^* + (l^*)^2 + l^* + n - r - s + l^* + 1) \\
\Longleftrightarrow\ & rs + (l^*)^2 + \frac{1}{4}(rl^* + sl^* + r + s) \geq rl^* + sl^* + \frac{1}{4}nl^* + \frac{1}{4}(l^*)^2 + \frac{2}{4}l^* + \frac{n}{4} + \frac{1}{4} \qquad **
\end{aligned}
$$

Ab hier verwende ich wieder die Parameter-Schreibweise, da nun n geeignet gewählt werden soll, damit die letzte Ungleichung zutrifft.
Der linke Teil der Ungleichung ** ist größer als $r(n)s(n)$, da alle Summanden positiv sind. Für genügend großes n gilt

$$1 \leq l^*(n) \leq r(n) \leq \frac{n}{10} \text{ und } s(n) \leq \frac{n}{10}$$

Hiermit folgt

$$4r(n) + 4s(n) + l^*(n) + 2 + \frac{1}{l^*(n)} < n$$

Durch Multiplikation mit $\frac{l^*(n)}{4}$ ergibt sich, daß der rechte Teil der Ungleichung ** kleiner als

$$\frac{1}{4}nl^*(n) + \frac{1}{4}nl^*(n) + \frac{n}{4}$$

ist. Dies kann wegen $l^*(n) \geq 1$ weiter durch $\frac{3}{4}nl^*(n)$ abgeschätzt werden. Für großes n konvergiert $l^*(n)$ gegen $\left\lfloor \frac{r(n)s(n)}{n} \right\rfloor$. Für genügend großes n gilt also

$$l^*(n) < \frac{4}{3}\left\lfloor \frac{r(n)s(n)}{n} \right\rfloor$$

und die rechte Seite der obigen Ungleichung ** ist insbesondere kleiner als $r(n)s(n)$. Daher gilt für genügend großes n die Abschätzung

$$\frac{P(l^*(n)+1)}{P(l^*(n))} \geq \frac{1}{4}$$

zu A5:

Nach Aussage A4 gilt

$$\frac{P(l^*(n)+1)}{P(l^*(n))} \geq \frac{1}{4}$$

wenn n groß genug ist. Außerdem gilt $P(l^*(n)+1) + P(l^*(n)) \leq 1$. Hieraus folgt

$$\begin{aligned} & 1 - P(l^*(n)) \geq \frac{1}{4} P(l^*(n)) \\ \iff \quad & \frac{4}{5} \geq P(l^*(n)) \end{aligned}$$

Da die Funktion P in $l^*(n)$ ihr Maximum für ganzzahlige Parameter annimmt, gilt $P(l) \leq P(l^*(n))$ für n groß genug und alle $l \in \{0, \ldots, r(n)\}$.

zu A6:

Sei n genügend groß, so daß $P(l) \leq \frac{4}{5}$ für jedes $l \in \{0, \ldots, r(n)\}$ zutrifft. In den folgenden Abschätzungen betrachte ich einen festen Wert n und schreibe r, a, b, c, d, l^* statt $r(n), a(n), b(n), c(n), d(n), l^*(n)$.

Sei $r = aq + b$ und $l^* = cq + d$ mit $a, c \in \mathbb{N}_0$ und $0 \leq b, d < q$. Dann gilt $a \geq c$ und

$$\begin{aligned} P(\equiv 0) &= \sum_{i=0}^{a} P(iq) \\ P(\not\equiv 0) &= \sum_{i=0}^{a-1} \sum_{j=1}^{q-1} P(iq+j) + \sum_{i=1}^{b} P(aq+i) \end{aligned}$$

Um zu zeigen, daß $P(\equiv 0) \leq 1 - (P(\equiv 0) - P(cq))$ gilt, schätze ich $P(\equiv 0)$ geeignet ab. Hierbei verwende ich:

- Wenn für beliebiges i alle $P(iq+j)$ mit $1 \leq j \leq q-1$ summiert werden, so kann dies nach unten durch $P(iq+1) + P(iq+(q-1))$ abgeschätzt werden.

- Nach Aussage A2 ist die Funktion P unimodal. Daraus folgt für $i \leq c-1$ die Abschätzung $P(iq+1) \geq P(iq)$ und für $i \geq c$ die Ungleichung $P((i+1)q-1) \geq P((i+1)q)$.

Also gilt

$$\begin{aligned}
P(\equiv 0) &= 1 - P(\not\equiv 0) \\
&= 1 - \left(\sum_{i=0}^{a-1} \sum_{j=1}^{q-1} P(iq+j) + \sum_{i=1}^{b} P(aq+i) \right) \\
&\leq 1 - \sum_{i=0}^{a-1} \left(P(iq+1) + P(iq+q-1) \right) \\
&\leq 1 - \left(\sum_{i=0}^{c-1} P(iq+1) + \sum_{i=c}^{a-1} P((i+1)q-1) \right) \\
&\leq 1 - \left(\sum_{i=0}^{c-1} P(iq) + \sum_{i=c}^{a-1} P((i+1)q) \right) \\
&= 1 - \left(\sum_{i=0}^{c-1} P(iq) + \sum_{i=c+1}^{a} P(iq) \right) \\
&= 1 - \left(P(\equiv 0) - P(cq) \right)
\end{aligned}$$

Da $P(cq) \leq P(l^*)$ gilt, ergibt sich

$$P(\equiv 0) \leq \frac{1 + P(l^*)}{2}$$

Mit der Abschätzung $P(l^*) \leq \frac{4}{5}$ aus Aussage A5 folgt $P(\equiv 0) \leq \frac{9}{10}$.

□

Ich verwende den obigen Satz, um einen weiteren Algorithmus für $MOD_q(k)$ mit $k \leq \frac{n}{3}$ zu beschreiben. Dieser verläuft ähnlich wie Algorithmus 4.8, verwendet allerdings mehrere geeignete Mengensysteme nacheinander, um alle wesentlichen Variablen zu lernen.
Die Anzahl der wesentlichen Variablen ist im folgenden Satz von n abhängig. Ich verwende die Schreibweise k statt $k(n)$ und k_j statt $k_j(n)$. Im folgenden Beweis werden Terme der Form $\frac{n}{h(n)^j}$ vorkommen. Diese sind im Allgemeinen nicht ganzzahlig. Ich verzichte darauf, die Gaußklammern stets anzugeben, da sich hierdurch keine wesentlichen Änderungen in den Rechnungen ergeben.

Satz 4.11: Algorithmus für $MOD_q(k)$ für kleines k

> *Sei $h : \mathbb{N} \to \mathbb{N}$ eine Funktion, die gegen Unendlich strebt. Für jedes $\epsilon > 0$ und jedes $c > 0$ gelte $h(n) < cn^\epsilon$ für n groß genug.*
>
> *Sei $k = \frac{n}{h(n)}$ und $3 \leq k \leq \frac{n}{3}$.*
>
> *Dann kann die Klasse $MOD_q(k)$ mit $O\left(k \log \frac{n}{k}\right)$ Anfragen gelernt werden.*

Beweis: Ich zeige zunächst, wie alle bis auf $\frac{n}{h(n)^2}$ wesentlichen Variablen gelernt werden können.
Sei

$$m := \left\lceil 7 \log \binom{n}{k} \right\rceil + 7$$

In Satz 4.10 wird gezeigt, daß für $S \subseteq [n]$ mit

$$\frac{n}{h(n)^2} \leq |S| \leq \frac{n}{h(n)}$$

und zufälliges $R \subseteq [n]$ mit $|R| = h(n)^2$ die Abschätzung

$$P(|S \cap R| \equiv 0 \bmod q) \leq \frac{9}{10}$$

gilt. Mengen R, für die $|S \cap R| \equiv 0 \bmod q$ gilt, sollen „schlecht für S" heißen. Also gilt $P(R \text{ ist schlecht für } S) \leq \frac{9}{10}$. Ich identifiziere S mit der Auswahl K der wesentlichen Variablen und κ mit k.
Die Wahrscheinlichkeit, daß m zufällige Mengen alle schlecht für eine Menge K von wesentlichen Variablen sind, ist also höchstens $(\frac{9}{10})^m$.
Sei $R_1, \ldots, R_m$ eine Auswahl von m zufälligen Mengen geeigneter Größe. Wie in Lemma B.2 gezeigt wird, gibt es für $k \leq \frac{n}{3}$ höchstens $2\binom{n}{k}$ verschiedene Möglichkeiten, die Menge S zu wählen. Also ist die Wahrscheinlichkeit, daß mindestens eine Menge S existiert, für die alle Mengen R_i schlecht sind, höchstens $2\binom{n}{k} \cdot (\frac{9}{10})^m$. Es gilt für beliebiges A und B die Formel

$$A^{\log B} = B^{\log A}$$

Dies wird in Lemma B.12 gezeigt. Außerdem gilt $2(\frac{9}{10})^7 < 1$ und $1 + 7 \log \frac{9}{10} < 0$. Dies wird in den folgenden Abschätzungen verwendet.

$$\begin{aligned}
2\binom{n}{k} \cdot \left(\frac{9}{10}\right)^m &= 2\binom{n}{k} \cdot \left(\frac{9}{10}\right)^{\lceil 7 \log \binom{n}{k} \rceil + 7} \\
&\leq 2\binom{n}{k} \cdot \left(\frac{9}{10}\right)^{7 \log \binom{n}{k} + 7} \\
&= 2\binom{n}{k} \left(\frac{9}{10}\right)^7 \cdot \left(\frac{9}{10}\right)^{\log\left(\binom{n}{k}^7\right)} \\
&= 2\binom{n}{k} \left(\frac{9}{10}\right)^7 \cdot \left(\binom{n}{k}^7\right)^{\log \frac{9}{10}} \\
&= 2\left(\frac{9}{10}\right)^7 \cdot \binom{n}{k}^{1 + 7 \log \frac{9}{10}} \\
&< 1
\end{aligned}$$

Also existiert ein System von Mengen $T_1, \ldots, T_m \subseteq [n]$ der Größe $h(n)^2$, so daß für jede Menge $S \subseteq [n]$ mit $\frac{n}{h(n)^2} \leq |S| \leq \frac{n}{h(n)}$ mindestens ein $1 \leq i \leq m$ existiert, so daß $|S \cap T_i| \not\equiv 0 \bmod q$ gilt.
Ähnlich wie in Algorithmus 4.8 kann das Mengensystem $T_1, \ldots, T_m$ verwendet werden, um wesentliche Variablen in $[n]$ zu lernen. Der Algorithmus wird so modifiziert, daß er stoppt, sobald nur noch $\frac{n}{h(n)^2}$ unbekannte wesentliche Variablen verbleiben. Hierfür werden

$$m + O(k \log |T_i|) = O\left(\log \binom{n}{k}\right) + O(k \log h(n)^2)$$

Anfragen gestellt.
Im Folgenden betrachte ich nur die Menge der noch nicht identifizierten Variablen. Hierin liegen $k_1 = \frac{n}{h(n)^2}$ wesentliche Variablen. Analog wie oben kann für $m_1 = \left\lceil 7 \log \binom{n}{k_1} \right\rceil + 7$ ein Mengensystem $T_1^1, \ldots, T_{m_1}^1$ konstruiert werden, mit dessen Hilfe alle bis auf $\frac{n}{h(n)^4}$ wesentliche Variablen mit $O(\log \binom{n}{k_1}) + O(k_1 \log h(n)^2)$ Anfragen gelernt werden können. Dieses Vorgehen wird solange fortgesetzt, bis alle wesentlichen Variablen bis auf eine identifiziert sind,. Die letzte wesentliche Variable wird mit binärer Suche identifiziert. Es gilt

$$k_j = \frac{n}{h(n)^{(2^j)}}$$

Für die Laufzeitanalyse bestimme ich zunächst, wieviele Anfragen die Mengen T_i^j direkt verursachen. Dies sind

$$\begin{aligned}
& O\left(\log \binom{n}{k}\right) + O\left(\log \binom{n}{k_1}\right) + O\left(\log \binom{n}{k_2}\right) + \ldots \\
= \quad & O\left(k \log \frac{n}{k} + k_1 \log \frac{n}{k_1} + k_2 \log \frac{n}{k_2} + \ldots\right) \\
= \quad & O\left(\frac{n}{h(n)} \log \frac{n}{\frac{n}{h(n)}} + \frac{n}{h(n)^2} \log \frac{n}{\frac{n}{h(n)^2}} + \frac{n}{h(n)^4} \log \frac{n}{\frac{n}{h(n)^4}} + \ldots\right) \\
= \quad & O\left(\frac{n}{h(n)} \log h(n)(1 + \frac{2}{h(n)} + \frac{4}{h(n)^3} + \frac{8}{h(n)^7} + \ldots)\right) \\
= \quad & O\left(\frac{n}{h(n)} \log h(n)\right) \\
= \quad & O\left(k \log \frac{n}{k}\right)
\end{aligned}$$

Hierbei wurde verwendet, daß $h(n) > 2$ für n genügend groß gilt und insbesondere die Reihe $1 + \frac{2}{h(n)} + \frac{4}{h(n)^3} + \frac{8}{h(n)^7} + \ldots$ durch eine Konstante nach oben beschränkt ist.

Die Anzahl Anfragen in den binären Suchen ist

$$\begin{aligned}
& O\left(k \log h(n)^2\right) + O\left(k_1 \log h(n)^4\right) + O\left(k_2 \log h(n)^8\right) + \dots \\
= \; & O\left(\frac{n}{h(n)} \log h(n)^2 + \frac{n}{h(n)^2} \log h(n)^4 + \frac{n}{h(n)^4} \log h(n)^8 + \dots\right) \\
= \; & O\left(\frac{n}{h(n)} \log h(n)(2 + \frac{4}{h(n)} + \frac{8}{h(n)^3} + \frac{16}{h(n)^7} + \dots)\right) \\
= \; & O\left(\frac{n}{h(n)} \log h(n)\right) \\
= \; & O\left(k \log \frac{n}{k}\right)
\end{aligned}$$

Insgesamt werden also $O(k \log \frac{n}{k})$ Anfragen gestellt. $\square$

Im folgenden Satz fasse ich die Ergebnisse für die Klasse $MOD_q(k)$ mit $k \leq \frac{n}{3}$ zusammen und zeige, daß hierfür bereits asymptotisch optimale Algorithmen bekannt sind.

Satz 4.12: Algorithmus für $MOD_q(k)$ mit $k \leq \frac{n}{3}$

> *Sei $k \leq \frac{n}{3}$. Dann gibt es einen Algorithmus, der die Klasse $MOD_q(k)$ mit $O(k \log \frac{n}{k})$ Anfragen lernt. Dies ist asymptotisch optimal.*

Beweis: Falls $k = \Omega(n)$ gilt, so ist der lineare Algorithmus aus Satz 4.5 bereits asymptotisch optimal, da die untere Schranke $k \log \frac{n}{k}$ dann ebenfalls linear in n ist.
Sei $k = O(n^{1-\epsilon})$ für ein $\epsilon > 0$. Der Algorithmus 4.8 lernt die Klasse $MOD_q(k)$ mit $O(\log \binom{n}{k} + k \log n)$ Anfragen. In Lemma B.6 wird gezeigt, daß für $k = O(n^{1-\epsilon})$ die Abschätzung $k \log n = O(k \log \frac{n}{k})$ gilt. Also gilt in diesem Fall die Behauptung.
Wenn k nicht von der Form $O(n^{1-\epsilon})$ für irgendein $\epsilon > 0$ ist und $k = \Omega(n)$ ebenfalls nicht zutrifft, kann k in der Form $k(n) = \frac{n}{h(n)}$ für eine Funktion $h : \mathbb{N} \to \mathbb{N}$ geschrieben werden. Die Funktion h strebt dabei gegen Unendlich und ist, für n groß genug, kleiner als cn^ϵ für beliebiges $c > 0$ und $\epsilon > 0$. Daher folgt die Behauptung direkt aus Satz 4.11. $\square$

4.2 Ein Algorithmus für $MOD_q(k)$ mit $k \geq \frac{2n}{3}$ und $q = O(\log \binom{n}{k})$

Im letzten Abschnitt wurde gezeigt, wie für $k \leq \frac{n}{3}$ die Klasse $MOD_q(k)$ für beliebigen Modulus q gelernt werden kann. Hierfür wurde unter anderem verwendet, daß ein nicht zu großes Mengensystem $T_1, \dots, T_m$ existiert, so daß ein Algorithmus stets in einer dieser Mengen mit binärer Suche eine

wesentliche Variable finden kann, solange nicht alle Variablen identifiziert sind.
Ich betrachte nun den Fall, daß $k \geq \frac{2n}{3}$ gilt. Dann ist $k' = n - k$ die Anzahl der unwesentlichen Variablen. Die Idee ist, ähnlich wie im letzten Abschnitt mit binären Suchen auf einem geeigneten Mengensystem alle unwesentlichen Variablen zu identifizieren. Dabei ergeben sich einige Unterschiede zwischen der binären Suche nach wesentlichen bzw. unwesentlichen Variablen, die ich im Folgenden kurz darstellen werde:

- Eine Anfrage A liefert genau dann eine Eins, wenn die Anzahl der wesentlichen Variablen kein Vielfaches von q ist. Um eine ähnliche Aussage über die Anzahl der unwesentlichen Variablen in einer Anfrage B zu erhalten, muß die Größe von B ein Vielfaches von q sein. Dann liefert die Anfrage B genau dann eine Eins, wenn die Anzahl der unwesentlichen Variablen in B kein Vielfaches von q ist.

- Wenn eine Menge A weniger als q Elemente enthält und eine Eins liefert, dann ist darunter mindestens eine wesentliche Variable. Daraus läßt sich keine analoge Aussage über die Anzahl der unwesentlichen Variablen in A ableiten. Um festzustellen, ob A mindestens eine unwesentliche Variable enthält, kann die Menge mit wesentlichen Variablen zu einer q-elementigen Anfrage aufgefüllt werden. Diese liefert genau dann eine Eins, wenn A mindestens eine unwesentliche Variable enthält. Für eine binäre Suche auf kleinen Mengen ist es also scheinbar notwendig, ausreichend viele wesentliche Variablen zu kennen.

- Wenn in einer Menge A eine wesentliche Variable x identifiziert wurde, so kann x aus A entfernt werden. Anschließend kann direkt an der Antwort auf die Anfrage $A \setminus \{x\}$ festgestellt werden, ob in A weitere wesentliche Variablen liegen.

  Wenn y hingegen eine unwesentliche Variable aus A ist, so stimmen die Antworten auf die Anfragen A und $A \setminus \{y\}$ immer überein. Daher ist es sinnvoll, die Variable y durch eine wesentliche Variable z zu ersetzen. Dann ändert sich die Größe von A nicht, aber gegebenenfalls die Antwort auf die neue Anfrage A. Hieran kann man erkennen, ob in A weitere unwesentliche Variablen liegen.

Im folgenden Satz wird gezeigt, daß ein Algorithmus existiert, der für $k' \leq \frac{n}{3}$ die Klasse $MOD_q(k)$ mit $O(q + k' \log n + \log \binom{n}{k'})$ Anfragen lernt. Der Beweis verläuft weitgehend analog zum Beweis von Satz 4.7. Allerdings werden hier zunächst hinreichend viele wesentliche Variablen mit linearer Suche bestimmt, um später binäre Suchen nach unwesentlichen Variablen durchführen zu können.

Satz 4.13: Algorithmus für $MOD_q(k)$ für großes k

> *Sei $k = n - k'$ mit $k' \leq \frac{n}{3}$. Dann kann die Klasse $MOD_q(k)$ mit $O(q + k' \log n + \log \binom{n}{k'})$ Anfragen gelernt werden.*

Beweis: Mit einelementigen Anfragen können $k' + q$ wesentliche Variablen $y_1, \ldots, y_{k'}$ und $z_1, \ldots, z_q$ bestimmt werden. Dies benötigt höchstens $2k' + q$ Anfragen, da es nur k' unwesentliche Variablen gibt.
Die y_i werden später verwendet, um in einer Menge, in der eine unwesentliche Variable x gefunden wurde, diese durch eine wesentliche zu ersetzen. Dadurch ändert sich die Größe der Menge nicht. Mit den z_i werden Dummymengen $D_1, \ldots, D_{q-1}$ gebildet mit $D_l := \{z_1, \ldots, z_{q-l}\}$, mit denen bei binären Suchen auf Mengen mit weniger als q Variablen diese zu q-elementigen Mengen aufgefüllt werden. Eine Menge D_l enthält genau $q - l$ wesentliche Variablen.
Ich gehe im Folgenden davon aus, daß die wesentlichen Variablen $y_1, \ldots, y_{k'}$ und $z_1, \ldots, z_q$ bekannt sind und aus der Menge der Variablen, die weiter betrachtet werden, bereits entfernt wurden und die Größe von n angepaßt wurde.

Ich gebe zunächst einen einfachen Algorithmus an, der auf Mengen $A \subseteq [n]$ arbeitet, die als Anfrage eine Antwort Eins erzeugen und entweder weniger als q oder ein Vielfaches von q Variablen enthalten. Mit binärer Suche kann in solchen Mengen mit $O(\log |A|)$ Anfragen eine unwesentliche Variable identifiziert werden.
Der Algorithmus arbeitet nur auf Mengen, in denen mindestens eine unwesentliche Variable liegt. Sobald eine solche Menge einelementig ist, ist die entsprechende Variable also unwesentlich.
Falls $|A| = cq$ mit $c \geq 2$ gilt, wird A in zwei etwa gleichgroße Mengen B und C aufgeteilt, die $\lfloor \frac{c}{2} \rfloor q$ bzw. $\lceil \frac{c}{2} \rceil q$ Elemente enthalten. Wenn die Menge B als Anfrage eine Eins liefert, sucht der Algorithmus in B weiter, andernfalls in C. Sobald A höchstens q Variablen enthält, wird A ebenfalls in entsprechende Mengen B und C aufgeteilt. Nun wird jedoch die Anfrage $B \cup D_{|B|}$ gestellt, die genau dann eine Eins liefert, wenn in B mindestens eine unwesentliche Variable liegt. Abhängig von der Antwort auf diese Anfrage wird in B oder C weitergesucht.

Algorithmus 4.14:

1. Falls $|A| = 1$ gilt, so enthält A genau eine unwesentliche Variable. STOP.

2. Sei $|A| = cq + r$ mit $c \in \mathbb{N}_0$ und $0 \leq r < q$.

   Falls $c \geq 2, r = 0$:

   - Sei $A = B \,\dot\cup\, C$ mit $|B| = \lfloor \frac{c}{2} \rfloor q$ und $|C| = \lceil \frac{c}{2} \rceil q$

- Frage B
- Falls die Antwort Eins ist, wird in B rekursiv weitergesucht, andernfalls in C

Falls $c \leq 1$:

- Sei $A = B \dot{\cup} C$ mit $|B| = \left\lfloor \frac{|A|}{2} \right\rfloor$ und $|C| = \left\lceil \frac{|A|}{2} \right\rceil$
- Frage $B \cup D_{|B|}$
- Falls die Antwort Eins ist, wird in B rekursiv weitergesucht, andernfalls in C

Ich werde diese binäre Suche in einem Algorithmus verwenden, der sukzessive alle unwesentlichen Variablen lernt. Dazu wird ein System von m Mengen $T_1, \ldots, T_m$ mit $m = \lceil \log \binom{n}{k'} \rceil + 2$ verwendet, so daß für jede Menge $K' \subseteq [n]$ mit $1 \leq |K'| \leq k'$ mindestens ein Index i existiert mit $|K' \cap T_i| \not\equiv 0 \bmod q$. Die Existenz eines solchen Systems wird in Satz 4.6 für $k' \leq \frac{n}{3}$ gezeigt.
Der Algorithmus ist im Wesentlichen analog zu Algorithmus 4.8 aus dem Beweis von Satz 4.7.

Algorithmus 4.15:

1. $maxi = 1$

   $G = \emptyset$

   $\forall i = 1 \ldots m : M_i = T_i$

2. Falls ein $j < maxi$ existiert mit $|M_j \cap G| \not\equiv 0 \bmod q$:

   Setze $l = q - (|M_j \cap G| \bmod q)$ und wiederhole die folgenden vier Punkte, bis $l = 0$ gilt

   - Setze $A = M_j \setminus G \cup \{y_1, \ldots, y_{q-l}\}$
   - Bestimme mit binärer Suche eine unwesentliche Variable $x \in A$
   - $G = G \cup \{x\}$
   - $l = l - 1$

   $M_j = M_j \setminus G$

   weiter bei Schritt 2

3. $M_{maxi} = M_{maxi} \setminus G$

4. Sei $|M_{maxi}| = cq + r$ für $c \in \mathbb{N}_0$ und $0 \leq r < q$

   Falls $r > 0$ gilt:

   - Sei $B \subseteq M_{maxi}$ beliebig mit $|B| = r$

- Frage $B \cup D_{|B|}$
  Falls die Antwort Null ist:
  - $M_{maxi} = M_{maxi} \setminus B$
  - weiter bei Schritt 4

  Falls die Antwort Eins ist:
  - Identifiziere eine unwesentliche Variable $x \in B$ mit binärer Suche
  - $G = G \cup \{x\}$
  - weiter bei Schritt 2

Falls $r = 0$ gilt:

- Frage M_{maxi}
  Falls die Antwort Null ist:
  - $maxi = maxi + 1$
  - weiter bei Schritt 3

  Falls die Antwort Eins ist:
  - Bestimme mit binärer Suche eine unwesentliche Variable $x \in M_{maxi}$
  - $G = G \cup \{x\}$
  - weiter bei Schritt 2

Korrektheit:
Ich verzichte an dieser Stelle auf einen vollständigen Korrektheitsbeweis, da dieser in weiten Teilen analog zum Beweis von Satz 4.7 ist, und beschränke mich auf die Unterschiede, die sich aus der Suche nach unwesentlichen Variablen ergeben.

- Für alle $i < maxi$ gilt stets $|M_i| \equiv 0 \bmod q$.

  Beweis: Am Anfang gilt $maxi = 1$, so daß die Aussage trivial ist. Der Index $maxi$ wird nur in Schritt 4 erhöht und nur dann, wenn $|M_{maxi}| \equiv 0 \bmod q$ gilt (und die Anfrage M_{maxi} eine Null liefert). Traf die Aussage also vor der Erhöhung von $maxi$ auf alle $i < maxi$ zu, so ist sie auch hinterher weiterhin korrekt.

  Eine Menge M_j mit $j < maxi$ wird nur in Schritt 2 verändert. Hier werden aus M_j alle unwesentlichen Variablen, die in G liegen, entfernt. Dies geschieht genau dann, wenn $l = q - (|M_j \cap G| \bmod q) = 0$ gilt. Dann ist die Anzahl der Variablen, die aus M_j entfernt werden, genau ein Vielfaches von q. Da die Größe von M_j vor dem Entfernen ein Vielfaches von q ist, trifft dies auch danach noch zu.

- Jedesmal, wenn eine binäre Suche durchgeführt wird, wird eine unwesentliche Variable gefunden.

  Beweis: Wenn in Schritt 2 die binäre Suche für die Menge A aufgerufen wird, ist die Größe von A ein Vielfaches von q, da in A genau die $q-l$ unwesentlichen Variablen aus M_j, die in G liegen, durch wesentliche ersetzt wurden und die Größe von M_j ein Vielfaches von q ist. Solange $l \neq 0$ gilt, ist die Anzahl der unwesentlichen Variablen in A kein Vielfaches von q, so daß die Voraussetzungen für die binäre Suche in Algorithmus 4.14 erfüllt sind.

  In Schritt 4 kann eine binäre Suche für die r-elementige Menge $B \subseteq M_{maxi}$ aufgerufen werden. Dies geschieht genau dann, wenn die Anfrage B, aufgefüllt auf q Variablen, eine Eins liefert. Da $r < q$ gilt, kann die binäre Suche in Algorithmus 4.14 eine unwesentliche Variable identifizieren.

  Wird in Schritt 4 die binäre Suche für die Menge M_{maxi} aufgerufen, so gilt $|M_{maxi}| \equiv 0 \bmod q$ und die Anfrage M_{maxi} liefert eine Eins. Also kann eine unwesentliche Variable gefunden werden.

- Jede unwesentliche Variable wird höchstens einmal mit einer binären Suche gefunden.

  Beweis: Sobald eine unwesentliche Variable mit einer binären Suche gefunden wurde, wird sie in die Menge G eingefügt. Vor dem Aufruf einer binären Suche für eine beliebige Menge X wird jedesmal die gesamte Menge G der bisher gefundenen unwesentlichen Variablen aus X entfernt.

Laufzeit:
Anfragen werden nur in Schritt 2 und Schritt 4 gestellt. In Schritt 2 wird mit einer binären Suche mit $O(\log|A|)$ Anfragen eine unwesentliche Variable identifiziert.
Wenn in Schritt 4 die Größe von M_{maxi} ein Vielfaches von q ist, wird die Anfrage M_{maxi} gestellt. Für jedes $maxi$ kann diese Anfrage höchstens einmal eine Null liefern, da danach der Index $maxi$ erhöht (und später nie wieder vermindert) wird. Die geschieht im gesamten Algorithmus höchstens m-mal. Falls die Antwort eine Eins ist, so wird mit einer binären Suche mit $O(\log|M_{maxi}|)$ Anfragen eine unwesentliche Variable identifiziert.
Wenn in Schritt 4 die Anfrage $B \cup D_{|B|}$ gestellt wird und eine Eins liefert, so wird danach mit binärer Suche mit $O(\log|B|)$ Anfragen eine unwesentliche Variable in B identifiziert. Falls die Anfrage eine Null liefert, so wird B aus M_{maxi} entfernt. Anschließend ist die Größe von M_{maxi} ein Vielfaches von q und entweder wird $maxi$ erhöht oder mit $O(\log|M_{maxi}|)$ Anfragen eine unwesentliche Variable bestimmt.

Da es genau k' unwesentliche Variablen gibt, werden in dem Algorithmus höchstens k' binäre Suchen durchgeführt. Für jede dieser Suchen werden $O(\log n)$ Anfragen gestellt. Außerdem werden für jeden Wert von $maxi$ höchstens $O(1)$ weitere Anfragen gestellt. Da $maxi$ ständig erhöht wird und zwischen 1 und m liegt, sind dies $O(m)$ Anfragen.
Vor Beginn des Algorithmus werden $k'+q$ wesentliche Variablen mit linearer Suche und $O(k'+q)$ Anfragen bestimmt.
Insgesamt stellt der Algorithmus also $O(k'+q+m+k'\log n)$ Anfragen. Mit $m = O(\log \binom{n}{k})$ folgt die Behauptung.

□

Die folgenden zwei Sätze sind einfache Übertragungen der Sätze 4.11 und 4.12 auf den Fall der unwesentlichen Variablen. Die Anzahl der wesentlichen bzw. unwesentlichen Variablen hängt von n ab. Ich verwende die Schreibweise k und k' statt $k(n)$ bzw. $k'(n)$.

Satz 4.16: Algorithmus für $MOD_q(k)$ für großes k

> *Sei $h : \mathbb{N} \to \mathbb{N}$ eine Funktion, die gegen Unendlich strebt. Für jedes $\epsilon > 0$ und jedes $c > 0$ gelte $h(n) < cn^\epsilon$ für n groß genug.*
>
> *Sei $k = n - k'$, $k' = \frac{n}{h(n)}$ und $3 \le k' \le \frac{n}{3}$.*
>
> *Dann kann die Klasse $MOD_q(k)$ mit $O(q + k'\log \frac{n}{k'})$ Anfragen gelernt werden.*

Beweis: Der Beweis verläuft vollkommen analog zum Beweis von Satz 4.11. Der einzige Unterschied ergibt sich in der Laufzeit, da vor Beginn des eigentlichen Algorithmus $O(q+k')$ Anfragen gestellt werden, um mit linearer Suche $q+k'$ wesentliche Variablen zu bestimmen. □

Satz 4.17: Algorithmus für $MOD_q(k)$ mit $k' \le \frac{n}{3}$, $q = O(\log \binom{n}{k})$

> *Sei $k = n - k'$, $k' \le \frac{n}{3}$ und $q = O(\log \binom{n}{k})$. Dann existiert ein Algorithmus, der die Klasse $MOD_q(k)$ mit $O(\log \binom{n}{k})$ Anfragen lernt. Dies ist asymptotisch optimal.*

Beweis: Wie im Beweis von Satz 4.12 läßt sich zeigen, daß ein Algorithmus existiert, der die Klasse $MOD_q(k)$ mit $O(q + k'\log \frac{n}{k'})$ Anfragen lernt. Aus $q = O(\log \binom{n}{k})$ folgt die Behauptung. □

Der Term $O(q)$ in der obigen Laufzeit entsteht dadurch, daß die q wesentlichen Variablen, die zur Konstruktion der Dummymengen notwendig sind, mit linearer Suche bestimmt werden. Ob sich solche Dummymengen mit weniger Anfragen bestimmen lassen, ist mir nicht bekannt.

4.3 Untere Schranken für großes q

In Satz 4.17 wird gezeigt, daß für kleines q und $k \geq \frac{2n}{3}$ ein asymptotisch optimaler Algorithmus existiert. In diesem Abschnitt zeige ich zunächst, daß für $q > \frac{2n}{3}$ und $k = q + c$ mindestens $\left\lceil \frac{k}{c+1} \right\rceil$ Anfragen notwendig sind, um $MOD_q(k)$ zu lernen. Wenn sich k und q nur um eine additive Konstante unterscheiden, ist also ebenfalls ein asymptotisch optimaler Algorithmus bekannt (vgl. Bemerkung zu Satz 4.5).
Anschließend untersuche ich, wieviele Anfragen ein Algorithmus, der nur Anfragen mit höchstens q Variablen verwendet, mindestens stellen muß.

Satz 4.18: Untere Schranke für $MOD_q(k)$ mit $k = q + c$

> *Sei $k = q + c$ und $q > \frac{2n}{3}$. Dann sind mindestens $\left\lceil \frac{k}{c+1} \right\rceil$ Anfragen notwendig, um $MOD_q(k)$ zu lernen.*

Beweis: Ich gebe eine Strategie für einen Gegenspieler an, die jeden Algorithmus zwingt, mindestens $\left\lceil \frac{k}{c+1} \right\rceil$ Anfragen zu stellen. Der Gegenspieler antwortet auf jede Anfrage und benennt zusätzlich geeignete Variablen als wesentlich, so daß die entsprechende Antwort nie inkonsistent werden kann.
Ich nehme an, daß ein Algorithmus keine Anfragen stellt, für die die Antwort bereits bekannt ist. Insbesondere werden keine leeren oder n-elementigen Anfragen gestellt.
Da $q > \frac{2n}{3}$ gilt, kann eine Anfrage genau dann eine Null erzeugen, wenn sie keine oder genau q wesentliche Variablen enthält.
Falls eine Anfrage A weniger als q Variablen enthält, antwortet der Gegenspieler eine Eins und benennt eine Variable in A als wesentlich. Hierdurch kann die Antwort nie mehr inkonsistent werden.
Eine Anfrage mit genau q Variablen muß mindestens eine wesentliche Variable enthalten, da es wegen $k \geq q > \frac{2n}{3}$ weniger als q unwesentliche Variablen gibt. Der Gegenspieler antwortet eine Eins und benennt genau $c + 1$ Variablen aus dem Komplement der Anfrage als wesentlich. Dann enthält die Anfrage höchstens $q - 1$ wesentliche Variablen und die Antwort kann nie mehr inkonsistent werden.
Wenn der Algorithmus eine Anfrage A mit $q + s$ Variablen mit $1 \leq s \leq n - k - 1$ stellt, so antwortet der Gegenspieler eine Eins und identifiziert $c + 1$ Variablen im Komplement von A als wesentlich. Dies ist möglich, da das Komplement von A mindestens $n - (q + s) \geq n - (q + n - k - 1) = k - q + 1 = c + 1$ Variablen enthält. Die Antwort Eins ist somit immer konsistent.
Auf Anfragen A mit $n - c$ Variablen antwortet der Gegenspieler eine Null und identifiziert im Komplement von A genau c Variablen als wesentlich.
Anfragen mit mehr als $n - c$ Variablen werden nicht gestellt, da für solche Anfragen weniger als c wesentliche Variablen im Komplement liegen und

somit stets mehr als q wesentliche Variablen in der Anfrage liegen müssen. Daher wäre die Antwort Eins bereits vorher bekannt.
Der Gegenspieler identifiziert für jede Anfrage höchstens $c+1$ wesentliche Variablen und liefert keine Informationen über die anderen Variablen. Also muß der Algorithmus mindestens $\left\lceil \frac{k}{c+1} \right\rceil$ Anfragen stellen, um alle wesentlichen Variablen zu lernen (vgl. Beweis von Satz 5.8).

□

Ich zeige nun, daß ein Algorithmus, der nur Anfragen mit höchstens q Variablen verwendet, mindestens $\lfloor \frac{n}{2} \rfloor$ Anfragen stellen muß, um die Klasse $MOD_q(n-k')$ zu lernen, wenn die Anzahl der unwesentlichen Variablen k' größer als $2\log n$ ist und $q > \frac{2n}{3}$ gilt. Eine Verallgemeinerung auf beliebige Anfragegröße ist mir nicht gelungen.
Für eine beliebige Folge von Anfragen antwortet ein Gegenspieler auf die ersten $\lfloor \frac{n}{2} \rfloor$ Anfragen stets eine Eins. Ich gebe ein Konstruktionsverfahren für eine Variablenbelegung an, die mit diesen Antworten konsistent ist. In jeder Anfrage mit weniger als q Variablen wird sofort eine wesentliche Variable identifiziert. Nachdem $\lfloor \frac{n}{2} \rfloor$ Anfragen gestellt wurden, werden geeignete unwesentliche Variablen identifiziert, so daß in jeder q-elementigen Anfrage mindestens eine dieser unwesentlichen Variablen liegt. Hierdurch werden alle bisher gegebenen Antworten konsistent.

Satz 4.19: Untere Schranke für $MOD_q(k)$ mit Anfragen $|A| \leq q$

> *Sei $k = n - k'$, $k' > 2\log n$ und $q > \frac{2n}{3}$. Ein Algorithmus, der nur Anfragen mit höchstens q Variablen verwendet, benötigt mindestens $\lfloor \frac{n}{2} \rfloor$ Anfragen, um $MOD_q(k)$ zu lernen.*

Beweis: Ich gebe eine Strategie für einen Gegenspieler an, die jeden Algorithmus zwingt, mindestens $\lfloor \frac{n}{2} \rfloor$ Anfragen zu stellen. Dabei nehme ich an, daß ein Algorithmus keine Anfragen stellt, für welche die Antwort bereits bekannt ist, also z. B. keine leeren Anfragen oder Mengen, die nur aus bereits identifizierten wesentlichen Variablen bestehen. Bereits als unwesentlich identifizierte Variablen kommen in späteren Anfragen nicht mehr vor. Ich nehme an, daß $\frac{n}{2}$ ganzzahlig ist. Falls n ungerade ist, verläuft der Beweis analog.
Der Gegenspieler antwortet auf die ersten $\frac{n}{2}$ Anfragen mit einer Eins. In Anfragen mit weniger als q Variablen identifiziert er sofort eine Variable als wesentlich. Nach insgesamt $\frac{n}{2}$ Anfragen legt der Gegenspieler eine Auswahl K von wesentlichen Variablen fest, die mit allen bisher gegebenen Antworten konsistent ist.
Ich nehme an, daß ein Algorithmus r Anfragen mit weniger als q Variablen und t Anfragen mit genau q Variablen stellt mit $r + t = \frac{n}{2}$. Diese Anfragen können in beliebiger Reihenfolge auftreten. Wenn eine Anfrage A weniger als q Variablen enthält, so kann sie nur dann eine Null erzeugen, wenn keine

wesentliche Variable in A liegt. Da der Gegenspieler in jeder solchen Anfrage *sofort* eine wesentliche Variable identifiziert und diese in K einfügt, kann keine dieser Anfragen inkonsistent werden. Mit diesen Anfragen werden $r \leq \frac{n}{2} < k$ wesentliche Variablen identifiziert. Diese werden während der ersten $\frac{n}{2}$ Anfragen festgelegt und sind somit bekannt, wenn der Gegenspieler anschließend eine Auswahl von wesentlichen Variablen bestimmt, die mit allen Antworten konsistent ist.
Da $k \geq q > \frac{2n}{3}$ gilt, kann eine Anfrage A mit genau q Variablen nicht ausschließlich aus unwesentlichen Variablen bestehen. Sie erzeugt genau dann eine Null, wenn sie nur wesentliche Variablen enthält. Damit für alle q-elementigen Anfragen die Antwort Eins, die der Gegenspieler gegeben hat, konsistent ist, muß in jeder solchen Anfrage mindestens eine unwesentliche Variable liegen.
Seien A_j für $1 \leq j \leq t$ die q-elementigen Anfragen. Ich nehme zunächst an, daß jedes A_j alle r bereits bekannten wesentlichen Variablen enthält. Wenn eine Anfrage A_j nicht alle r bereits bekannten wesentlichen Variablen enthält, werden die folgenden Abschätzungen noch besser. Dies wird später untersucht. Ich zeige, daß

$$\log_{\frac{n-r}{n-q}} t + 1$$

unwesentliche Variablen ausreichen, damit in jedem A_j mindestens eine hiervon liegt. Eine Variable x soll *frei* heißen, wenn sie noch nicht als wesentlich oder unwesentlich festgelegt ist. Eine Menge A_j heißt *offen*, solange noch keine unwesentliche Variable in A_j bekannt ist, sonst *geschlossen*.
Ich gebe ein Verfahren an, das sukzessiv solche freien Variablen auswählt und als unwesentlich festlegt, die viele Mengen schließen. Sei t_i die Anzahl von offenen Mengen nach Auswahl von i unwesentlichen Variablen. Dann gilt $t_0 = t$.
Durch Induktion über i zeige ich folgende Aussage:

(Ψ) Nach i Festlegungen von unwesentlichen Variablen sind noch $n - r - i$ freie Variablen bekannt und es verbleiben $t_i \leq t(\frac{n-q}{n-r})^i$ offene Mengen.

IA: $i = 0$. Zu Beginn des Verfahrens sind $n - r$ freie Variablen bekannt und es sind $t_0 = t$ offene Mengen vorhanden.
IS: $i \to i+1$. Nach dem i-ten Schritt gibt es noch $n-r-i$ freie Variablen und $t_i \leq t(\frac{n-q}{n-r})^i$ offene Mengen. Jede dieser Mengen A enthält genau q Variablen. Hiervon sind nach Voraussetzung r Variablen wesentlich. Die anderen $q - r$ Variablen sind frei, da A sonst bereits geschlossen wäre. Alle offenen Mengen zusammen enthalten also $t_i(q - r)$ freie Variablen (mehrfach vorkommende Variablen werden auch mehrfach gezählt). Da es genau $n-r-i$ verschiedene freie Variablen gibt, muß es eine freie Variable x_i geben, die in mindestens $\frac{t_i(q-r)}{n-r-i}$ offenen Mengen liegt. Dieses x_i wird eine unwesentliche Variable. Hierdurch werden mindestens

$$\frac{t_i(q-r)}{n-r-i} \geq \frac{t_i(q-r)}{n-r}$$

Mengen geschlossen und es bleiben

$$t_{i+1} \leq t_i - \frac{t_i(q-r)}{n-r} = t_i \frac{n-q}{n-r}$$

offene Mengen. Mit der Induktionsvoraussetzung folgt $t_{i+1} \leq t(\frac{n-q}{n-r})^{i+1}$ und es bleiben noch $n-r-(i+1)$ freie Variablen.

Nach $i_0 = \log_{\frac{n-r}{n-q}} t$ Schritten verbleiben noch höchstens

$$t_{i_0} \leq t\left(\frac{n-q}{n-r}\right)^{i_0} = t\frac{1}{(\frac{n-r}{n-q})^{i_0}} = 1$$

offene Mengen. Die letzte offene Menge kann ggf. mit einer weiteren freien Variablen geschlossen werden.

Ich zeige nun, daß sich t_i höchstens verkleinert, wenn die offenen Mengen auch weniger als r der bereits bekannten wesentlichen Variablen enthalten können. Der Beweis von Ψ verläuft weitgehend analog mit dem Unterschied, daß in einer Menge A_j, die weniger als r bekannte wesentliche Variablen enthält, mehr als $q-r$ freie Variablen liegen. Also ist die Summe aller freien Variablen in offenen Mengen größer als $t_i(q-r)$. Hierdurch werden die nachfolgenden Ungleichungen höchstens günstiger und t_i gegebenenfalls sogar kleiner.

Ich habe bisher gezeigt, daß $\log_{\frac{n-r}{n-q}} t+1$ unwesentliche Variablen ausreichen, damit alle Antworten des Gegenspielers konsistent sind. Es bleibt zu zeigen, daß

$$\log_{\frac{n-r}{n-q}} t+1 < k'$$

gilt. Da $t+r=\frac{n}{2}$ gilt, folgt $t \leq \frac{n}{2}$ und $r \leq \frac{n}{2}$. Mit $q > \frac{2n}{3}$ gilt

$$\begin{aligned}
\log_{\frac{n-r}{n-q}} t + 1 &= \frac{\log t}{\log(n-r) - \log(n-q)} + 1 \\
&< \frac{\log t}{\log \frac{n}{2} - \log \frac{n}{3}} + 1 \\
&= \frac{\log t}{\log \frac{3}{2}} + 1 \\
&\leq \frac{\log n - 1}{\log 3 - 1} + 1 \\
&\leq \frac{\log n - 1}{\frac{1}{2}} + 1 \\
&= 2 \log n - 1 \\
&\leq k'
\end{aligned}$$

Nach den Festlegungen, die der Gegenspieler bisher getroffen hat, sind alle bisherigen Antworten konsistent. Insgesamt werden $r \leq \frac{n}{2} < k$ wesentliche und $\log_{\frac{n-r}{n-q}} t + 1 < k'$ unwesentliche Variablen festgesetzt. Anschließend existieren also noch mindestens eine wesentliche und mindestens eine unwesentliche Variable und der Algorithmus muß noch mindestens eine weitere Anfrage stellen, um alle Variablen zu lernen. □

Falls Anfragen mit mehr als q Variablen zulässig sind, ist mir nur die allgemeine untere Schranke $\lceil \log \binom{n}{k} \rceil$ bekannt. Das Problem, eine bessere untere Schranke zu finden, ist ähnlich schwierig wie bei Anzahl-Funktionen, da für $q > \frac{n}{2}$ der Wertevektor nur 2 Nullen an der ersten und $(q+1)$-ten Position enthält. Für große Anfragen ist die Null am Anfang jedoch irrelevant, so daß im Wesentlichen eine Anzahl-Funktion vorliegt.

Kapitel 5

Anzahl-Funktionen

Eine Anzahl-Funktion der Ordnung ρ gibt genau dann eine Eins aus, wenn genau ρ Einsen in der Eingabe sind. Der Wertevektor einer Anzahl-Funktion hat die Gestalt $v = (0, \ldots, 0, 1, 0, \ldots, 0)$, wobei die Eins genau in der $(\rho+1)$-ten Koordinate steht. Jede symmetrische Funktion läßt sich als Disjunktion von Anzahl-Funktionen schreiben, d. h. Anzahl-Funktionen bilden in gewisser Weise eine „Basis“ der symmetrischen Funktionen.
In diesem Kapitel untersuche ich Klassen von Anzahl-Funktionen mit k wesentlichen von n Variablen. Hierfür gebe ich einzelne Resultate an, z. B. zwei untere Schranken sowie für einige sehr spezielle Werte von k und ρ effiziente Algorithmen. Außerdem zeige ich, daß man Anzahl-Funktionen genau dann schnell lernen kann, wenn man für das Lottospiel Tippsysteme mit wenigen Tippreihen kennt, so daß man garantiert ρ Richtige hat. Für den Fall $\rho = 1$ gebe ich einen Algorithmus an, der Anzahl-Funktionen mit Hilfe von Pflasterungen lernt.

Definition 5.1: Anzahl-Funktion

> *Sei $n \in \mathbb{N}$ und $\rho \in \mathbb{N}_0$. Eine Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt* **Anzahl-Funktion ρ-ter Ordnung**, *wenn $f(x) = 1$ genau dann gilt, wenn $|x|_1 = \rho$, d. h. x enthält genau ρ Einsen.*
>
> *Man schreibt $f(x) = anz_\rho(x)$.*

Definition 5.2: $ANZ_\rho^n(k)$

> *Seien $n \in \mathbb{N}$ und $k, \rho \in \mathbb{N}_0$ mit $0 \leq k \leq n$.*
>
> *Dann besteht die Klasse $ANZ_\rho^n(k)$ aus allen Anzahl-Funktionen ρ-ter Ordnung auf n Variablen mit k wesentlichen Variablen. Unwesentliche Variablen werden durch Null ersetzt.*
>
> $$ANZ_\rho^{\mathbf{n}}(\mathbf{k}) := \{anz_{\rho,K} \mid K \subseteq [n], |K| = k\}$$

Ich schreibe auch $ANZ_\rho(k)$, wenn n aus dem Zusammenhang eindeutig ist.

Für $k = 0$ oder $k = n$ enthält die Klasse $ANZ_\rho(k)$ genau eine Funktion.
Für $\rho > k$ enthält $ANZ_\rho(k)$ genau die konstante Null-Funktion.
Eine Anfrage in $ANZ_0(k)$ liefert genau dann eine Null, wenn mindestens eine wesentliche Variable darin enthalten ist. Also kann diese Klasse mit einem Algorithmus für negierte Disjunktionen mit k wesentlichen Variablen gelernt werden. Für $\rho = k$ erzeugt eine Anfrage genau dann eine Eins, wenn alle wesentlichen Variablen in dieser Anfrage liegen. Dies entspricht genau einer Konjunktion. Für diese beiden Klassen werden in [24] asymptotisch optimale Algorithmen angegeben.
Ich beschränke die folgenden Betrachtungen auf

$$\boxed{n \in \mathbb{N},\, 2 \leq k \leq n-1 \text{ und } 1 \leq \rho \leq k-1}$$

da alle anderen Fälle entweder trivial oder bereits untersucht sind. Diese Voraussetzungen werde ich ggf. verschärfen.

Ich zeige zunächst, daß sich aus Algorithmen für $ANZ_\rho(k)$ direkt Algorithmen für $ANZ_{k-\rho}(k)$ ergeben, die genauso viele Anfragen stellen. Daher kann ich mich in Beweisen stets auf $\rho \leq \frac{k}{2}$ beschränken.

Satz 5.3: Symmetrie von $ANZ_\rho(k)$

> *Aus einem Algorithmus für $ANZ_\rho(k)$ kann ein Algorithmus für $ANZ_{k-\rho}(k)$ konstruiert werden, so daß beide Algorithmen für jede Auswahl von k wesentlichen Variablen jeweils gleich viele Anfragen stellen.*

Beweis: Sei Δ ein Algorithmus für die Klasse $ANZ_\rho(k)$. Hieraus konstruiere ich einen Algorithmus Γ, der die Klasse $ANZ_{k-\rho}(k)$ lernt.
Der Algorithmus Δ kann durch einen binären Entscheidungsbaum T_Δ dargestellt werden (vgl. Seite 19), an dessen Knoten die Anfragen stehen, die der Algorithmus stellt. Die beiden Söhne eines Knotens entsprechen den Antworten Eins oder Null auf eine Anfrage. An den Blättern des Baumes stehen die durch den Algorithmus identifizierten Funktionen (d. h. die Mengen der wesentlichen Variablen).
Der Entscheidungsbaum T_Γ für Algorithmus Γ wird konstruiert, indem im Baum T_Δ jede Anfrage durch ihr Komplement ersetzt wird. Dies ist in Abbildung 5.1 veranschaulicht. An den Blättern stehen dabei die Mengen der wesentlichen Variablen, die mit $K_1, \ldots, K_6$ bezeichnet werden.
Ich zeige, daß der Algorithmus Γ alle Funktionen aus $ANZ_{k-\rho}(k)$ lernt. Sei dazu K eine beliebige Auswahl von k wesentlichen Variablen. Dann gibt es im Entscheidungsbaum von Δ einen Pfad von Anfragen $A_1, \ldots, A_t$, so daß nach der letzten Anfrage die Menge K der wesentlichen Variablen identifiziert wird.

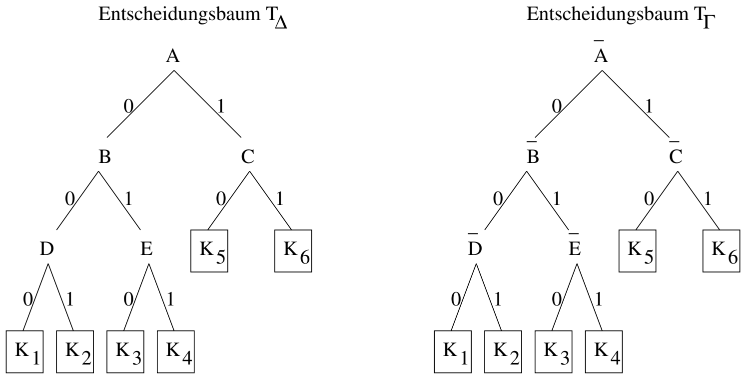

Abbildung 5.1: Konstruktion von T_Γ aus T_Δ

Für eine beliebige Anfrage S, die der Algorithmus Δ stellt, um K zu lernen, gilt

$$\begin{array}{ll} & S \text{ liefert eine Eins in } ANZ_\rho(k) \\ \Longleftrightarrow & \text{In } S \text{ liegen genau } \rho \text{ Variablen aus } K \\ \Longleftrightarrow & \text{In } \bar{S} \text{ liegen genau } k-\rho \text{ Variablen aus } K \\ \Longleftrightarrow & \bar{S} \text{ liefert eine Eins in } ANZ_{k-\rho}(k) \end{array}$$

Also stimmen die Antworten, die der Algorithmus Γ auf seine Anfragen erhält, stets mit denen überein, die der Algorithmus Δ bekommt. Algorithmus Γ stellt die Anfragen $\bar{A}_1, \ldots, \bar{A}_t$ und identifiziert die Menge K der wesentlichen Variablen mit genauso vielen Anfragen wie Δ.
Da auf analoge Weise jeder Algorithmus für $ANZ_{k-\rho}(k)$ in einen Algorithmus für $ANZ_\rho(k)$ transformiert werden kann, folgt die Behauptung. $\square$

Ich untersuche nun, für welche Parameter k und ρ asymptotisch optimale Algorithmen für die Klasse $ANZ_\rho(k)$ bekannt sind. Dabei verwende ich im Vorgriff bereits die unteren Schranken $\lceil \log \binom{n}{k} \rceil$ und $\left\lceil \frac{k}{\rho+1} \right\rceil$, die erst im nächsten Abschnitt bewiesen werden.
In Satz 2.5 wird gezeigt, daß für symmetrische Funktionen ein Algorithmus existiert, dessen Laufzeit linear in n ist. Dieser Algorithmus ist für k nahe bei $\frac{n}{2}$ asymptotisch optimal, da dann die untere Schranke $\lceil \log \binom{n}{k} \rceil$ ebenfalls linear in n ist.
Falls ρ konstant und k linear in n ist, so ist die untere Schranke $\left\lceil \frac{k}{\rho+1} \right\rceil$ aus Satz 5.8 ebenfalls linear in n, so daß auch in diesem Fall ein asymptotisch optimaler Algorithmus bekannt ist.
Für $k = O(1)$ gibt Damaschke in [8] einen asymptotisch optimalen Algorithmus an.

In Kapitel 2 wird im Beweis von Satz 2.7 ein Algorithmus angegeben, der

eine symmetrische Funktion mit $O(\frac{n}{r} + k' \log \frac{n}{k'})$ Anfragen lernt. Dabei ist r die erste Position im Wertevektor der Funktion, an der ein Wechsel zwischen einer Null und einer Eins vorliegt. Für Anzahl-Funktionen gilt also $r = \rho$. Mit k' wird die Anzahl der unwesentlichen Variablen bezeichnet. Dieser Algorithmus setzt voraus, daß $k' < \lfloor \frac{n}{r} \rfloor$ gilt, und stellt zu Beginn $\lfloor \frac{n}{r} \rfloor$ disjunkte r-elementige Mengen als Anfragen. Dann muß mindestens eine dieser Mengen genau r unwesentliche Variablen enthalten.
Dieser Algorithmus kann verallgemeinert werden, indem für eine geeignete Konstante c zu Beginn $k'+c$ disjunkte Mengen der Größe $\lceil \frac{\rho}{c} \rceil$ gefragt werden. Ich gebe diese Verallgemeinerung nur für die Anzahl-Funktionen an. Eine Übertragung auf beliebige symmetrische Funktionen ist leicht.

Satz 5.4: Algorithmus für $ANZ_\rho(n - k')$

> *Sei $k = n - k'$. Sei $c \in [\rho]$ mit $(k' + c) \lceil \frac{\rho}{c} \rceil \leq n$.*
>
> *Dann kann die Klasse $ANZ_\rho(n - k')$ mit $O(\binom{k'+c}{c} + k' \log \rho)$ Anfragen gelernt werden.*

Beweis: Der folgende Algorithmus lernt alle unwesentlichen Variablen.

Algorithmus 5.5:

1. Bilde $k' + c$ disjunkte Mengen der Größe $\lceil \frac{\rho}{c} \rceil$.

2. Jede Kombination von genau c dieser Mengen wird ggf. auf Größe ρ verkleinert und als Anfrage gestellt. Mindestens eine dieser Anfragen liefert eine Eins. Diese enthält genau ρ wesentliche Variablen $w_1, \ldots, w_\rho$.

   Jede Anfrage mit Antwort Null wird eine Gewinnerin.

3. Definiere Dummymengen $D_j := \{w_1, \ldots, w_{\rho-j}\}$ für $1 \leq j \leq \rho - 1$.

4. Bilde die Menge A, die alle Variablen enthält, die im ersten Schritt in keiner der angefragten Mengen lagen. Frage $A \cup D_{|A|}$. Falls diese Anfrage eine Null liefert, wird A eine weitere Gewinnerin.

5. Sei R eine Gewinnerin. Falls R einelementig ist, so ist die in R liegende Variable unwesentlich. Andernfalls wird R in zwei (fast) gleichgroße Mengen U und V aufgeteilt. Frage $U \cup D_{|U|}$ und $V \cup D_{|V|}$. Jede Null identifiziert eine neue Gewinnerin, die weiter untersucht wird.

Korrektheit:
Im ersten Schritt werden $k' + c$ disjunkte Mengen gebildet. Dabei enthält jede der Mengen mindestens eine Variable. Alle Mengen können gebildet werden, da n nach Voraussetzung groß genug ist. Da es nur k' unwesentliche Variablen gibt, müssen mindestens c verschiedene Mengen existieren, die keine unwesentliche Variable enthalten. Im zweiten Schritt werden alle

Kombinationen von c Mengen angefragt. Dann muß mindestens eine dieser Kombinationen eine Eins liefern, da sie genau ρ wesentliche Variablen enthält. Hiermit können im dritten Schritt die Dummymengen D_j definiert werden. Eine Dummymenge D_j enthält genau $\rho - j$ wesentliche Variablen. Für eine beliebige Menge S mit $|S| \leq \rho$ liefert $S \cup D_{|S|}$ genau dann eine Null, wenn S mindestens eine unwesentliche Variable enthält.
Jede Anfrage, die eine Gewinnerin wird, muß mindestens eine unwesentliche Variable enthalten. Im vierten Schritt werden ggf. die Variablen untersucht, die im ersten Schritt noch nicht betrachtet wurden. Nach diesem Schritt muß jede Variable in genau einer der angefragten Mengen gelegen haben. Dies gilt insbesondere für die k' unwesentlichen Variablen. Diese liegen in den Gewinnerinnen.
Im letzten Schritt werden mit den binären Suchen alle unwesentlichen Variablen identifiziert. Dies geschieht genauso wie in Algorithmus 2.8.

Laufzeit:
Im ersten Schritt werden keine Anfragen gestellt. Im zweiten Schritt werden genau $\binom{k'+c}{c}$ Kombinationen gebildet und angefragt. Für die Definition der Dummymengen ist keine Anfrage notwendig. Im vierten Schritt wird ggf. eine Anfrage gestellt. Für die binären Suchen im letzten Schritt werden $O(k' \log \rho)$ Anfragen gestellt (vgl. Beweis von Satz 2.7).
Insgesamt werden also $O(\binom{k'+c}{c} + k' \log \rho)$ Anfragen gestellt. □

Falls $k' = O(1)$ gilt und ρ geeignet klein ist, ist dieser Algorithmus für geeignetes c asymptotisch optimal, da der Term $\binom{k'+c}{c} \leq \left(\frac{e(k'+c)}{c}\right)^c$ für kleines c ebenfalls sehr klein ist.

5.1 Untere Schranken

In diesem Kapitel gebe ich zwei untere Schranken für die Klasse $ANZ_\rho(k)$ an. Die erste ergibt sich direkt aus der allgemeinen unteren Schranke für symmetrische Funktionen. Für die zweite Schranke beschreibe ich eine Strategie für einen Gegenspieler, die jeden Algorithmus durch geeignete Antworten zwingt, mindestens $\left\lceil \frac{k}{\rho+1} \right\rceil$ Anfragen zu stellen. Außerdem beschreibe ich einen weiteren Ansatz für eine untere Schranke, mit dem sich jedoch vermutlich keine Verbesserung der bekannten Schranken erreichen läßt.

Lemma 5.6: Größe von $ANZ_\rho(k)$

Die Klasse $ANZ_\rho(k)$ enthält genau $\binom{n}{k}$ Funktionen.

Beweis: Sei v der Wertevektor einer Funktion aus $ANZ_\rho(k)$. Dann hat v in den ersten ρ Koordinaten $v_0, \ldots, v_{\rho-1}$ eine Null und an der folgenden Position v_ρ eine Eins. Daher folgt die Behauptung direkt aus Satz 2.3. □

Satz 5.7: Erste untere Schranke für $ANZ_\rho(k)$

> *Es sind mindestens* $\lceil \log \binom{n}{k} \rceil$ *Anfragen notwendig, um* $ANZ_\rho(k)$ *zu lernen. Es gilt* $\lceil \log \binom{n}{k} \rceil \geq k \log \frac{n}{k}$.

Beweis: Es gilt $|ANZ_\rho(k)| = \binom{n}{k}$. Die erste Behauptung folgt direkt aus der unteren Schranke für allgemeine Funktionenklassen in Satz 1.2. Die Abschätzung folgt aus Lemma B.1. □

Satz 5.8: Zweite untere Schranke für $ANZ_\rho(k)$

> *Es sind mindestens* $\left\lceil \frac{k}{\rho+1} \right\rceil$ *Anfragen notwendig, um die Klasse* $ANZ_\rho(k)$ *zu lernen.*

Beweis: Ich nehme an, daß ein Algorithmus nur Anfragen mit mindestens ρ und höchstens $\rho + n - k$ Variablen stellt, da sonst die Antwort Null bereits vorher bekannt ist. Aus demselben Grund soll jede Anfrage höchstens ρ bereits als wesentlich identifizierte Variablen enthalten. Ich zeige, daß ein Gegenspieler jeden Algorithmus zwingen kann, mindestens $\left\lceil \frac{k}{\rho+1} \right\rceil$ Anfragen zu stellen, und gebe eine entsprechende Strategie an.
Für die ersten $\left\lceil \frac{k}{\rho+1} \right\rceil - 1$ Anfragen eines Algorithmus gibt der Gegenspieler eine Eins aus, falls die Anfrage genau ρ Variablen enthält, andernfalls eine Null. Außerdem identifiziert er in jeder Anfrage genau ρ bzw. $\rho + 1$ wesentliche Variablen. Dabei werden zunächst die Variablen als wesentlich benannt, die bereits in einem früheren Schritt identifiziert wurden. Der Algorithmus erhält über die weiteren in der Anfrage enthaltenen Variablen keine Informationen. Nach den $\left\lceil \frac{k}{\rho+1} \right\rceil - 1$ Anfragen sind höchstens $k - 1$ wesentliche Variablen bekannt (vgl. Lemma B.9) und der Algorithmus muß noch mindestens eine weitere Anfrage stellen, um alle wesentlichen Variablen zu lernen. □

Ich werde nun eine weitere Idee für eine untere Schranke für $k \leq \frac{n}{2}$ beschreiben. Diese beruht darauf abzuschätzen, wieviele potentielle Auswahlen von k wesentlichen Variablen unmöglich werden, wenn für eine Anfrage eine Null geantwortet wird. Eine wesentliche Grundlage für diesen Ansatz ist die Vermutung, daß sich jeder effiziente Algorithmus für $ANZ_\rho(k)$ in zwei Phasen unterteilen läßt. Dieser Ansatz führt in der Form, wie ich ihn hier beschreibe, **nicht** zu einer Verbesserung der bekannten unteren Schranken.
In Satz 5.9 wird gezeigt, daß für $k \leq \frac{n}{2}$ die wesentlichen Variablen mit $O(k \log \frac{n}{k})$ Anfragen gelernt werden können, sobald eine Menge bekannt ist, die genau ρ wesentliche Variablen enthält, als Anfrage also eine Eins liefert. Andererseits liefert eine Antwort Null einem Algorithmus nur sehr wenig strukturelle Information, da solch eine Antwort nur besagt, daß die Anfrage nicht genau ρ wesentliche Variablen enthält (im Gegensatz z. B. zu

Threshold-Funktionen, bei denen eine Null bedeutet, daß „zu wenig" wesentliche Variablen in der Anfrage sind). Deswegen **vermute** ich, daß jeder effiziente Algorithmus in zwei Phasen unterteilt werden kann. Zunächst stellt er solange Anfragen, bis für eine Menge A eine Eins geantwortet wird. In der zweiten Phase identifiziert er dann mit Hilfe dieser Menge A mit $O(k \log \frac{n}{k})$ Anfragen alle wesentlichen Variablen.
Da die zweite Phase bereits linear in einer bekannten unteren Schranke ist, kann eine Verbesserung der unteren Schranken nur dadurch erreicht werden, daß gezeigt wird, daß die erste Phase sehr viele Anfragen benötigt. Hierfür gebe ich im folgenden einen Ansatz an.[1]
Sei $\mathcal{M}$ die Menge aller möglichen Auswahlen von k wesentlichen Variablen. Dann können bei einer Anfrage A, die eine Null liefert, die Mengen aus $\mathcal{M}$ entfernt werden, die mit A genau ρ-elementigen Schnitt haben. Sei α der maximale Anteil von Mengen, der durch eine Anfrage aus $\mathcal{M}$ entfernt werden kann. Ich zeige, daß ein Algorithmus in der ersten Phase mindestens $\frac{1}{\alpha}$ Anfragen stellen muß, um garantiert eine Eins zu erhalten.
Für eine Menge A sei $\mathcal{M}_A$ die Menge aller Variablenauswahlen, die nach einer Antwort Null auf die Anfrage A noch möglich sind. Für eine Folge von Anfragen $(A_i)_{i\in I}$ muß dann die tatsächliche Auswahl wesentlicher Variablen in $\bigcap_{i\in I} \mathcal{M}_{A_i}$ liegen. Mit $p = |I|$ gilt:

$$\begin{aligned}
|\mathcal{M}_{A_i}| &\geq \binom{n}{k}(1-\alpha) \\
|\bigcup_{i\in I} \mathcal{M}_{A_i}| &\leq \binom{n}{k} \\
|\bigcap_{i\in I} \mathcal{M}_{A_i}| &\geq \binom{n}{k} - p\binom{n}{k}\alpha.
\end{aligned}$$

Die letzte Aussage folgt mit Induktion über p.
Ein Algorithmus muß solange Anfragen stellen, bis er eine Antwort Eins erhält oder die Menge der noch möglichen Variablenauswahlen einelementig wird. Dann gilt $\binom{n}{k} - p\binom{n}{k}\alpha \leq 1$. Daraus folgt

$$p \geq \frac{1}{\alpha}\frac{\binom{n}{k}-1}{\binom{n}{k}} \approx \frac{1}{\alpha}$$

Also muß jeder Algorithmus mindestens $\frac{1}{\alpha}$ Anfragen stellen, um eine Antwort Eins zu erzwingen. Dieser Ausdruck soll nun bestimmt werden.
Sei

$$f(l) := \frac{\binom{l}{\rho}\binom{n-l}{k-\rho}}{\binom{n}{k}}$$

[1] Da dieser Ansatz nicht zu einer neuen unteren Schranke führt, erspare ich der Leserin an einigen Stellen die ausführlichen Rechnungen.

der Anteil von allen Variablenauswahlen, die mit einer beliebigen l-elementigen Anfrage genau ρ-elementigen Schnitt haben (die Parameter n, k und ρ werden im Sinne der Lesbarkeit weggelassen). Dann wird durch eine Anfrage A, die eine Null liefert, genau ein Anteil $f(|A|)$ aller Mengen aus $\mathcal{M}$ entfernt. Es gilt

$$\alpha = \max_l f(l)$$

Für $l = \frac{n\rho}{k}$ ist die erwartete Anzahl wesentlicher Variablen in einer l-elementigen Anfrage genau ρ, daher **vermute** ich, daß dieses l (sofern es ganzzahlig ist) die Funktion f maximiert. Sei

$$T(n, k, \rho) := \sqrt{\frac{(n-k)(k-\rho)\rho}{nk}}$$

Mit Hilfe der Stirlingschen Formel (vgl. Satz B.10) kann

$$\frac{c}{T(n,k,\rho)} \leq \alpha \leq \frac{d}{T(n,k,\rho)} \tag{5.1}$$

mit $c \approx 0,23$ und $d \approx 0,61$ gezeigt werden.
Für festes n und k wird die Funktion T maximal für $\rho = \frac{k}{2}$. Hiermit läßt sich zeigen, daß

$$T(n,k,\rho) \leq k \log \frac{n}{k} \tag{5.2}$$

für jedes $k \leq n - 1$ gilt.
Das Ziel der Rechnung war, die Größe von $\frac{1}{\alpha}$ zu bestimmen, da dies die minimale Anzahl Anfragen angibt, die ein Algorithmus stellen muß, um garantiert mindestens eine Eins als Antwort zu erhalten. In Gleichung 5.1 wird gezeigt, daß $\frac{1}{\alpha} \approx T(n, k, \rho)$ gilt. Deswegen folgt aus Gleichung 5.2, daß mit diesem Ansatz die bekannte untere Schranke $k \log \frac{n}{k}$ nicht verbessert werden kann.

Warum scheitert dieser Ansatz? Das Problem besteht in der Annahme, daß mit jeder Anfrage, die eine Null liefert, stets ein gleichgroßer Anteil von Mengen ausgeschlossen werden kann und daß für je zwei Anfragen die ausgeschlossenen Mengen verschieden sind. Dies ist in der Regel nicht der Fall, da dieser Anteil für „spätere" Anfragen viel kleiner wird, weil viele Mengen bereits zuvor ausgeschlossen wurden. Um dies zu berücksichtigen, müßte man für eine Folge von Anfragen $A_1, \ldots, A_p$ für jede Menge A_i bestimmen, wieviele potentielle Auswahlen von wesentlichen Variablen, die nach den Anfragen $A_1, \ldots, A_{i-1}$ noch möglich waren, durch A_i ausgeschlossen werden. Hiermit **könnte** sich eine Verbesserung der unteren Schranken ergeben. Leider ist mir solch eine Herleitung nicht gelungen.

5.2 Kombinatorische Problemstellung

In diesem Abschnitt wird für $k \leq \frac{n}{2}$ gezeigt, daß nach der ersten Anfrage, die eine Eins liefert, höchstens weitere $O(k \log \frac{n}{k})$ Anfragen benötigt werden, um alle wesentlichen Variablen zu identifizieren. Mit der unteren Schranke $k \log \frac{n}{k}$ aus Satz 5.7 ergibt sich, daß für Anzahl-Funktionen genau dann ein effizienter Algorithmus bekannt ist, wenn man ein System von Anfragen kennt, von denen mindestens eine Anfrage eine Eins liefert. Solche Mengensysteme entsprechen geeigneten Strategien für das Lottospiel.

Satz 5.9: Identifikation aller Variablen nach erster Eins

> *Sei $k \leq \frac{n}{2}$. Wenn ein Algorithmus für $ANZ_\rho(k)$ eine Eins als Antwort erhält, so kann er anschließend mit höchstens $6k \log \frac{n}{k}$ Anfragen alle wesentlichen Variablen identifizieren.*

Beweis: Wegen der Symmetrie von $ANZ_\rho(k)$ werde ich mich auf $\rho \leq \frac{k}{2}$ beschränken.
Sei A eine Anfrage der Größe l, die eine Antwort Eins erzeugt, d. h. genau ρ wesentliche Variablen enthält. Eine Anfrage $B \subseteq A$ liefert genau dann eine Eins, wenn alle ρ wesentlichen Variablen, die in A liegen, auch in B enthalten sind. Dies entspricht gerade einer Konjunktion. Daher können in A alle wesentlichen Variablen mit einem Algorithmus für $AND^l(\rho)$ gelernt werden. Hierfür wird in [24] ein Algorithmus angegeben, der höchstens

$$T_1 = min\left(l-1, \rho \log \frac{l}{\rho} + 2\rho\right)$$

Anfragen stellt. Ich zeige, daß für $k \leq \frac{n}{2}$ und $\rho \leq \frac{k}{2}$ die Abschätzung $T_1 \leq 2k \log \frac{n}{k}$ gilt:

- 1. Fall: $k \leq \frac{n}{e}$. Die Funktion $k \log \frac{n}{k}$ ist[2] für $k \leq \frac{n}{e}$ monoton steigend in k. Daraus folgt $\rho \log \frac{l}{\rho} \leq k \log \frac{n}{k}$. Außerdem gilt $2\rho \leq k \leq k \log \frac{n}{k}$.

- 2. Fall: $\frac{n}{e} < k \leq \frac{n}{2}$. Es gilt $k \log \frac{n}{k} \geq \frac{n}{2}$. Aus $\rho \leq \frac{k}{2} \leq \frac{n}{4}$ folgt $\rho \log \frac{l}{\rho} \leq \frac{n}{4} \log 4 = \frac{n}{2} \leq k \log \frac{n}{k}$. Außerdem gilt $2\rho \leq k \leq \frac{n}{2} \leq k \log \frac{n}{k}$.

Im Komplement von A liegen genau $k - \rho$ wesentliche Variablen. Für eine Menge $C \subseteq \bar{A}$ liefert die Anfrage $A \cup C$ genau dann eine Null, wenn mindestens eine wesentliche Variable in C liegt. Dies entspricht einer negierten Disjunktion. Mit einem entsprechend modifizierten Algorithmus für $OR^{n-l}(k-\rho)$ können alle nicht in A liegenden wesentlichen Variablen identifiziert werden. Dafür wird ebenfalls in [24] ein Algorithmus angegeben, der höchstens

$$T_2 = \min\left(n-l-1, (k-\rho) \log \frac{n-l}{k-\rho}\right) + 2(k-\rho)$$

[2] Zur Diskussion von $k \log \frac{n}{k}$ vgl. Seite 28.

Anfragen benötigt. Ich zeige, daß $T_2 \leq 4k \log \frac{n}{k}$ gilt:

- 1. Fall: $k \leq \frac{n}{e}$. Die Funktion $k \log \frac{n}{k}$ ist monoton steigend in k, also gilt $(k-\rho) \log \frac{n}{k-\rho} \leq k \log \frac{n}{k}$. Außerdem gilt $2(k-\rho) \leq 2k \leq 2(k \log \frac{n}{k})$.
- 2. Fall: $\frac{n}{e} < k \leq \frac{n}{2}$. Es gilt $k \log \frac{n}{k} \geq \frac{n}{2}$. Daraus folgt $(k-\rho) \log \frac{n}{k-\rho} \leq n \leq 2k \log \frac{n}{k}$ und $2(k-\rho) \leq 2k \leq n \leq 2k \log \frac{n}{k}$.

Insgesamt sind nach höchstens $6k \log \frac{n}{k}$ Anfragen alle wesentlichen Variablen bekannt. □

Da $k \log \frac{n}{k}$ eine untere Schranke ist, kann die Klasse $ANZ_\rho(k)$ für $k \leq \frac{n}{2}$ effizient gelernt werden, wenn schnell eine Anfrage gefunden werden kann, die eine Eins erzeugt.

Ich gebe nun eine Problemstellung Φ für ein kombinatorisches Problem an und zeige anschließend, daß dieses im Wesentlichen äquivalent zur Suche nach einer Auswahl von Anfragen für $ANZ_\rho(k)$ ist, von denen stets mindestens eine Anfrage eine Eins liefert.

Definition 5.10: Problemstellung Φ

> *Gegeben:* $n, k, \rho \in \mathbb{N}$ *mit* $2 \leq k \leq \frac{n}{2}$ *und* $1 \leq \rho \leq \frac{k}{2}$
>
> *Gesucht: Ein Mengensystem* $\mathcal{D}$ *aus Teilmengen von* $[n]$ *mit folgender Eigenschaft:*
>
> > *Für jede* k*-elementige Menge* $T \subseteq [n]$ *gibt es mindestens eine Menge* A *in* $\mathcal{D}$*, so daß* $|T \cap A| = \rho$ *gilt.*

Satz 5.11: Beziehung zwischen $ANZ_\rho(k)$ und Φ

> *Seien* $n, k, \rho \in \mathbb{N}$ *mit* $2 \leq k \leq \frac{n}{2}$ *und* $1 \leq \rho \leq \frac{k}{2}$.
>
> 1. *Sei* $\mathcal{D}$ *ein Mengensystem, das Problemstellung* Φ *genügt. Werden die Mengen aus* $\mathcal{D}$ *als Anfragen für* $ANZ_\rho(k)$ *verwendet, so muß mindestens eine dieser Anfragen eine Eins als Antwort liefern.*
> 2. *Sei* Δ *ein Algorithmus für* $ANZ_\rho(k)$ *und* T_Δ *der zugehörige Entscheidungsbaum (vgl. Abbildung 5.2). Sei* t *die maximale Länge eines Pfades von der Wurzel von* T_Δ *aus, der nur aus Null-Kanten besteht. Seien* $A_1, \ldots, A_t$ *die entsprechenden Anfragen. Sei* B *die Menge von wesentlichen Variablen, die der Algorithmus findet, wenn auch auf* A_t *eine Null geantwortet wird, und* B' *eine beliebige* ρ*-elementige Teilmenge von* B.
>
>   *Dann genügt das Mengensystem* $\mathcal{D} = \{A_1, \ldots, A_t\} \cup \{B'\}$ *der Problemstellung* Φ.

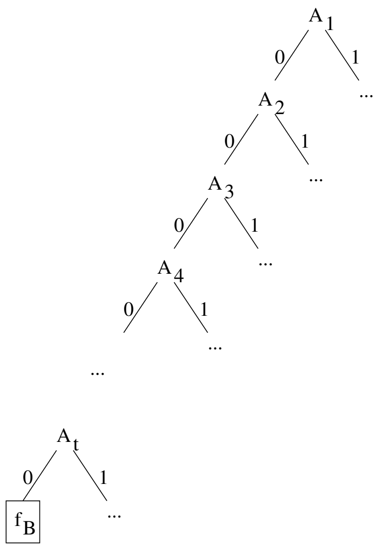

Abbildung 5.2: Baum für den Algorithmus Δ

Beweis: Die erste Aussage ergibt sich direkt aus der Problemstellung Φ, wenn die Menge T mit der Auswahl von k wesentlichen Variablen identifiziert wird.
Ich zeige nun die zweite Aussage. Sei dazu $T \subseteq [n]$ mit $|T| = k$. Die Menge T wird als Auswahl von wesentlichen Variablen in $ANZ_\rho(k)$ interpretiert. Dann gibt es eine Folge von Anfragen in dem Algorithmus Δ, die die Menge T identifiziert. Falls diese Folge keine Eins als Antwort erzeugt, gilt $T = B$. Die Menge B' enthält also genau ρ Variablen aus T.
Andernfalls gibt es ein minimales $1 \leq l \leq t$, so daß der Algorithmus die Anfrage A_l stellt und eine Eins als Antwort erhält. Dann enthält die Anfrage $A_l \in \mathcal{D}$ genau ρ Variablen aus T. In beiden Fällen genügt das oben konstruierte Mengensystem $\mathcal{D}$ also der Problemstellung Φ.

$\square$

Mit Satz 5.9 ergibt sich, daß ein „kleines" Mengensystem $\mathcal{D}$ für Problemstellung Φ direkt einen guten Algorithmus für $ANZ_\rho(k)$ impliziert. Andererseits ist eine untere Schranke für die Größe eines solchen Mengensystems $\mathcal{D}$ auch eine untere Schranke für die Klasse $ANZ_\rho(k)$.
Die Problemstellung Φ entspricht der Suche nach einem Tippsystem für das Lottospiel, mit dem man garantiert gewinnt. Beim Lotto existieren n Kugeln, aus denen in einer Ziehung genau k Kugeln gezogen werden. Gesucht ist nun ein System von Tippreihen, so daß darunter für jede mögliche Ziehung stets mindestens eine Tippreihe ist, in der genau ρ „Richtige" sind. Bei dieser Analogie sind zwei „unrealistische" Eigenschaften zu beachten: Erstens möchte man **genau** ρ Richtige haben, d. h. man verliert auch dann, wenn

man mehr richtige Zahlen getippt hat. Außerdem dürfen die Tippreihen beliebig viele Zahlen haben im Gegensatz zum echten Lotto, bei dem (normalerweise) in jeder Tippreihe genau k Zahlen ausgewählt werden müssen. Ich möchte hier nur einige Stichworte aus Bereichen der Kombinatorik angeben, die sich mit Problemstellungen, die Φ sehr ähnlich sind, beschäftigen. Dies sind die Gebiete Lotto-Garantiesysteme, Steiner-Systeme, Turán-Designs, Blockpläne und Coverings. Viele der Probleme, die in diesen Gebieten untersucht werden, stimmen „fast“, aber nicht genau, mit Φ überein. Bei den Lotto-Garantiesystemen werden z. B. meist nur Mengensysteme untersucht, in denen alle Mengen genau die gleiche Größe haben. Bei Steiner-Systemen wird die Anzahl der Mengen, in denen eine Auswahl von Objekten liegen darf, fest vorgegeben.
Ich habe leider keine Problemstellung gefunden, die Φ genau entspricht, und eine geeignete Umformulierung ist mir auch nicht gelungen.

5.3 Ein Algorithmus für $ANZ_1(k)$

In diesem Abschnitt gebe ich einen Algorithmus für die Klasse $ANZ_1(k)$ an, der $k\binom{\log n}{\log k}$ Anfragen stellt, von denen mindestens eine Anfrage garantiert eine Eins liefert. Für $k \leq \frac{n}{2}$ können anschließend alle wesentlichen Variablen mit $O(k \log \frac{n}{k})$ Anfragen gelernt werden, wie in Satz 5.9 gezeigt wird. Für den Algorithmus wird ein geeignetes System von Pflasterungen konstruiert und als Anfragen verwendet.
Für $k \geq \frac{n}{2}$ ist die untere Schranke $\left\lceil \frac{k}{\rho+1} \right\rceil$ bereits linear in n, da hier $\rho = 1$ gilt. Also folgt aus Satz 2.5, daß bereits ein asymptotisch optimaler Algorithmus bekannt ist. Deswegen werde ich mich auf $k \leq \frac{n}{2}$ beschränken.
Ich werde im Folgenden voraussetzen, daß einige Zahlen, z. B. n und k, Potenzen von 2 sind. Eine Verallgemeinerung auf beliebige natürliche Zahlen ist vermutlich **nicht** trivial.
Eine Menge $M \subseteq [n]$ ist eine Pflasterung, wenn die Menge $[n]$ durch geeignete Verschiebungen von M genau überdeckt werden kann.

Definition 5.12: Pflasterung[3]

> *Seien $m, l \in \mathbb{N}$ mit $1 \leq l \leq m$ Potenzen von 2. Eine Menge $M \subseteq [m]$ mit $|M| = \frac{m}{l}$ heißt* **l-Pflasterung von [m]**, *wenn Zahlen $\delta_1, \ldots, \delta_l \in \mathbb{N}_0$ existieren mit $0 = \delta_1 < \delta_2 < \ldots < \delta_l$, so daß gilt*
> $$\bigcup_{\nu=1}^{l} (\delta_\nu + M) = [m]$$
> *Die Zahlen δ_i werden* **Translationen** *genannt.*

[3] Durch diesen Begriff wird das Titelbild der Arbeit motiviert.

Falls die überdeckte Menge $[m]$ aus dem Zusammenhang eindeutig ist, wird diese nicht explizit angegeben. Offensichtlich liegt die Zahl 1 in jeder Pflasterung, da $\delta_1 = 0$ gilt.
Beispiel: Sei $m = 16$, $l = 4$ und $M = \{1, 3, 9, 11\}$. Dann ist M eine 4-Pflasterung von $\{1, \ldots, 16\}$ mit den Translationen $\delta_1 = 0, \delta_2 = 1, \delta_3 = 4$ und $\delta_4 = 5$. In Abbildung 5.3 wird dieses Beispiel veranschaulicht.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$0 + M$	x		x						x		x					
$1 + M$		x		x						x		x				
$4 + M$					x		x						x		x	
$5 + M$						x		x						x		x

Abbildung 5.3: Eine 4-Pflasterung von $\{1, \ldots, 16\}$ mit $M = \{1, 3, 9, 11\}$

Ich zeige zunächst, daß die zu einer Pflasterung M gehörenden Translationen eindeutig sind. Anschließend geben ich zwei Möglichkeiten an, Pflasterungen zu konstruieren.

Lemma 5.13: Eindeutigkeit von Translationen

Sei M eine l-Pflasterung von $[m]$.

Dann sind die Translationen $\delta_1 < \delta_2 < \ldots < \delta_l$ eindeutig.

Beweis: Ich nehme an, daß zwei verschiedene Systeme von Translationen $\delta_1 < \delta_2 < \ldots < \delta_l$ und $\gamma_1 < \gamma_2 < \ldots < \gamma_l$ für die Pflasterung M existieren, und führe dies zum Widerspruch.
Sei i_0 der minimale Index, so daß $\delta_{i_0} \neq \gamma_{i_0}$ gilt. O. B. d. A. gelte $\delta_{i_0} < \gamma_{i_0}$. Ich betrachte das Objekt $x = \delta_{i_0} + 1$. Da stets $1 \in M$ gilt, folgt $x \in \delta_{i_0} + M$. Für $j < i_0$ kann x nicht in der Menge $\gamma_j + M$ liegen, da diese mit $\delta_j + M$ übereinstimmt. Da $\delta_{i_0} < \gamma_{i_0} < \gamma_{i_0+1} < \ldots < \gamma_l$ gilt, folgt für jedes $j \in \{i_0, \ldots, l\}$, daß $\min(\gamma_j + M) > \delta_{i_0} + 1 = x$ gilt. Also kann x nicht in der Menge

$$\bigcup_{j=1}^{l} (\gamma_j + M)$$

liegen. Dies ist ein Widerspruch zur Voraussetzung, daß M mit den Translationen $\gamma_1, \ldots, \gamma_l$ eine Pflasterung ist. $\square$

Lemma 5.14: Erste rekursive Konstruktion von Pflasterungen

Seien $m \in \mathbb{N}$ und $1 \leq l \leq m$ Potenzen von 2. Sei M eine l-Pflasterung von $[m]$.

Dann ist M auch eine $2l$-Pflasterung von $[2m]$.

Beweis: Da die Menge M eine l-Pflasterung von $[m]$ ist, existieren Translationen $0 = \delta_1 < \delta_2 < \ldots < \delta_l$, so daß

$$\bigcup_{i=1}^{l} (\delta_i + M) = [m]$$

gilt. Außerdem gilt $|M| = \frac{m}{l} = \frac{2m}{2l}$.
Sei

$$\gamma_i := \begin{cases} \delta_i & \text{für } 1 \leq i \leq l \\ m + \delta_{i-l} & \text{für } l+1 \leq i \leq 2l \end{cases}$$

Dann gilt $0 = \gamma_1 < \gamma_2 < \ldots < \gamma_{2l}$ und

$$\begin{aligned} \bigcup_{i=1}^{2l} (\gamma_i + M) &= \bigcup_{i=1}^{l} (\gamma_i + M) \cup \bigcup_{i=l+1}^{2l} (\gamma_i + M) \\ &= \{1, \ldots, m\} \cup \bigcup_{i=l+1}^{2l} ((m + \delta_{i-l}) + M) \\ &= \{1, \ldots, m\} \cup \left(m + \bigcup_{i=1}^{l} (\delta_l + M) \right) \\ &= \{1, \ldots, m\} \cup (m + \{1, \ldots, m\}) \\ &= \{1, \ldots, m\} \cup \{m+1, \ldots, 2m\} \\ &= [2m] \end{aligned}$$

Also ist M eine $2l$-Pflasterung von $[2m]$. □

Lemma 5.15: Zweite rekursive Konstruktion von Pflasterungen

> *Seien $m, l \in \mathbb{N}$ mit $1 \leq l \leq m$ Potenzen von 2. Sei M eine l-Pflasterung von $[m]$.*
>
> *Dann ist $M \cup (m + M)$ eine l-Pflasterung von $[2m]$.*

Beweis: Da die Menge M eine l-Pflasterung von $[m]$ ist, existieren Translationen $0 = \delta_1 < \delta_2 < \ldots < \delta_l$, so daß

$$\bigcup_{i=1}^{l} (\delta_i + M) = [m]$$

gilt. Außerdem gilt $|M| = \frac{m}{l}$.
Sei $M' := M \cup (m + M)$ und $\gamma_i := \delta_i$ für $1 \leq i \leq l$.

Dann gilt $0 = \gamma_1 < \gamma_2 < \ldots < \gamma_l$ und $|M'| = 2|M| = \frac{2m}{l}$. Außerdem gilt

$$\begin{aligned}
\bigcup_{i=1}^{l}(\gamma_i + M') &= \bigcup_{i=1}^{l}(\gamma_i + M) \cup \bigcup_{i=1}^{l}(\gamma_i + (m + M)) \\
&= \bigcup_{i=1}^{l}(\delta_i + M) \cup \left(m + \bigcup_{i=1}^{l}(\delta_i + M)\right) \\
&= \{1, \ldots, m\} \cup (m + \{1, \ldots, m\}) \\
&= \{1, \ldots, m\} \cup \{m + 1, \ldots, 2m\} \\
&= [2m]
\end{aligned}$$

Also ist M' eine l-Pflasterung von $[2m]$. □

Mit Hilfe der rekursiven Konstruktionen von Pflasterungen gebe ich ein System von Pflasterungen $\mathcal{M}$ an, das später für einen Algorithmus für $ANZ_1(k)$ verwendet wird. Das System $\mathcal{M}$ wird so definiert, daß für jede nichtleere, höchstens l-elementige Menge $S \subseteq [m]$ mindestens eine Menge in $\mathcal{M}$ bei geeigneter Verschiebung genau einelementigen Schnitt mit S hat.

Satz 5.16: System von Pflasterungen

Seien $m, l \in \mathbb{N}$ mit $1 \leq l \leq m$ Potenzen von 2.

Dann existiert ein System von Mengen $\mathcal{M}(m,l) = \{M^i |\ i \in I\}$ mit folgenden Eigenschaften:

1. *Jedes $M^i \in \mathcal{M}(m,l)$ ist eine l-Pflasterung von $[m]$. Die Translationen der Pflasterung M^i werden mit δ^i_ν für $1 \leq \nu \leq l$ bezeichnet.*
2. *Es gilt $|\mathcal{M}(m,l)| = \binom{\log m}{\log l}$.*
3. *Für jede Menge $S \subseteq \mathbb{N}$ gilt: Wenn in der Menge $[m]$ mindestens ein und höchstens l Objekte aus S liegen, so gibt es mindestens ein $M^i \in \mathcal{M}(m,l)$ und eine Translation δ^i_ν, so daß $|S \cap (\delta^i_\nu + M^i)| = 1$ gilt.*

Beweis: Seien m und l Potenzen von 2 mit $1 \leq l \leq m$. Ich werde durch Induktion über m für jedes m und l ein geeignetes System von Mengen $\mathcal{M}(m,l)$ angeben, das alle geforderten Eigenschaften erfüllt. Dieses System wird rekursiv definiert.

IA: $m = 1$.
Es gilt $l = 1$ wegen $1 \leq l \leq m$. Das einelementige System $\mathcal{M}(1,1) = \{M^1\}$ mit $M^1 = \{1\}$ und $\delta^1_1 = 0$ erfüllt alle drei Eigenschaften.

m	l	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
8	1	x	x	x	x	x	x	x	x								
8	2	o	o	o	o												
8	2	o	o			o	o										
8	2	o		o		o		o									
8	4	#	#														
8	4	#		#													
8	4	#				#											
8	8	*															
16	4	o	o	o	o												
16	4	o	o			o	o										
16	4	o		o		o		o									
16	4	#	#							#	#						
16	4	#		#						#		#					
16	4	#				#				#				#			
16	8	#	#														
16	8	#		#													
16	8	#				#											
16	8	*								*							
16	16	*															

Abbildung 5.4: Einige Pflasterungen von $\{1,\ldots,8\}$ und $\{1,\ldots,16\}$

IS: $m \to 2m$.
Falls $l = 1$ gilt, so kann ein geeignetes System direkt angegeben werden. Sei dazu $\mathcal{M}(2m,1) = \{M^1\}$ mit $M^1 = \{1,\ldots,2m\}$ und $\delta_1^1 = 0$. Dann erfüllt das System $\mathcal{M}(2m,1)$ alle drei Eigenschaften.
Auch im Fall $l = 2m$ reicht ein einelementiges System von Mengen. Sei $\mathcal{M}(2m,2m) = \{M^1\}$ mit $M^1 = \{1\}$ und $\delta_\nu^1 = \nu - 1$ für $1 \le \nu \le 2m$. Dann erfüllt das System $\mathcal{M}(2m,2m)$ alle drei Eigenschaften.
Ich betrachte nun den Fall $1 < l < 2m$. Da l und m Potenzen von 2 sind, folgt somit $2 \le l \le m$. Also gibt es nach Induktionsvoraussetzung Mengensysteme $\mathcal{P}(m,l) = \{P^i \mid i \in I\}$ und $\mathcal{Q}(m,\frac{l}{2}) = \{Q^j \mid j \in J\}$, die jeweils alle drei Eigenschaften erfüllen.
Hiermit wird das System

$$\mathcal{M}(2m,l) := \{P \cup (m+P) \mid P \in \mathcal{P}(m,l)\} \cup \mathcal{Q}\left(m,\frac{l}{2}\right)$$

definiert. In Abbildung 5.4 sind einige auf diese Weise erzeugte Pflasterungen abgebildet.
Ich zeige nun, daß das so definierte Mengensystem $\mathcal{M}(2m,l)$ alle drei Eigenschaften erfüllt.

Sei $A \in \mathcal{M}(2m,l)$. Falls A von der Form $P \cup (m+P)$ für ein $P \in \mathcal{P}(m,l)$ ist, so ist A eine l-Pflasterung von $[2m]$, wie in Lemma 5.15 gezeigt wird. Andernfalls liegt A in $\mathcal{Q}(m, \frac{l}{2})$ und ist ebenfalls eine l-Pflasterung von $[2m]$, wie in Lemma 5.14 gezeigt wird. Also ist die erste Eigenschaft erfüllt.
Es gilt

$$|\mathcal{M}(2m,l)| \leq |\mathcal{P}(m,l)| + |\mathcal{Q}(m, \frac{l}{2})|$$

Ich zeige, daß die durch $\mathcal{P}(m,l)$ und $\mathcal{Q}(m, \frac{l}{2})$ erzeugten Mengen jeweils verschieden sind. Sei dazu A von der Form $A = P \cup (m+P)$ für ein $P \in \mathcal{P}(m,l)$ und $B \in \mathcal{Q}(m, \frac{l}{2})$. Da P Pflasterung von $[m]$ ist, muß es mindestens ein Element $u \geq 1$ in P geben. Also enthält A mindestens ein Element $u' = m + u > m$. Andererseits liegen in B nur Objekte $v \leq m$, da B Pflasterung von $[m]$ ist. Also sind A und B verschieden.
Die zweite Eigenschaft folgt nun direkt aus der Induktionsvoraussetzung für die Mengensysteme $\mathcal{P}(m,l)$ und $\mathcal{Q}(m, \frac{l}{2})$:

$$\begin{aligned}
|\mathcal{M}(2m,l)| &= |\mathcal{P}(m,l)| + |\mathcal{Q}(m, \frac{l}{2})| \\
&= \binom{\log m}{\log l} + \binom{\log m}{\log \frac{l}{2}} \\
&= \binom{\log m}{\log l} + \binom{\log m}{\log l - 1} \\
&= \binom{\log m + 1}{\log l} \\
&= \binom{\log 2m}{\log l}
\end{aligned}$$

Es bleibt zu zeigen, daß $\mathcal{M}(2m,l)$ die dritte Eigenschaft erfüllt. Sei $S \subseteq \mathbb{N}$ eine beliebige Menge, so daß in der Menge $[2m]$ mindestens ein und höchstens l Objekte aus S liegen. Ich nehme o. B. d. A. an, daß $S \subseteq [2m]$ gilt, da die nicht in $[2m]$ liegenden Elemente von S keinen Einfluß darauf haben, ob die dritte Eigenschaft für $\mathcal{M}(2m,l)$ zutrifft oder nicht.
Es wird unterschieden, wie viele Elemente aus S in der ersten Hälfte $[m]$ liegen:

- 1. Fall: In $[m]$ liegen alle Objekte aus S. Dann gibt es nach Induktionsvoraussetzung eine Pflasterung $P^i \in \mathcal{P}(m,l)$ sowie eine Translation δ^i_ν, so daß $\delta^i_\nu + P^i$ genau ein Element aus S enthält. Da $S \subseteq [m]$ gilt, enthält auch $\delta^i_\nu + (P^i \cup (m + P^i))$ genau ein Objekt aus S. Da $P^i \cup (m + P^i) \in \mathcal{M}(2m,l)$ gilt, folgt hieraus die dritte Eigenschaft.

- 2. Fall: In $[m]$ liegen keine Objekte aus S. Da $S \subseteq [2m]$ gilt, liegen also in der Menge $\{m+1, \ldots, 2m\}$ mindestens ein und höchstens l Elemente aus S. Dies ist weitgehend analog zum ersten Fall mit dem Unterschied, daß die Objekte nun in der zweiten statt in der ersten Hälfte von $[2m]$ liegen. Sei $S' := (-m) + S$. Dann gilt $S' \subseteq [m]$ und $1 \leq S' \leq l$. Also gibt es eine Pflasterung $P^i \in \mathcal{P}(m,l)$ und eine Translation δ^i_ν, so daß
$$|S' \cap (\delta^i_\nu + P^i)| = 1$$
gilt. Wenn beide Mengen um m verschoben werden, ändert sich die Größe des Durchschnitts nicht, es folgt also
$$|(m + S') \cap (m + \delta^i_\nu + P^i)| = 1$$
Da $m + S' = S \subseteq \{m+1, \ldots, 2m\}$ gilt, ändert sich der Durchschnitt auch dann nicht, wenn die zweite Menge um $\delta^i_\nu + P^i$ vergrößert wird, da P^i eine Pflasterung von $[m]$ ist. Hieraus folgt
$$|S \cap (\delta^i_\nu + (P^i \cup (m + P^i)))| = 1$$
Dies ist genau die dritte Eigenschaft, da $P^i \cup (m + P^i)$ in $\mathcal{M}(2m,l)$ liegt und δ^i_ν eine zulässige Translation hierfür ist.

- 3. Fall: In $[m]$ liegen mindestens ein und höchstens $\frac{l}{2}$ Elemente aus S. Dann folgt die dritte Eigenschaft für $\mathcal{M}(2m,l)$ direkt aus der Induktionsvoraussetzung für $\mathcal{Q}(m, \frac{l}{2})$.

- 4. Fall: In $[m]$ liegen mehr als $\frac{l}{2}$, aber nicht alle Objekte aus S. Dann liegen in $\{m+1, \ldots, 2m\}$ mindestens ein und höchstens $\frac{l}{2}$ Elemente aus S. Sei $S_1 := S \cap \{m+1, \ldots, 2m\}$ und $S_2 := (-m) + S_1$. Dann gilt $S_2 \subseteq [m]$ und $1 \leq |S_2| \leq \frac{l}{2}$. Also gibt es eine Pflasterung $Q^i \in \mathcal{Q}(m, \frac{l}{2})$ und eine Translation δ^i_ν mit
$$|S_2 \cap (\delta^i_\nu + Q^i)| = 1$$
Hieraus folgt durch Verschiebung um m, daß ebenfalls
$$|(m + S_2) \cap ((m + \delta^i_\nu) + Q^i)| = 1$$
gilt. Da $m + S_2 = S_1$ und $(m + \delta^i_\nu) + Q^i \subseteq \{m+1, \ldots, 2m\}$ gilt, ergibt sich
$$|S \cap ((m + \delta^i_\nu) + Q^i)| = 1$$
Hieraus folgt direkt die dritte Eigenschaft, da $m + \delta^i_\nu$ eine zulässige Translation für die Pflasterung $Q^i \in \mathcal{M}(2m,l)$ ist.

$\square$

Ich verwende das im letzten Satz konstruierte System von Pflasterungen, um einen Algorithmus für $ANZ_1(k)$ anzugeben.

Satz 5.17: Algorithmus für $ANZ_1(k)$ mit Pflasterungen

> *Seien $n, k \in \mathbb{N}$ mit $1 \leq k \leq n$ Potenzen von 2.*
>
> *Dann gibt es ein System von $k\binom{\log n}{\log k}$ Anfragen für $ANZ_1(k)$, so daß für jede beliebige Auswahl von k wesentlichen Variablen mindestens eine dieser Anfragen eine Eins liefert.*

Beweis: In Satz 5.16 wird gezeigt, daß es für alle Potenzen m und l von 2 mit $1 \leq l \leq m$ ein System von Pflasterungen $\mathcal{M}(m, l)$ mit folgender Eigenschaft gibt:

> Für jede Menge $S \subseteq \mathbb{N}$ gilt: Wenn in der Menge $[m]$ mindestens ein und höchstens l Objekte aus S liegen, so gibt es mindestens ein $M^i \in \mathcal{M}(m, l)$ und eine Translation δ^i_ν, so daß $|S \cap (\delta^i_\nu + M^i)| = 1$ gilt.

Ich setze $m = n$ und $k = l$. Für jede Pflasterung $M^i \in \mathcal{M}(n, k)$ und jede zulässige Translation δ^i_ν wird die Anfrage $\delta^i_\nu + M^i$ gestellt. Für eine beliebige Auswahl von k wesentlichen Variablen liefert mindestens eine dieser Anfragen eine Eins. Dies folgt direkt aus der Eigenschaft des Mengensystems $\mathcal{M}(n, k)$.
Da das Mengensystem $\mathcal{M}(n, k)$ genau $\binom{\log n}{\log k}$ Pflasterungen enthält und für jede Pflasterung k Translationen existieren, werden insgesamt $k\binom{\log n}{\log k}$ Anfragen gestellt.

$\square$

Ich vermute, daß es nicht notwendig ist, für jede Pflasterung aus $\mathcal{M}(n, k)$ alle Verschiebungen anzufragen, da auf diese Weise für jede Auswahl von k wesentlichen Variablen mehrere Mengen gefragt werden, die eine Eins erzeugen. Eine wesentliche Reduktion der Anzahl der Anfragen ist mir mit dieser Idee jedoch nicht gelungen.
In Satz 5.9 wird gezeigt, wie für $k \leq \frac{n}{2}$ die Klasse $ANZ_\rho(k)$ gelernt werden kann, wenn eine Anfrage bekannt ist, die eine Antwort Eins erzeugt. Zusammen mit dem obigen Satz gibt es also einen Algorithmus, der die Klasse $ANZ_1(k)$ mit $O(k \log \frac{n}{k} + k\binom{\log n}{\log k})$ Anfragen lernt, falls k und n Potenzen von 2 sind. Der erste Ausdruck dieser Laufzeit ist linear in der unteren Schranke aus Satz 5.7. Der zweite Term $k\binom{\log n}{\log k})$ ist jedoch nicht linear in einer bekannten unteren Schranke. Daher ist nicht klar, ob der Algorithmus asymptotisch optimal ist oder nicht. Wenn k klein ist (z. B. $k \leq n^{\frac{1}{5}}$), so ist der zweite Ausdruck kleiner als n, so daß der Algorithmus eine sublineare Laufzeit hat und besser als der lineare Algorithmus 2.6 ist.

Kapitel 6

Gewichtete Threshold-Funktionen

In diesem Kapitel untersuche ich gewichtete Threshold-Funktionen mit Gewichten Eins und Zwei. Diese Funktionen geben genau dann eine Eins aus, wenn die gewichtete Summe der Einsen in der Eingabe mindestens so groß ist wie der Thresholdwert.[1] Ich betrachte die Klasse der gewichteten Threshold-Funktionen, bei denen die Anzahl k der wesentlichen Variablen, der Thresholdwert t sowie der Gewichtsvektor bekannt sind. Es wird die Größe dieser Klasse bestimmt und ein in der Anzahl der Variablen n linearer Algorithmus beschrieben. Anschließend gebe ich einen weiteren Algorithmus für $t < k$ an, der für spezielle Parameter eine sublineare Laufzeit hat, z. B. für $k \leq \frac{n}{4}$ die Laufzeit $O(k \log \frac{n}{k})$.

Für einen Gewichtsvektor $a \in \mathrm{IR}^n$ und $t \geq 0$ ist $f : \{0,1\}^n \to \{0,1\}$ eine gewichtete Threshold-Funktion, wenn $f(x) = 1$ genau für $<a, x> \geq t$ gilt, d. h. wenn die gewichtete Summe aller Einsen in x mindestens t beträgt. Ich werde mich auf die Gewichte Eins und Zwei beschränken. Mit dem Gewichtsvektor a ist bekannt, welche der Variablen Gewicht Eins und welche Gewicht Zwei haben. Diese Variablen heißen Eins- bzw. Zwei-Variablen. Der Funktionswert von f hängt nur davon ab, wie viele Eins- bzw. Zwei-Variablen in der Eingabe x mit einer Eins belegt sind, nicht jedoch davon, in welcher Reihenfolge dies geschieht. Daher kann ich o. B. d. A. annehmen, daß die Variablen nach Gewichten aufsteigend geordnet sind, d. h. ich betrachte nur Gewichtsvektoren der Form $a = (1, \ldots 1, 2, \ldots 2)$. Jeder Gewichtsvektor einer Threshold-Funktion mit Gewichten Eins und Zwei kann durch entsprechende Umnummerierung der Variablen in diese Form gebracht werden. Dann ist es für eine solche Threshold-Funktion ausreichend, die Anzahl der Einsen e im Gewichtsvektor sowie den Threshold-Wert t anzugeben, um

[1] Für die gewichteten Threshold-Funktionen bezeichne ich den Thresholdwert mit t und nicht mit τ wie im ungewichteten Fall.

diese eindeutig zu beschreiben.

Definition 6.1: Gewichtete Threshold-Funktion

> *Seien $n \in \mathbb{N}$ und $e, t \in \mathbb{N}_0$ mit $0 \leq e \leq n$. Eine Funktion $f : \{0,1\}^n \to \{0,1\}$ heißt* **gewichtete Threshold-Funktion mit e Eins-Gewichten und Thresholdwert t**, *wenn $f(x) = 1$ genau dann gilt, wenn die gewichtete Summe aller Einsen in x mindestens t ist, d. h. es gilt $<a, x> \geq t$ für den Gewichtsvektor a, der aus e Einsen und $n - e$ Zweien besteht.*
>
> *Man schreibt $f(x) = wthr_{e,t}(x)$.*

Eine ungewichtete Threshold-Funktion (wie in Kapitel 3) ist also eine gewichtete Threshold-Funktion mit Gewichtsvektor $a = (1, \ldots, 1)$ bzw. $e = n$. Die gewichteten Threshold-Funktionen sind **nicht** symmetrisch. Sie bestehen aber aus zwei symmetrischen Teilen, d. h. der Funktionswert ist nur von den Anzahlen der mit Eins belegten Eins- und Zwei-Variablen abhängig, nicht jedoch davon, *welche* Eins- bzw. Zwei-Variablen dies sind.

Definition 6.2: $WTHR^n_{e,t}(k)$

> *Sei $n \in \mathbb{N}$, $k, e, t \in \mathbb{N}_0$ mit $0 \leq k \leq n$ und $0 \leq e \leq n$.*
>
> *Dann besteht die Klasse $WTHR^n_{e,t}(k)$ aus allen gewichteten Threshold-Funktionen mit e Eins-Gewichten und Threshold-Wert t auf n Variablen mit k wesentlichen Variablen. Unwesentliche Variablen werden durch Null ersetzt.*

$$WTHR^{\mathbf{n}}_{\mathbf{e,t}}(\mathbf{k}) := \{wthr_{e,t,K} \mid K \subseteq [n], |K| = k\}$$

Ich schreibe auch $WTHR_{e,t}(k)$, wenn n aus dem Zusammenhang eindeutig bestimmbar ist.
Wenn $k = 0$ oder $k = n$ gilt, so ist die Klasse $WTHR_{e,t}(k)$ einelementig, da es jeweils nur eine Auswahl von wesentlichen Variablen gibt. Für einen Gewichtsvektor a mit e Einsen gilt

$$\sum_{i=1}^{n} a_i = e + 2 \cdot (n - e) = 2n - e$$

Für $t > 2n - e$ besteht die Klasse $WTHR_{e,t}(k)$ nur aus der konstanten Null-Funktion. Falls $t = 2n - e$ gilt, so enthält $WTHR_{e,t}(k)$ die konstante Eins-Funktion für $k = n$ und sonst die konstante Null-Funktion. Wenn $t > 2k$ gilt, so enthält $WTHR_{e,t}(k)$ ebenfalls nur die konstante Null-Funktion. Falls $t = 0$ gilt, so liegt nur die konstante Eins-Funktion in der Klasse $WTHR_{e,0}(k)$. Für $t = 1$ liefert eine Anfrage genau dann eine Eins, wenn sie mindestens eine wesentliche Variable enthält. In diesem Fall kann $WTHR_{e,1}(k)$ mit einem

Algorithmus für $OR(k)$ gelernt werden. Hierfür wird in [24] ein asymptotisch optimaler Algorithmus angegeben. Falls $e = n$ gilt, so gibt es nur Eins-Variablen. Dann gilt $WTHR_{n,t}(k) = THR_t(k)$. Für $e = 0$ haben alle Variablen Gewicht Zwei. Mit $t' = \lceil \frac{t}{2} \rceil$ gilt $WTHR_{0,t}(k) = THR_{t'}(k)$. Diese Klassen werden in Kapitel 3 behandelt.
Sei $t = 2k$ und K eine Auswahl von k wesentlichen Variablen. Falls K mindestens eine Variable mit Gewicht Eins enthält, so ist die zugehörige Funktion $wthr_{e,t,K}$ die konstante Null-Funktion. Andernfalls ist K eine k-elementige Teilmenge der $n - e$ Variablen mit Gewicht Zwei. Für $t' = \lceil \frac{t}{2} \rceil$ entspricht dies genau einer Funktion in $THR_{t'}^{n-e}(k)$. Mit der Anfrage $\{x_1, \ldots, x_n\}$ kann getestet werden kann, ob die gesuchte Funktion die Null-Funktion ist. Andernfalls kann ein Algorithmus für die Klasse $THR_{t'}^{n-e}(k)$ verwendet werden, um die Klasse $WTHR_{e,2k}(k)$ zu lernen. In Kapitel 3 werden untere Schranken und Algorithmen für Klassen von Threshold-Funktionen angegeben.
Alle bisher beschriebenen Fälle sind entweder trivial oder können auf bereits untersuchte Funktionenklassen zurückgeführt werden. In den folgenden Untersuchungen beschränke ich mich daher auf

$$\boxed{n \in \mathbb{N}, 1 \leq k \leq n-1,\ 2 \leq t < min(2n-e, 2k), \text{ und } 1 \leq e \leq n-1}$$

Diese Voraussetzungen gelten für alle nachfolgenden Aussagen und werden gegebenenfalls verschärft.

Bei gewichteten Threshold-Funktionen kann es vorkommen, daß unterschiedliche Auswahlen von wesentlichen Variablen trotzdem gleiche Funktionen erzeugen. Sei $n = 20, k = 5, t = 6$ und $e = 5$. Dann ist die Menge $K = \{x_1, x_6, x_7, x_8, x_9\}$ eine zulässige Auswahl von wesentlichen Variablen. Ich nehme weiter an, daß ein Algorithmus bereits alle Zwei-Variablen gelernt hat. Trotzdem kann die Eins-Variable x_1 nicht identifiziert werden. Jede Anfrage mit mindestens drei Zwei-Variablen hat mindestens Gewicht 6 und liefert eine Eins, so daß das Hinzufügen von Eins-Variablen die Antwort nicht ändert. Ebenso liefert eine Anfrage mit höchstens zwei Zwei-Variablen stets eine Null. Die wesentliche Eins-Variable x_1 kann also nicht identifiziert werden. Dies bedeutet, daß alle Mengen, die eine Eins-Variable und $k-1$ Zwei-Variablen enthalten, die gleichen Funktionen erzeugen. Dies tritt jedoch nur für gerades t auf. In diesen Fällen besteht die Lernaufgabe also darin, die wesentlichen Zwei-Variablen zu finden, da hiermit die ausgewählte Funktion bereits identifiziert ist. Eine analoge Situation ist in den anderen Funktionenklassen bisher nur dann aufgetreten, wenn die konstante Null- oder Eins-Funktion betrachtet wurde.

6.1 Größe der Klasse $WTHR_{e,t}(k)$

In diesem Abschnitt bestimme ich die Größe der Klasse $WTHR_{e,t}(k)$. Es gibt $\binom{n}{k}$ Möglichkeiten, die k wesentlichen Variablen auszuwählen. Einige

dieser Variablenwahlen erzeugen jedoch dieselben Funktionen, z. B. erzeugt eine Variablenwahl die konstante Null-Funktion, wenn sie mehr als $2k - t$ Eins-Variablen enthält. Ich gebe eine Formel mit booleschen Prädikaten an, die die Berechnung der Größe von $WTHR_{e,t}(k)$ ermöglicht. Hieraus kann mit Hilfe von Satz 1.2 direkt eine untere Schranke für die Anzahl notwendiger Anfragen abgeleitet werden.
Mit w wird der Gewichtsvektor der untersuchten Funktion $f \in WTHR_{e,t}(k)$ bezeichnet. Für eine Auswahl U von Variablen sei $wgt(U)$ die Summe der Gewichte dieser Variablen, d. h.

$$wgt(U) := \sum_{x_i \in U} w_i$$

Dies entspricht der gewichteten Summe, wenn alle Variablen aus U mit Eins belegt werden.
Ich gebe zunächst vier Lemmata an, die für eine wesentliche Variable v eine Menge U bestimmen, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt, d. h. die beiden Anfragen U und $U \cup \{v\}$ liefern unterschiedliche Antworten. Diese Lemmata werden im anschließenden Satz verwendet, um für zwei verschiedene Auswahlen von wesentlichen Variablen zu zeigen, daß diese unterschiedliche Funktionen erzeugen.

Lemma 6.3: Trennende Anfrage für ungerades $t \leq k$

Sei $t \leq k$, t ungerade, A eine Auswahl von k wesentlichen Variablen und $v \in A$.

Dann existiert eine Menge $U \subseteq A \backslash \{v\}$, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt.

Beweis: Da t ungerade ist, ist $\frac{t-1}{2}$ ganzzahlig.
Falls mindestens $\frac{t-1}{2}$ Zwei-Variablen in $A \backslash \{v\}$ existieren, kann eine Menge U so gewählt werden , daß genau $\frac{t-1}{2}$ Zwei-Variablen in U liegen. Dann gilt $wgt(U) = t - 1$ und $wgt(U \cup \{v\}) \geq t$.
Andernfalls werden alle l Zwei-Variablen aus $A \backslash \{v\}$ sowie weitere $t - 2l - 1$ Eins-Variablen für die Menge U gewählt. Dann gilt $wgt(U) = t - 1$ und $wgt(U \cup \{v\}) \geq t$. □

Lemma 6.4: Trennende Anfrage für gerades $t \leq k$

Sei $t \leq k$, t gerade, A eine Auswahl von k wesentlichen Variablen mit mindestens zwei Eins-Variablen. Sei $v \in A$ eine Eins-Variable.

Dann existiert eine Menge $U \subseteq A \backslash \{v\}$, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt.

Beweis: Sei $w \neq v$ eine weitere Eins-Variable aus A. Es werden solange Variablen aus $A \backslash \{v, w\}$ in U eingefügt, bis zum ersten Mal $wgt(U) \geq t-2$ gilt. Da jede Variable höchstens Gewicht Zwei hat, gilt $wgt(U) \leq t-1$, da sonst bereits vorher $wgt(U) \geq t-2$ eingetreten wäre. Falls $wgt(U) = t-2$ gilt, so wird w in U eingefügt, andernfalls wird U nicht verändert. Anschließend gilt $wgt(U) = t-1$ und $wgt(U \cup \{v\}) = t$. □

Lemma 6.5: Trennende Anfrage für ungerades $t > k$

> *Sei $t = k + c$ mit $0 < c < k$ und t ungerade. Sei A eine Auswahl von k wesentlichen Variablen mit a Eins-Variablen, wobei $0 \leq a \leq k - c$ gilt. Sei $v \in A$.*
>
> *Dann existiert eine Menge $U \subseteq A \backslash \{v\}$, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt.*

Beweis: Falls v Eins-Variable ist, so existieren weitere $a-1$ Eins-Variablen und $k-a$ Zwei-Variablen. Dann gilt

$$\begin{aligned} wgt(A \backslash \{v\}) &= a - 1 + 2(k-a) \\ &= k + (k-a) - 1 \\ &\geq k + c - 1 \\ &= t - 1 \end{aligned}$$

Falls mindestens eine weitere Eins-Variable $w \neq v$ existiert, so kann U so gebildet werden, daß $wgt(U) = t-1$ gilt. Hiermit folgt $wgt(U \cup \{v\}) = t$. Falls v die einzige Eins-Variable ist, so wird U aus $\frac{t-1}{2}$ Zwei-Variablen gebildet. Dann gilt $wgt(U) = t-1$ und $wgt(U \cup \{v\}) = t$.
Falls v Zwei-Variable ist, so werden solange Variablen aus $A \backslash \{v\}$ in U eingefügt, bis zum ersten Mal $wgt(U) \geq t-2$ gilt. Da jede Variable höchstens Gewicht Zwei hat, gilt $wgt(U) \leq t-1$. Hieraus folgt $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$. □

Lemma 6.6: Trennende Anfrage für gerades $t > k$

> *Sei $t = k + c$ mit $0 < c < k$ und t gerade. Sei A eine Auswahl von k wesentlichen Variablen mit a Eins-Variablen, wobei $2 \leq a \leq k - c$ gilt. Sei $v \in A$.*
>
> *Dann existiert eine Menge $U \subseteq A \backslash \{v\}$, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt.*

Beweis: Falls v Eins-Variable ist, so existieren $a-1$ Eins-Variablen und $k-a$ Zwei-Variablen in A. Dann gilt $wgt(A \backslash \{v\}) \geq t-1$ (vgl. Beweis von Lemma 6.5). Wegen $a \geq 2$ existiert mindestens eine weitere Eins-Variable

$w \neq v$. Daher kann U so gebildet werden, daß $wgt(U) = t - 1$ gilt. Hiermit folgt $wgt(U \cup \{v\}) = t$.
Falls v Zwei-Variable ist, so werden solange Variablen aus $A \backslash \{v\}$ in U eingefügt, bis zum ersten Mal $wgt(U) \geq t - 2$ gilt. Da jede Variable höchstens Gewicht Zwei hat, gilt $wgt(U) \leq t - 1$. Hieraus folgt $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$. □

Im nächsten Satz werde ich die Größe der Klasse $WTHR_{e,t}(k)$ bestimmen. Diese hängt davon ab, wie sich die Parameter e, t und k zueinander verhalten. Um nicht in der Formulierung des Satzes mehrere fast gleiche Formeln für die verschiedenen Fälle notieren zu müssen, verwende ich boolesche Prädikate.

Definition 6.7: Boolesches Prädikat $\{B\}$

> *Sei B eine Aussage, d. h. B kann wahr oder falsch sein. Das* **boolesche Prädikat $\{\mathbf{B}\}$** *ist Eins, wenn B zutrifft, sonst Null.*

Da boolesche Prädikate den Wert Null oder Eins annehmen, kann mit ihnen wie üblich gerechnet werden. Insbesondere kann B mit beliebigen Ausdrücken multipliziert werden. Dies verwende ich, um Fallunterscheidungen direkt in Formeln anzugeben. Wenn z. B. die Variable s für $x > 5$ den Wert x und sonst den Wert $x+3$ annehmen soll, so kann dies mit $s = x + \{x \leq 5\} \cdot 3$ ausgedrückt werden. Dadurch lassen sich komplexe Formeln mit mehreren Fällen sehr kompakt darstellen.

Satz 6.8: Größe von $WTHR_{e,t}(k)$

> *Die Klasse $WTHR_{e,t}(k)$ enthält genau*
>
> $$\sum_{r=\max(0,k-(n-e))}^{\min(e,k,2k-t)} \binom{e}{r}\binom{n-e}{k-r} \quad + \quad \{t > k\} \cdot 1 \qquad - \quad \{t \text{ gerade }, k-1 < n-e\} \cdot (e-1)\binom{n-e}{k-1}$$
>
> *verschiedene Funktionen.*

Beweis: Ich betrachte zunächst, wieviele unterschiedliche Auswahlen von wesentlichen Variablen es gibt. Anschließend untersuche ich, welche dieser Auswahlen gleiche Funktionen erzeugen. Dazu werden mehrere Fälle unterschieden. Falls t gerade ist, so erzeugen alle Auswahlen mit denselben $k-1$ Zwei-Variablen dieselbe Funktion. Falls $t > k$ gilt, so erzeugt jede Variablenwahl mit mehr als $2k - t$ Eins-Variablen die konstante Null-Funktion. Hieraus läßt sich die Anzahl verschiedener Funktionen bestimmen.

Sei K eine beliebige Auswahl von k wesentlichen Variablen. Die Menge K enthalte genau r Eins-Variablen und $k - r$ Zwei-Variablen. Dann muß $0 \leq$

$r \leq e$ und $0 \leq k - r \leq n - e$ gelten. Dies ist äquivalent zu

$$\max(0, k - (n-e)) \leq r \leq \min(e, k)$$

Es gibt genau $\binom{e}{r}$ Möglichkeiten, die Eins-Variablen zu wählen, sowie $\binom{n-e}{k-r}$ Möglichkeiten für die Zwei-Variablen. Insgesamt existieren also

$$\sum_{r=\max(0,k-(n-e))}^{\min(e,k)} \binom{e}{r}\binom{n-e}{k-r} \tag{6.1}$$

verschiedene Variablenwahlen. In [12, S. 169, Gleichung (5.22)] wird gezeigt, daß diese Summe genau $\binom{n}{k}$ ergibt.
Ich untersuche nun, wieviele dieser Variablenwahlen unterschiedliche Funktionen erzeugen.

1. Fall: t ungerade, $t \leq k$.
Seien A und B verschiedene Auswahlen von k wesentlichen Variablen. Ich zeige, daß die durch A und B definierten Funktionen unterschiedlich sind, indem ich eine Variablenbelegung angebe, für die die beiden Funktionen verschiedene Ausgaben erzeugen. Da die beiden Mengen verschieden sind, existiert ein $v \in A \oplus B$, o. B. d. A. gelte $v \in A$.
Sei U eine geeignete Auswahl von Variablen aus A, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt. Solch ein U existiert, wie in Lemma 6.3 gezeigt wird. Die Variablen aus U sowie die Variable v werden mit Eins belegt, alle anderen Variablen mit Null. Auf dieser Eingabe erzeugt die durch A definierte Funktion eine Eins. Die durch B definierte Funktion gibt eine Null aus, da v nicht in B liegt. Die beiden Funktionen sind verschieden.
Daraus folgt, daß in diesem Fall alle möglichen Auswahlen von k wesentlichen Variablen unterschiedliche Funktionen erzeugen. Mit

$$t \leq k \iff 2k - t \geq k$$

und Gleichung 6.1 folgt die Behauptung.

2. Fall: t gerade, $t \leq k$.
Ich zeige, daß zwei Auswahlen von k wesentlichen Variablen genau dann verschiedene Funktionen erzeugen, wenn sie nicht die gleichen $k-1$ Zwei-Variablen enthalten.
Seien A und B verschiedene Auswahlen von k wesentlichen Variablen. Sei $A = A_1 \dot{\cup} A_2$ und $B = B_1 \dot{\cup} B_2$, wobei die jeweils erste Menge die Eins-Variablen und die zweite Menge die Zwei-Variablen enthält.

- Sei $A_2 \neq B_2$. Dann existiert eine Variable $v \in A_2 \oplus B_2$. O. B. d. A. gelte $v \in A_2$. Da $t \leq k$ gilt, läßt sich eine Menge $U \subseteq A \backslash \{v\}$ angeben, so daß $t - 2 \leq wgt(U) \leq t - 1$ gilt. Die Variablen aus U sowie die Variable v werden mit Eins belegt. Für diese Eingabe gibt die durch A erzeugte

Funktion eine Eins aus, die durch B erzeugte Funktion hingegen eine Null. Also sind die beiden Funktionen verschieden.

- Sei $A_2 = B_2$ und $|A_1| \neq 1$. Dann muß $|A_1| \geq 2$ gelten, da die beiden Mengen A und B verschieden sind. Außerdem existiert eine Variable $v \in A_1 \oplus B_1$, o. B. d. A. gelte $v \in A_1$. Dann existiert eine Menge $U \subseteq A$, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt. Dies wird in Lemma 6.4 gezeigt. Die Variablen aus U sowie die Variable v werden mit Eins belegt, alle anderen mit Null. Für diese Eingabe gibt die durch A erzeugte Funktion eine Eins aus, die durch B erzeugte Funktion hingegen eine Null. Also sind die beiden Funktionen verschieden.

- Sei $A_2 = B_2$ und $|A_1| = 1$. Die beiden Mengen unterscheiden sich also nur in der einzigen Eins-Variable. Ich betrachte zunächst die durch A erzeugte Funktion. Sei v die Eins-Variable aus A. Ich zeige, daß für jede Auswahl S von Variablen die Antworten der durch A erzeugten Funktion auf die Anfragen S und $S \cup \{v\}$ gleich sind. Falls S die Variable v enthält, so ist dies klar. Sei also $v \notin S$. Ich nehme o. B. d. A. an, daß S nur Variablen aus A enthält, da unwesentliche Variablen die Antworten nicht beeinflussen.

  Falls die Menge S höchstens $\frac{t}{2} - 1$ Variablen aus $A \backslash \{v\}$ enthält, so gilt $wgt(S) \leq 2(\frac{t}{2} - 1) = t - 2$. Die Antworten auf die Anfragen S und $S \cup \{v\}$ sind beide Null, da v eine Eins-Variable ist.

  Falls die Menge S mindestens $\frac{t}{2}$ Variablen aus $A \backslash \{v\}$ entält, so gilt $wgt(S) \geq 2\frac{t}{2} = t$. Die Antworten auf die Anfragen S und $S \cup \{v\}$ sind beide Eins.

  Da sich die Mengen A und B nur durch die Eins-Variable unterscheiden, können die entsprechenden Funktionen nicht unterschieden werden, da eine einzelne Eins-Variable die Antworten nicht beeinflussen kann.

Ich untersuche nun die Anzahl der verschiedenen Funktionen, die für gerades $t \leq k$ in $WTHR_{e,t}(k)$ liegen. Falls $k - 1 > n - e$ gilt, so kann in einer Auswahl von k Variablen nicht genau eine Eins-Variable liegen. In diesem Fall erzeugen alle Auswahlen von wesentlichen Variablen verschiedene Funktionen. Es liegen also

$$\sum_{r=\max(0,k-(n-e))}^{\min(e,k)} \binom{e}{r}\binom{n-e}{k-r}$$

verschiedene Funktionen in $WTHR_{e,t}(k)$.
Andernfalls gilt $k - 1 \leq n - e$. Die Variablenwahlen, die dieselben $k - 1$ Zwei-Variablen enthalten, erzeugen dieselbe Funktion. Dies bedeutet, daß diese $\binom{e}{1}\binom{n-e}{k-1}$ Variablenwahlen nur $\binom{n-e}{k-1}$ verschiedene Funktionen erzeugen.

Insgesamt liegen in diesem Fall

$$\begin{aligned} &\sum_{r=\max(0,k-(n-e))}^{\min(e,k)} \binom{e}{r}\binom{n-e}{k-r} - \binom{e}{1}\binom{n-e}{k-1} + \binom{n-e}{k-1} \\ = &\sum_{r=\max(0,k-(n-e))}^{\min(e,k)} \binom{e}{r}\binom{n-e}{k-r} - (e-1)\binom{n-e}{k-1} \end{aligned}$$

verschiedene Funktionen in $WTHR_{e,t}(k)$. Mit $t \leq k \iff 2k - t \geq k$ folgt die Behauptung.

3. Fall: t ungerade, $t > k$.
Sei $t = k + c$ mit $0 < c < k$. Ich zeige, daß alle Variablenwahlen mit mehr als $k - c$ Eins-Variablen die Null-Funktion erzeugen. Alle anderen Variablenwahlen definieren verschiedene Funktionen. Seien A und B verschiedene Variablenwahlen mit a bzw. b Eins-Variablen.

- Sei $a, b > k - c$. Es gilt $wgt(A) = a + 2(k - a) = 2k - a < k + c = t$. Also ist die durch A erzeugte Funktion die konstante Null-Funktion. Da gleiches für B gilt, stimmen die beiden Funktionen überein.

- Sei $a > k - c, b \leq k - c$. Die durch A definierte Funktion ist die konstante Null-Funktion. Es gilt

$$\begin{aligned} wgt(B) &= b + 2(k - b) \\ &= 2k - b \\ &\geq k + c \\ &= t \end{aligned}$$

  Die durch B definierte Funktion liefert für die Anfrage B eine Eins. Die beiden Funktionen sind also verschieden.

- Sei $a \leq k - c, b > k - c$. Dies ist analog zum vorherigen Punkt.

- Sei $a, b \leq k - c$. Es existiert eine Variable $v \in A \oplus B$, o. B. d. A. sei $v \in A$. Dann kann eine Menge U von Variablen aus A gefunden werden mit $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$. Dies wird in Lemma 6.5 gezeigt. Die Variablen aus U sowie v werden mit Eins belegt, alle anderen mit Null. Für diese Eingabe liefert die durch A erzeugte Funktion eine Eins, die durch B definierte Funktion hingegen eine Null. Die beiden Funktionen sind also verschieden.

Für Mengen mit mehr als $k - c = 2k - t$ Eins-Variablen ist die hierdurch erzeugte Funktion die konstante Null-Funktion. Alle Mengen mit höchstens $2k - t$ Eins-Variablen erzeugen verschiedene Funktionen. Somit enthält die Klasse $WTHR_{e,t}(k)$ in diesem Fall genau

$$\sum_{r=\max(0,k-(n-e))}^{\min(e,k,2k-t)} \binom{e}{r}\binom{n-e}{k-r} + 1$$

verschiedene Funktionen.

4. Fall: t gerade, $t > k$.
Sei $t = k + c$ mit $0 < c < k$. Ich zeige, daß alle Variablenwahlen mit mehr als $k - c$ Eins-Variablen die Null-Funktion erzeugen. Die anderen Variablenwahlen definieren verschiedene Funktionen, wenn sie nicht genau eine Eins-Variable enthalten.
Seien A und B verschiedene Variablenwahlen mit a bzw. b Eins-Variablen.

1. Unterfall: $n-e < k-1$. Es kann keine Auswahl von k Variablen mit genau einer Eins-Variablen geben. Also muß $a, b > 1$ gelten. Ich unterscheide, ob a und b größer als $k - c$ sind oder nicht, d. h. ob die Funktionen konstant Null sind oder nicht.

- Sei $a, b > k - c$. Dann gilt

$$\begin{aligned} wgt(A) &= a + 2(k - a) \\ &< k + c \\ &= t \end{aligned}$$

  und $wgt(B) < t$. Also sind die durch A und B erzeugten Funktionen beide die konstante Null-Funktion und somit identisch.

- Sei $a > k - c$ und $b \leq k - c$. Dann sind die durch A und B definierten Funktionen verschieden. Die erste Funktion ist die konstante Null-Funktion. Es gilt $wgt(B) \geq t$, d. h. die zweite Funktion ist nicht die Null-Funktion.

- Sei $a \leq k - c$ und $b > k - c$. Dann sind die beiden Funktionen verschieden, wie man analog zum vorherigen Punkt sieht.

- Sei $a, b \leq k - c$. Da A und B verschieden sind, existiert ein $v \in A \oplus B$, o. B. d. A. gelte $v \in A$. Dann existiert eine Menge $U \subseteq A \backslash \{v\}$, so daß $wgt(U) < t$ und $wgt(U \cup \{v\}) \geq t$ gilt. Dies wird in Lemma 6.6 gezeigt. Die Variablen aus U sowie die Variable v werden mit Eins belegt, alle anderen mit Null. Auf dieser Eingabe liefert die durch A erzeugte Funktion eine Eins, die durch B erzeugte Funktion hingegen eine Null.

Die Variablenmengen mit mehr als $k-c = 2k-t$ Eins-Variablen erzeugen alle die konstante Null-Funktion. Alle anderen Variablenauswahlen definieren verschiedene Funktionen. Hieraus folgt die Behauptung.

2. Unterfall: $n - e \geq k - 1$. Sei $A = A_1 \dot{\cup} A_2$ und $B = B_1 \dot{\cup} B_2$, wobei die jeweils erste Menge genau die Eins-Variablen aus A bzw. B enthalte. Ich unterscheide, ob die beiden Mengen A_2 und B_2 übereinstimmen, ob $a = 1$ gilt und ob $a, b > k - c$ gilt.

- Sei $A_2 = B_2$ und $a = 1$. Die beiden Mengen unterscheiden sich nur in der einzigen Eins-Variable. Analog zum zweiten Fall läßt sich zeigen, daß die durch A und B erzeugten Funktionen identisch sind, da die einzige Eins-Variable für gerades t das Antwortverhalten der Funktionen nicht beeinflußt.

- Sei $A_2 = B_2$ und $a > 1$. Wie im 1. Unterfall läßt sich zeigen, daß die durch A und B erzeugten Funktionen genau dann identisch sind, wenn $a, b > k - c$ gilt, da dann beide Mengen die konstante Null-Funktion definieren.

- Sei $A_2 \neq B_2$. Wie im 1. Unterfall läßt sich auch hier zeigen, daß die durch A und B erzeugten Funktionen genau dann identisch sind, wenn $a, b > k - c$ gilt, da dann beide Mengen die konstante Null-Funktion definieren.

Jede Auswahl von mehr als $k - c$ Eins-Variablen erzeugt die konstante Null-Funktion.
Alle Auswahlen von Variablen mit denselben $k - 1$ Zwei-Variablen stimmen überein. Also erzeugen diese $\binom{e}{1}\binom{n-e}{k-1}$ Variablenwahlen nur $\binom{n-e}{k-1}$ verschiedene Funktionen.
Insgesamt liegen in diesem Fall

$$\sum_{r=\max(0,k-(n-e))}^{\min(e,k,2k-t)} \binom{e}{r}\binom{n-e}{k-r} + 1 - (e-1)\binom{n-e}{k-1}$$

verschiedene Funktionen in $WTHR_{e,t}(k)$.

$\square$

Aus der allgemeinen unteren Schranke für Funktionenklassen (vgl. Satz 1.2) und der Größe von $WTHR_{e,t}(k)$ kann natürlich direkt eine untere Schranke für diese Funktionenklasse angegeben werden. Diese untere Schranke ist genau der Logarithmus der Größe der Funktionenklasse $WTHR_{e,t}(k)$ aus dem letzten Satz. Ich verzichte hier auf eine explizite Angabe.

6.2 Ein Algorithmus für $WTHR_{e,t}(k)$ mit in n linearer Laufzeit

Satz 6.9: Linearer Algorithmus für $WTHR_{e,t}(k)$

Die Klasse $WTHR_{e,t}(k)$ kann mit höchstens $2n$ Anfragen gelernt werden.

Beweis: Ich gebe zwei Hilfsalgorithmen an, die mit linearer Suche arbeiten. Beide Algorithmen verwenden höchstens $2n-1$ Anfragen. Mit der Anfrage $\{x_{e+1},\ldots,x_n\}$ kann entschieden werden, welcher der beiden Algorithmen ausgeführt wird. Insgesamt werden höchstens $2n$ Anfragen gestellt. Beide Algorithmen stoppen, sobald k wesentliche Variablen gefunden wurden.

Der erste Algorithmus 6.10 setzt voraus, daß die Menge aller Zwei-Variablen eine Eins erzeugt. Der Algorithmus identifiziert zunächst alle Zwei-Variablen mit linearer Suche. Anschließend unterscheidet der Algorithmus, ob der Thresholdwert t gerade oder ungerade ist, und identifiziert die gesuchte Funktion.
Für ungerades t wählt der Algorithmus genau $\frac{t-1}{2}$ wesentliche Zwei-Variablen. Dies ist möglich, da die Anfrage aller Zwei-Variablen nach Voraussetzung eine Eins liefert. Daher müssen mindestens $\frac{t-1}{2}$ wesentliche Zwei-Variablen existieren. Die gewichtete Summe dieser Variablen ist genau $t-1$. Mit dieser Menge können alle Eins-Variablen identifiziert werden. Damit sind alle wesentlichen Variablen bekannt.
Falls t gerade ist, so muß zunächst ein spezieller Fall betrachtet werden. Falls genau $k-1$ wesentliche Zwei-Variablen existieren, so ist die gewichtete Summe dieser Variablen gerade und es gibt genau eine wesentliche Eins-Variable x. Da die Variable x die Antworten der gesuchten Funktion nicht beeinflußt, stoppt der Algorithmus, da die Funktion bereits vollständig identifiziert ist. Andernfalls gibt es mindestens zwei wesentliche Eins-Variablen. Der Algorithmus wählt $\frac{t}{2}-1$ wesentliche Zwei-Variablen. Die gewichtete Summe dieser Variablen ist genau $t-2$. Es werden solange Eins-Variablen hinzugefügt, bis eine Eins als Antwort erzeugt wird. Mit dieser Menge können alle Eins-Variablen mit linearer Suche gelernt werden.

Der zweite Algorithmus 6.11 setzt voraus, daß die Menge aller Zwei-Variablen eine Null liefert. Zur Menge aller Zwei-Variablen werden höchstens solange Eins-Variablen hinzugefügt, bis eine Antwort Eins erzeugt wird. Falls keine Eins gefunden wird, ist die gesuchte Funktion die konstante Null-Funktion und der Algorithmus stoppt. Anderfalls können alle Eins-Variablen mit linearer Suche identifiziert werden.
Falls mindestens $t-2$ wesentliche Eins-Variablen gefunden wurden, so werden hiermit alle Zwei-Variablen identifiziert. Andernfalls werden zur Menge aller wesentlichen Eins-Variablen solange Zwei-Variablen hinzugefügt, bis

eine Eins erzeugt wird. Hiermit können dann alle Variablen identifiziert werden.

Ich gebe nun die beiden Algorithmen an. Zu jedem Schritt wird am Rand die maximale Anzahl der gestellten Anfragen angegeben, um die Laufzeitberechnung zu vereinfachen.

Algorithmus 6.10:

1. Frage $\{x_{e+1}, \ldots, x_i\}$ für $i = n, \ldots, e$ solange, bis für $i = i_0 - 1$ zum ersten Mal eine Null geantwortet wird. Dann ist x_{i_0} eine wesentliche Variable und $A = \{x_{e+1}, \ldots, x_{i_0}\}$ liefert eine Eins. — Anfragen: $n - i_0 + 2$

2. Frage $A \backslash \{x_{i_0}\} \cup \{x_i\}$ für $i_0 + 1 \leq i \leq n$. Jede Eins identifiziert eine wesentliche Variable. — $n - i_0$

3. Frage $A \backslash \{x_i\}$ für $e + 1 \leq i \leq i_0 - 1$. Jede Null identifiziert eine wesentliche Variable. — $i_0 - e - 1$

4. Falls t ungerade ist: Wähle $\frac{t-1}{2}$ der bekannten wesentlichen Zwei-Variablen aus und bilde die Menge B aus diesen Variablen. Die gewichtete Summe der Variablen aus B ist genau $t - 1$. Frage $B \cup \{x_i\}$ für $1 \leq i \leq e$. Jede Eins identifiziert eine wesentliche Variable. Anschließend sind alle Variablen identifiziert. STOP. — e

5. Andernfalls ist t gerade. Falls genau $k - 1$ wesentliche Zwei-Variablen gefunden wurden, so ist die Funktion bereits vollständig gelernt, da die eine Eins-Variable keinen Einfluß auf die Antworten der Funktion hat. STOP.

6. Andernfalls gibt es mindestens zwei wesentliche Eins-Variablen. Wähle $\frac{t}{2} - 1$ der wesentlichen Zwei-Variablen aus und bilde die Menge C aus diesen Variablen.

7. Frage $C \cup \{x_1, \ldots, x_j\}$ für $1 \leq j \leq e$ solange, bis zum ersten Mal für $j = j_0$ eine Eins geantwortet wird. Dann ist x_{j_0} eine wesentliche Variable. — j_0

8. Frage $C \cup \{x_{j_0}\} \cup \{x_i\}$ für $1 \leq i \leq e$ mit $i \neq j_0$. Jede Eins identifiziert eine wesentliche Variable. — $e - 1$

Laufzeit:
Für ungerades t werden maximal

$$(n - i_0 + 2) + (n - i_0) + (i_0 - e - 1) + e = 2n - i_0 + 1$$

Anfragen gestellt.

Falls t gerade ist, verwendet der Algorithmus höchstens

$$(n - i_0 + 2) + (n - i_0) + (i_0 - e - 1) + j_0 + (e - 1) = 2n - i_0 + j_0$$

Anfragen. Wegen $i_0 > e \geq j_0 \geq 1$ stellt Algorithmus A in beiden Fällen höchstens $2n - 1$ Anfragen.

Algorithmus 6.11:

1. Sei $D = \{x_{e+1}, \ldots, x_n\}$.
2. Frage $D \cup \{x_1, \ldots, x_j\}$ für $1 \leq j \leq e$ höchstens solange, bis zum ersten Mal eine Eins geantwortet wird. Falls keine Eins erzeugt wird, so ist die gesuchte Funktion die konstante Null-Funktion, STOP. Andernfalls werde für $j = j_0$ die erste Eins erzeugt. Dann ist x_{j_0} wesentliche Variable.
   Anfragen: e
3. Frage $D \cup \{x_1, \ldots, x_{j_0}\} \backslash \{x_i\}$ für $1 \leq i \leq j_0 - 1$. Jede Null identifiziert eine wesentliche Variable.
   $j_0 - 1$
4. Frage $D \cup \{x_1, \ldots, x_{j_0 - 1}\} \cup \{x_i\}$ für $j_0 + 1 \leq i \leq e$. Jede Eins identifiziert eine wesentliche Variable.
   $e - j_0$
5. Falls mindestens $t - 2$ wesentliche Eins-Variablen gefunden wurden, so wird eine Menge E gebildet, die aus genau $t - 2$ solcher Variablen besteht. Frage $E \cup \{x_i\}$ für $e + 1 \leq i \leq n$. Jede Eins identifiziert eine wesentliche Zwei-Variable. Anschließend sind alle Variablen bekannt. STOP.
   $n - e$
6. Andernfalls sind weniger als $t - 2$ wesentliche Eins-Variablen bekannt. Sei F die Menge dieser Variablen.
7. Frage $F \cup \{x_{e+1}, \ldots, x_i\}$ für $e + 1 \leq i \leq n$ solange, bis für $i = i_0$ eine Eins geantwortet wird. Die Variable x_{i_0} ist wesentlich.
   $n - e$
8. Frage $F \cup \{x_{e+1}, \ldots, x_{i_0}\} \backslash \{x_m\}$ für $e + 1 \leq m \leq i_0 - 1$. Jede Null identifiziert eine wesentliche Variable.
   $i_0 - e - 1$
9. Frage $F \cup \{x_{e+1}, \ldots, x_{i_0 - 1}\} \cup \{x_m\}$ für $i_0 + 1 \leq m \leq n$. Jede Eins identifiziert eine wesentliche Variable.
   $n - i_0$

Laufzeit:
Falls mindestens $t - 2$ wesentliche Eins-Variablen existieren, so stellt der Algorithmus höchstens

$$e + (j_0 - 1) + (e - j_0) + (n - e) = n + e - 1$$

Anfragen. Andernfalls verwendet der Algorithmus maximal

$$e + (j_0 - 1) + (e - j_0) + (n - e) + (i_0 - e - 1) + (n - i_0) = 2n - 2$$

Anfragen. Da $e < n$ gilt, stellt Algorithmus B in beiden Fällen höchstens $2n - 1$ Anfragen.

□

In manchen Schritten lassen sich einige Anfragen einsparen, indem z. B. ausgenutzt wird, daß in einer Menge die Anzahl der wesentlichen Variablen bekannt ist, oder mit binärer Suche. Hierdurch läßt sich jedoch die Güte des Algorithmus nicht wesentlich verbessern (vgl. Bemerkungen zu Algorithmus 2.6).

6.3 Ein Algorithmus für $WTHR_{e,t}(k)$ mit $t < k$

Ich gebe einen Algorithmus an, der für $t < k$ die Klasse $WTHR_{e,t}(k)$ lernt. Dieser bestimmt zunächst einen minimalen Index s, so daß die Anfrage $x_1, \ldots, x_s$ eine Eins liefert. Anschließend werden alle wesentlichen Variablen identifiziert, indem auf geeigneten Mengen Algorithmen für Disjunktionen und Konjunktionen verwendet werden. Solche Algorithmen werden in [24] angegeben. Bei diesen Algorithmen muß unterschieden werden, ob für eine Menge U, für die ein Algorithmus aufgerufen wird, die Anzahl der wesentlichen Variablen, die in U liegen, bekannt ist oder nicht. Ich werde die Bezeichnung $OR^n(r)$ verwenden, wenn dem Algorithmus die Anzahl r der wesentlichen Variablen bekannt ist, andernfalls OR. Mit $T(OR^n(r))$ bezeichne ich die Laufzeit für einen Algorithmus, der die Anzahl der wesentlichen Variablen r kennt. Falls diese Anzahl dem Algorithmus nicht bekannt ist, wird die Laufzeit mit $T(OR^n, l)$ bezeichnet, wobei l die Anzahl der vom Algorithmus gefundenen wesentlichen Variablen angibt. Für $AND^n(r)$ und AND^n gilt entsprechendes.
Ich werde zu jedem Schritt die maximale Anzahl notwendiger Anfragen am Rand angeben.
Der Algorithmus stoppt, sobald k wesentliche Variablen identifiziert wurden.

Algorithmus 6.12:

1. Mit binärer Suche wird das minimale $s \in [\frac{t}{2}, n-k+t]$ bestimmt, so daß die Anfrage $x_1, \ldots, x_s$ eine Eins liefert. x_s ist wesentliche Variable. Das gesuchte s muß mindestens $\frac{t}{2}$ sein, da sonst eine Antwort Eins nicht möglich ist. Andererseits gibt es nur $n - k$ unwesentliche Variablen, so daß eine Anfrage mit $n - k + t$ Variablen auf jeden Fall eine Eins liefert. Anfragen: $\log(n-k+t-\frac{t}{2}+1)$

2. Falls $s \leq e$ gilt, so folgt $wgt(\{x_1, \ldots, x_s\}) = t$ (vgl. Seite 106) und die Menge $\{x_1, \ldots, x_s\}$ enthält genau t wesentliche Variablen. Mit

einem Algorithmus für $AND^s(t)$ werden alle wesentlichen Variablen in $\{x_1, \ldots, x_s\}$ bestimmt. Mit einem Algorithmus für $OR^{n-s}(k-t)$, in dem jede Anfrage mit der Menge $\{x_1, \ldots, x_{s-1}\}$ ergänzt wird, werden alle wesentlichen Variablen in $\{x_{s+1}, \ldots, x_n\}$ bestimmt. Danach sind alle wesentlichen Variablen bekannt. STOP.

$T(AND^s(t))$

$T(OR^{n-s}(k-t))$

3. Andernfalls gilt $s > e$. Die Variable x_s ist Zwei-Variable. Mit einem Algorithmus für AND^{s-e}, in dem jede Anfrage mit der Menge $\{x_1, \ldots, x_e\}$ ergänzt wird, werden alle wesentlichen Variablen in $\{x_{e+1}, \ldots, x_s\}$ identifiziert. Dies seien l wesentliche Variablen.

   $T(AND^{s-e}, l)$

4. Mit einem Algorithmus für OR^{n-s}, in dem jede Anfrage mit der Menge $\{x_1, \ldots, x_{s-1}\}$ ergänzt wird, werden alle wesentlichen Variablen in $\{x_{s+1}, \ldots, x_n\}$ identifiziert. Dies seien m wesentliche Variablen. Danach sind alle Zwei-Variablen identifiziert. Es sei $r_2 = l + m$.

   $T(OR^{n-s}, m)$

5. Falls $r_2 = k - 1$ gilt und t gerade ist, so kann die verbleibende Eins-Variable nicht identifiziert werden. Die Funktion ist vollständig gelernt. STOP.

6. Falls $r_2 = k - 1$ gilt und t ungerade ist, so kann die wesentliche Eins-Variable mit einer binären Suche auf $x_1, \ldots, x_e$ gelernt werden, bei der jeder Anfrage $\frac{t-1}{2}$ wesentliche Zwei-Variable hinzugefügt werden. Nach dieser Suche sind alle Variablen identifiziert. STOP.

   $\log e$

7. Andernfalls gilt $r_2 < k - 1$. Sei $r_1 = k - r_2$. Falls t und r_1 beide gerade oder beide ungerade sind, so gilt $wgt(\{x_1, \ldots, x_s\}) = t$, da s minimal gewählt wurde. Mit einem Algorithmus für $AND^e(r_1)$, bei dem jeder Anfrage die Menge $\{x_{e+1}, \ldots, x_s\}$ hinzugefügt wird, können alle wesentlichen Eins-Variablen gelernt werden. Danach sind alle Variablen identifiziert. STOP.

   $T(AND^e(r_1))$

8. Falls t und r_1 verschiedene Parität haben, so gilt $wgt(\{x_1, \ldots, x_s\}) = t + 1$. Mit folgender modifizierter binären Suche wird eine wesentliche Eins-Variable identifiziert. Dabei gilt stets, daß $A \cup W$ eine Eins liefert, A mindestens eine wesentliche Variable enthält und W eine Null liefert.

   Zu Beginn wird $A = \{x_1, \ldots, x_e\}$ und $W = \{x_{e+1}, \ldots, x_s\}$ gesetzt. Solange, bis A einelementig wird, werden folgende Schritte ausgeführt:

   - Teile A in zwei annähernd gleichgroße Mengen B und C auf.
   - Frage $B \cup W$.
   - Falls die Antwort Null ist, so entält C mindestens eine wesentliche Variable. Setze $W = W \cup B$ und $A = C$.
   - Falls die Antwort Eins ist, so enthält B mindestens eine wesentliche Variable. Setze $A = B$.

Die in A liegende Variable v ist eine wesentliche Eins-Variable. $\log e$

9. Es gilt $wgt(\{x_1, \dots, x_s\} \backslash \{v\}) = t$. Mit einem Algorithmus für die Klasse $AND^{e-1}(r_1 - 1)$, bei dem v aus jeder Anfrage entfernt und $\{x_{e+1}, \dots, x_s\}$ jeder Anfrage hinzugefügt wird, können alle verbleibenden Eins-Variablen identifiziert werden. $T(AND^{e-1}(r_1 - 1))$

Laufzeit:
In [24] wird gezeigt, daß $AND^a(b)$ bzw. $OR^a(b)$ ohne Kenntnis von b mit jeweils $O(2b(\log \frac{a}{b} + 2))$ Anfragen gelernt werden können. Diese Laufzeitabschätzung wird im Folgenden verwendet:

- Für $s \leq e$:

$$O\left(\log n + t \log \frac{s}{t} + (k - t) \log \frac{n - s}{k - t} + k\right)$$

- Für $s > e, r \geq k - 1$:

$$O\left(\log n + l \log \frac{s - e}{l} + m \log \frac{n - s}{m} + l + m\right)$$

- Für $s > e, r < k - 1, t$ und r_1 gleiche Parität:

$$O\left(\log n + l \log \frac{s - e}{l} + m \log \frac{n - s}{m} + r_1 \log \frac{e}{r_1} + k\right)$$

- Für $s > e, r < k - 1, t$ und r_1 ungleiche Parität:

$$O\left(\log n + l \log \frac{s - e}{l} + m \log \frac{n - s}{m} + r_1 \log \frac{e}{r_1} + k\right)$$

Ich verzichte darauf, hier einen vollständigen Vergleich zwischen den unteren Schranken (vgl. Bemerkung nach Satz 6.8) und der Laufzeit der beschriebenen Algorithmen anzugeben und beschränke mich auf ein Beispiel. Sei $e = \frac{n}{2}$, $t < k$, t ungerade und $k \leq \frac{n}{4}$. Dann ist die Funktion $k \log \frac{n}{k}$ monoton steigend und es gilt $k \leq k \log \frac{n}{k}$. Der Algorithmus 6.12 hat dann Laufzeit $O(k \log \frac{n}{k})$, da $m, l, r_1, r_2 \leq k$ und $s \leq n$ gilt. Die untere Schranke lautet in diesem Fall (vgl. [12, S. 169, Gleichung (5.22)])

$$\begin{aligned}
\left\lceil \log \left(\sum_{r=\max(0,k-(n-e))}^{\min(e,k,2k-t)} \binom{e}{r} \binom{n-e}{k-r} \right) \right\rceil &= \left\lceil \log \left(\sum_{r=0}^{k} \binom{e}{r} \binom{n-e}{k-r} \right) \right\rceil \\
&= \left\lceil \log \binom{n}{k} \right\rceil \\
&\geq k \log \frac{n}{k}
\end{aligned}$$

Also ist der Algorithmus asymptotisch optimal für diese Parameter.

Da gewichtete Threshold-Funktionen den ungewichteten Threshold-Funktionen sehr ähnlich sind, vermute ich, daß sich die entsprechenden Ergebnisse aus Kapitel 3 und aus [24] auf gewichtete Threshold-Funktionen übertragen lassen.

Anhang A

Binäre Suche

Die binäre Suche taucht in verschiedenen Formen an mehreren Stellen in dieser Arbeit auf. Ich möchte hier eine formale Definition der entsprechenden Suchaufgabe angeben und zeigen, daß auf n Objekten mit $\lceil \log n \rceil$ geeigneten Tests ein Objekt identifiziert werden kann.

Eine Suchaufgabe besteht aus einer Eingabemenge $S = \{x_1, \ldots, x_n\}$ und einem Suchkriterium $f : \mathcal{P}(S) \to \{0, 1\}$, das für jede Menge $R \subseteq S$ angibt, ob diese das Kriterium erfüllt ($f(R) = 1$) oder nicht ($f(R) = 0$). Es gibt mindestens einen Index $i \in [n]$, so daß $\{x_i\}$ das Kriterium erfüllt. Wenn eine Menge R das Kriterium erfüllt, so auch jede Obermenge von R. Außerdem existiert in solch einer Menge mindestens ein $x \in R$, so daß $\{x\}$ das Kriterium erfüllt.

Gesucht ist ein Objekt $x_i \in S$, welches das Kriterium erfüllt.

Satz A.1: Binäre Suche

> *Gegeben sei eine allgemeine Suchaufgabe mit Eingabemenge $S = \{x_1, \ldots, x_n\}$ und Suchkriterium $f : \mathcal{P}(S) \to \{0, 1\}$. Diese Aufgabe kann mit binärer Suche mit höchstens $\lceil \log n \rceil$ Aufrufen von f im worst-case gelöst werden.*

Beweis: Die binäre Suche startet mit $R = S$. Die aktuell untersuchte Menge R wird in zwei etwa gleichgroße Teilmengen R_1 und R_2 aufgeteilt, die $\left\lceil \frac{|R|}{2} \right\rceil$ bzw. $\left\lfloor \frac{|R|}{2} \right\rfloor$ Elemente enthalten. Es wird $f(R_1)$ bestimmt. Falls $f(R_1) = 1$ gilt, so wird $R = R_1$, andernfalls $R = R_2$ gesetzt, und die Suche rekursiv fortgesetzt. Die Suche endet, wenn die untersuchte Menge R einelementig ist. Die Menge R erfüllt das Kriterium, das Element $x \in R$ wird als Suchergebnis ausgegeben.

Für die Laufzeitanalyse sei $T(r)$ die worst-case-Laufzeit für eine r-elementige

Menge. Es gilt

$$\begin{aligned} T(1) &= 0 \\ T(r) &= \max\left(1 + T\left(\left\lceil\frac{r}{2}\right\rceil\right), 1 + T\left(\left\lfloor\frac{r}{2}\right\rfloor\right)\right) \\ &= 1 + T\left(\left\lceil\frac{r}{2}\right\rceil\right) \end{aligned}$$

Für eine Eingabemenge der Größe $r = 2^t$ muß der Algorithmus genau t-mal die Menge halbieren, um eine einelementige Menge zu erhalten, also genau t-mal das Kriterium f auswerten. Da die Funktion T monoton steigend ist, gilt $T(r) \leq T(2^{\lceil \log r \rceil})$. Also benötigt der Algorithmus für eine r-elementige Menge im worst-case höchstens $\lceil \log r \rceil$ Aufrufe von f. $\square$

Anhang B

Formelsammlung

Dieses Kapitel enthält eine Zusammenstellung von einigen Formeln und Abschätzungen, die ich im Laufe der Arbeit verwende. Ich verzichte hier bewußt auf „schöne" Überleitungen zwischen den einzelnen Aussagen, da dieser Teil als Formelsammlung gedacht ist und wohl niemand dieses Kapitel als zusammenhängenden Text lesen wird. Aussagen, die ich aus der Literatur übernommen habe, sind durch entsprechende Referenzen am Rand gekennzeichnet. Sofern in der Literatur ein Beweis der Aussagen fehlt, gebe ich einen solchen explizit an.

Binomialkoeffizienten

Lemma B.1: [24, S. 4]

Seien $n, k \in \mathbb{N}$ mit $1 \leq k \leq n-1$.

Dann gilt

(A1) $$\binom{n}{k} \geq \left(\frac{n}{k}\right)^k$$

(A2) $$\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$$

(A3) $$\binom{n}{k} > \frac{1}{e^{\frac{n}{12k(n-k)}}} \cdot \frac{1}{\sqrt{2\pi}} \cdot \frac{n^{n+\frac{1}{2}}}{k^{k+\frac{1}{2}}(n-k)^{n-k+\frac{1}{2}}}$$

(A4) $$\binom{n}{k} < \frac{e^{\frac{1}{12n}}}{\sqrt{2\pi}} \cdot \frac{n^{n+\frac{1}{2}}}{k^{k+\frac{1}{2}}(n-k)^{n-k+\frac{1}{2}}}$$

Beweis: Für die zweite, dritte und vierte Abschätzung wird die Stirlingsche Formel aus Satz B.10 verwendet.

zu A1:

$$
\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\
&= \frac{n(n-1)\ldots(n-k+1)}{k(k-1)\ldots 1} \\
&= \frac{n}{k}\frac{n-1}{k-1}\cdots\frac{n-(k-1)}{k-(k-1)} \\
&\geq \frac{n}{k}\frac{n}{k}\cdots\frac{n}{k} \\
&= \left(\frac{n}{k}\right)^k
\end{aligned}
$$

zu A2:

$$
\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\
&= \frac{n(n-1)\ldots(n-k+1)}{k!} \\
&\leq \frac{n^k}{k!} \\
&\leq \frac{n^k e^k}{e k^k} \\
&\leq \left(\frac{ne}{k}\right)^k
\end{aligned}
$$

zu A3:

$$
\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\
&> \sqrt{2\pi}\frac{n^{n+\frac{1}{2}}}{e^n} \cdot \frac{1}{\sqrt{2\pi}}\frac{e^k}{k^{k+\frac{1}{2}}}\frac{1}{e^{\frac{1}{12k}}} \cdot \frac{1}{\sqrt{2\pi}}\frac{e^{n-k}}{(n-k)^{n-k+\frac{1}{2}}}\frac{1}{e^{\frac{1}{12(n-k)}}} \\
&= \frac{1}{\sqrt{2\pi}}\frac{n^{n+\frac{1}{2}}}{k^{k+\frac{1}{2}}(n-k)^{n-k+\frac{1}{2}}}\frac{1}{e^{\frac{1}{12k}}e^{\frac{1}{12(n-k)}}} \\
&= \frac{1}{\sqrt{2\pi}}\frac{n^{n+\frac{1}{2}}}{k^{k+\frac{1}{2}}(n-k)^{n-k+\frac{1}{2}}}\frac{1}{e^{\frac{n}{12k(n-k)}}}
\end{aligned}
$$

zu A4:

$$\begin{aligned}
\binom{n}{k} &= \frac{n!}{k!(n-k)!} \\
&< \sqrt{2\pi}\frac{n^{n+\frac{1}{2}}}{e^n}e^{\frac{1}{12n}} \cdot \frac{1}{\sqrt{2\pi}}\frac{e^k}{k^{k+\frac{1}{2}}} \cdot \frac{1}{\sqrt{2\pi}}\frac{e^{n-k}}{(n-k)^{n-k+\frac{1}{2}}} \\
&= \frac{1}{\sqrt{2\pi}}\frac{n^{n+\frac{1}{2}}}{k^{k+\frac{1}{2}}(n-k)^{n-k+\frac{1}{2}}}e^{\frac{1}{12n}}
\end{aligned}$$

□

Lemma B.2:

Seien $n, k \in \mathbb{N}$ mit $1 \leq k \leq \frac{n}{3}$.
Dann gilt

$$\text{(B1)} \qquad \binom{n}{k-1} \leq \frac{1}{2}\binom{n}{k}$$

$$\text{(B2)} \qquad \sum_{r=1}^{k}\binom{n}{r} \leq 2\binom{n}{k}$$

Beweis:
zu B1:

$$\begin{aligned}
\binom{n}{k} &= \frac{n \cdot \ldots \cdot (n-k+1)}{k \cdot \ldots \cdot 1} \\
&= \frac{n-k+1}{k}\binom{n}{k-1} \\
&\geq \left(3 - 1 + \frac{3}{n}\right)\binom{n}{k-1} \\
&\geq 2\binom{n}{k}
\end{aligned}$$

zu B2:
Ich zeige zunächst mit Induktion, daß für $0 \leq i \leq k-1$ die Abschätzung

$$\binom{n}{k-i} \leq \left(\frac{1}{2}\right)^i \binom{n}{k}$$

gilt. Der Fall $i = 0$ ist trivial.

IA: $i = 1$. Dies ist die Aussage von der Aussage B1 in diesem Lemma.
IS: $i \to i + 1$.

$$\begin{aligned}
\binom{n}{k-(i+1)} &= \binom{n}{k-i-1} \\
&\leq \frac{1}{2}\binom{n}{k-i} \\
&\leq \left(\frac{1}{2}\right)^{i+1}\binom{n}{k}
\end{aligned}$$

Hieraus folgt

$$\begin{aligned}
\sum_{r=1}^{k}\binom{n}{r} &= \sum_{i=0}^{k-1}\binom{n}{k-i} \\
&\leq \sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^{i}\binom{n}{k} \\
&= \binom{n}{k}\sum_{i=0}^{k-1}\left(\frac{1}{2}\right)^{i} \\
&\leq 2\binom{n}{k}
\end{aligned}$$

□

Lemma B.3:

Sei $n \in \mathbb{N}$.

Dann gilt

$$\sum_{k=1}^{n} k \cdot \binom{n}{k} = n2^{n-1}$$

Beweis:

$$\begin{aligned}
\sum_{k=1}^{n} k \cdot \binom{n}{k} &= \sum_{k=1}^{n}\frac{n!}{(k-1)!(n-k)!} \\
&= \sum_{k=1}^{n} n\frac{(n-1)!}{(k-1)!(n-1-k+1)!} \\
&= \sum_{k=1}^{n} n\binom{n-1}{k-1} \\
&= n\sum_{k=0}^{n-1}\binom{n-1}{k} \\
&= n2^{n-1}.
\end{aligned}$$

□

O-Notation

Lemma B.4: [24, S. 17]

Seien $n, k \in \mathbb{N}$ *mit* $k = \Omega(\sqrt{\frac{n}{\log n}})$ *und* $k \leq \frac{n}{2}$.

Dann gilt

$$\frac{n}{k} = O\left(k \log \frac{n}{k}\right)$$

Beweis: Es existiert eine Konstante $c > 0$, so daß $k \geq c\sqrt{\frac{n}{\log n}}$ für genügend großes n gilt. Ich zeige, daß eine Konstante $d > 0$ existiert, so daß

$$\frac{n}{k} \leq dk \log \frac{n}{k} \tag{B.1}$$

für genügend großes n gilt. Hieraus folgt die Behauptung.
In Lemma B.15 wird gezeigt, daß die Funktion $k^2 \log \frac{n}{k}$ für $k \leq \frac{n}{\sqrt{e}}$ monoton steigend in k ist. Daher wird die rechte Seite minimal, wenn k minimal wird. Da $k \geq c\sqrt{\frac{n}{\log n}}$ gilt, genügt es, für $k = c\sqrt{\frac{n}{\log n}}$ zu zeigen, daß eine geeignete Konstante d existiert. Sei $d := \frac{2}{c^2}$. Dann gilt für großes n (z. B. $n \geq 2^{c^2}$)

$$\begin{aligned}
& 1 \leq \frac{\log \frac{n \log n}{c^2}}{\log n} \\
\Longleftrightarrow \quad & n \leq d\frac{c^2}{2} \frac{\log \frac{n \log n}{c^2}}{\log n} \\
\Longleftrightarrow \quad & n \leq d\frac{c^2}{\log n} \log \sqrt{\frac{n \log n}{c^2}} \\
\Longleftrightarrow \quad & n \leq dc^2 \frac{n}{\log n} \log \frac{n}{c\sqrt{\frac{n}{\log n}}}
\end{aligned}$$

Hieraus folgt mit $k = c\sqrt{\frac{n}{\log n}}$ die Ungleichung B.1.

□

Lemma B.5: [24, S. 18]

Sei $n, k \in \mathbb{N}$ *mit* $k = O(n^{1-\epsilon})$ *für ein* $\epsilon > 0$.

Dann gilt

$$k \log k = O\left(k \log \frac{n}{k}\right)$$

Beweis: Sei $k \leq cn^{1-\epsilon}$ für eine Konstante $c > 0$ und n groß genug. Ich zeige, daß eine Konstante $d > 0$ existiert, so daß

$$k \log k \leq dk \log \frac{n}{k}$$

für genügend großes n gilt.
Nach Division durch k ist die linke Seite der obigen Ungleichung monoton steigend und die rechte Seite monoton fallend in k, daher tritt der „schlechteste" Fall in dieser Ungleichung genau dann ein, wenn k maximal wird. Deswegen reicht es, $k = cn^{1-\epsilon}$ zu betrachten. Dann ist zu zeigen, daß

$$\log\left(cn^{1-\epsilon}\right) \leq d \log\left(\frac{1}{c}n^{\epsilon}\right)$$

gilt. Diese Ungleichung ist äquivalent zu

$$c^{1+d} \leq n^{(d+1)\epsilon-1}$$

Für $d := \frac{2-\epsilon}{\epsilon}$ gilt $(d+1)\epsilon - 1 = 1$ und somit die obige Ungleichung für genügend großes n.

$\square$

Lemma B.6:

Seien $n, k \in \mathbb{N}$ mit $k = O(n^{1-\epsilon})$ für ein $\epsilon > 0$.

Dann gilt

$$k \log n = O\left(k \log \frac{n}{k}\right)$$

Beweis: Sei $k \leq cn^{1-\epsilon}$ für eine Konstante $c > 0$ und n groß genug.
Ich definiere die Konstante $d := \frac{2}{\epsilon}$ und zeige, daß für großes n die Ungleichung

$$\log n \leq d \log \frac{n}{k}$$

gilt. Hieraus folgt die Behauptung.
Da der rechte Term monoton fallend in k ist, reicht es, $k = cn^{1-\epsilon}$ zu betrachten. Dann gilt für n groß genug:

$$\begin{aligned}
& n \leq \frac{n^2}{c^d} \\
\iff \quad & n \leq \left(\frac{n^{\epsilon}}{c}\right)^d \\
\iff \quad & n \leq \left(\frac{n}{cn^{1-\epsilon}}\right)^d \\
\iff \quad & n \leq \left(\frac{n}{k}\right)^d \\
\iff \quad & \log n \leq d \log \frac{n}{k}
\end{aligned}$$

□

Lemma B.7: [24, S. 16]

> *Sei $n, k \in \mathbb{N}$ mit $1 \leq k \leq n$.*
> *Dann gilt $k = \Omega(\sqrt{\frac{n}{\log n}})$ oder es existiert ein $\epsilon > 0$, so daß $k = O(n^{1-\epsilon})$ gilt.*

Beweis: Falls $k \geq \sqrt{\frac{n}{\log n}}$ gilt, so folgt direkt $k = \Omega(\sqrt{\frac{n}{\log n}})$.
Andernfalls gilt $k < \sqrt{\frac{n}{\log n}}$, also auch $k < \sqrt{n}$ und somit $k = O(n^{\frac{1}{2}}) = O(n^{1-\epsilon})$ für $\epsilon = \frac{1}{2}$. □

Gaußklammern

Lemma B.8:

> *Seien $a, b \in \mathbb{R}$ mit $0 \leq b \leq a$.*
> *Dann gilt $\lceil a \rceil - \lfloor b \rfloor \leq \lceil a - b \rceil + 1$.*

Beweis: Seien a_g und b_g die ganzzahligen Anteile und a_r und b_r die nicht-ganzzahligen Anteile von a und b, d. h. es gilt

$$\begin{aligned} a &= a_g + a_r \\ b &= b_g + b_r \end{aligned}$$

mit $a_g, b_g \in \mathbb{N}_0$ und $0 \leq a_r, b_r < 1$. Mit der folgenden vollständigen Fallunterscheidung ergibt sich die Behauptung.

- 1. Fall: $a_r = b_r = 0$. Es gilt $\lceil a \rceil - \lfloor b \rfloor = a_g - b_g$ und $\lceil a - b \rceil = a_g - b_g$.
- 2. Fall: $a_r = 0, b_r > 0$. Es gilt $\lceil a \rceil - \lfloor b \rfloor = a_g - b_g$ und $\lceil a - b \rceil = a_g - b_g + \lceil -b_r \rceil = a_g - b_g$.
- 3. Fall: $a_r > 0, b_r = 0$. Es gilt $\lceil a \rceil - \lfloor b \rfloor = a_g + 1 - b_g$ und $\lceil a - b \rceil = a_g - b_g + 1$.
- 4. Fall: $a_r > 0, b_r > 0$.
  - 1. Unterfall: $a_r = b_r$: Es gilt $\lceil a \rceil - \lfloor b \rfloor = a_g + 1 - b_g$ und $\lceil a - b \rceil = a_g - b_g$.
  - 2. Unterfall: $a_r > b_r$: Es gilt $\lceil a \rceil - \lfloor b \rfloor = a_g + 1 - b_g$ und $\lceil a - b \rceil = a_g - b_g + \lceil a_r - b_r \rceil = a_g - b_g + 1$, da $0 < a_r - b_r < 1$.
  - 3. Unterfall: $a_r < b_r$: Es gilt $\lceil a \rceil - \lfloor b \rfloor = a_g + 1 - b_g$ und $\lceil a - b \rceil = a_g - b_g + \lceil a_r - b_r \rceil = a_g - b_g$, da $-1 < a_r - b_r < 0$.

□

Lemma B.9:

Sei $k, p \in \mathbb{N}$ mit $1 \leq p \leq k$.

Dann gilt

$$\left(\left\lceil\frac{k}{p}\right\rceil - 1\right) * p \leq k - 1$$

Beweis: Sei $k = ap + r$ mit $a \in \mathbb{N}_0$ und $0 \leq r < p$.
1. Fall: $r = 0$.

$$\begin{aligned}\left(\left\lceil\frac{k}{p}\right\rceil - 1\right) p &= (a-1)p \\ &= k - p \\ &\leq k - 1\end{aligned}$$

2. Fall: $r > 0$.

$$\begin{aligned}\left(\left\lceil\frac{k}{p}\right\rceil - 1\right) p &= \left(\left\lceil\frac{ap+r}{p}\right\rceil - 1\right) p \\ &= \left(a + \left\lceil\frac{r}{p} - 1\right\rceil\right) p \\ &= ap \\ &= k - r \\ &\leq k - 1\end{aligned}$$

□

Weitere Formeln

[18, S. 530] **Satz B.10: Stirlingsche Formel**

Sei $n \in \mathbb{N}$. Dann gilt

$$\sqrt{2\pi}\frac{n^{n+\frac{1}{2}}}{e^n} < n! < \sqrt{2\pi}\frac{n^{n+\frac{1}{2}}}{e^n} e^{\frac{1}{12n}}$$

Lemma B.11:

Seien $a, B, C > 0$.

Dann gilt

$$\log_C a = \frac{\log_B a}{\log_B C}$$

Beweis:

$$\begin{aligned} & \log_C a = \frac{\log_B a}{\log_B C} \\ \Longleftrightarrow \quad & \log_C a \log_B C = \log_B a \\ \Longleftrightarrow \quad & \left(B^{\log_B C}\right)^{\log_C a} = a \\ \Longleftrightarrow \quad & a = a \end{aligned}$$

□

Lemma B.12:

Seien $A, B > 0$.

Dann gilt

$$A^{\log B} = B^{\log A}$$

Beweis:

$$\begin{aligned} A^{\log B} &= A^{\frac{\log_A B}{\log A}} \\ &= B^{\frac{1}{\log A}} \\ &= B^{\frac{\log_A A}{\log A}} \\ &= B^{\log A} \end{aligned}$$

□

Lemma B.13:

Sei $n \in \mathbb{N}$.

Dann gilt

$$\sum_{l=1}^{n} l2^l = (n-1)2^{n+1} + 2$$

Beweis: Die Behauptung wird durch Induktion über n gezeigt:
IA: $n = 1$.

$$\begin{aligned} \sum_{l=1}^{1} l2^l &= 2 \\ &= (1-1)2^2 + 2 \end{aligned}$$

IS: $n \to n + 1$.

$$\begin{aligned} \sum_{l=1}^{n+1} l2^l &= \sum_{l=1}^{n} l2^l + (n+1)2^{n+1} \\ &= (n-1)2^{n+1} + 2 + n2^{n+1} + 2^{n+1} \\ &= 2n2^{n+1} + 2 \\ &= ((n+1)-1)2^{(n+1)+1} + 2 \end{aligned}$$

□

Lemma B.14:

Sei $r \in \mathbb{N}$. Für $i \in [r]$ sei $a_i \in]0, n]$, so daß

$$\sum_{i=1}^{r} a_i = n$$

gilt.

Dann wird $\sum_{i=1}^{r} \log a_i$ maximal genau dann, wenn $a_i = \frac{n}{r}$ für alle $i \in [r]$ gilt.

Beweis: Es existieren $\mu_i \in]0, 1]$, so daß $a_i = \mu_i n$ und $\sum_{i=1}^{r} \mu_i n = n$ gilt. Dann folgt

$$\begin{aligned} \sum_{i=1}^{r} \log a_i &= \sum_{i=1}^{r} \log(\mu_i n) \\ &= \sum_{i=1}^{r} (\log \mu_i + \log n) \\ &= r \log n + \sum_{i=1}^{r} \log \mu_i \\ &= \log n^r + \log \prod_{i=1}^{r} \mu_i \end{aligned}$$

Da der Logarithmus eine streng monoton steigende Funktion ist, wird der Ausdruck

$$\log \prod_{i=1}^{r} \mu_i$$

genau dann maximal, wenn

$$\prod_{i=1}^{r} \mu_i$$

maximal wird. Das ist genau dann der Fall, wenn alle μ_i gleichgroß sind, also $\mu_i = \frac{1}{r}$ gilt. Daraus folgt $a_i = \mu_i n = \frac{n}{r}$.

□

Lemma B.15:

Sei $n \in \mathbb{N}$ und $f : \mathbb{R}^+ \to \mathbb{R}$ mit $f(x) = x^2 \log \frac{n}{x}$.

Dann ist f monoton steigend für $x \leq \frac{n}{\sqrt{e}}$.

Beweis: Die Funktion f ist differenzierbar nach x und für die Ableitung f' gilt

$$f'(x) = \frac{2x}{\ln 2} \ln \frac{n}{x} - \frac{x}{\ln 2}$$

Die Funktion f ist genau dann monoton steigend, wenn $f'(x) \geq 0$ gilt. Es folgt

$$\begin{aligned} f'(x) \geq 0 \quad &\Longleftrightarrow \quad \frac{2x}{\ln 2} \ln \frac{n}{x} \geq \frac{x}{\ln 2} \\ &\Longleftrightarrow \quad \ln \frac{n}{x} \geq \frac{1}{2} \\ &\Longleftrightarrow \quad \frac{n}{x} \geq \sqrt{e} \\ &\Longleftrightarrow \quad \frac{n}{\sqrt{e}} \geq n \end{aligned}$$

□

Lemma B.16:

Sei $n, k \in \mathbb{N}$ mit $n \geq 2$ und $1 \leq k \leq \frac{n}{2}$.

Dann gilt

$$\log n \leq k \log \frac{n}{k}$$

Beweis: Sei $f(x) := x \log \frac{n}{x} - \log n$. Dann ist die Funktion f differenzierbar und es gilt

$$\begin{aligned} f'(x) &= \log n - \log x - x \frac{1}{x \ln 2} \\ &= \log \frac{n}{x} - \log e \end{aligned}$$

Für die Ableitung f gilt dann

$$\begin{aligned} f'(x) > 0 \quad &\Longleftrightarrow \quad \log \frac{n}{x} > \log e \\ &\Longleftrightarrow \quad \frac{n}{x} > e \\ &\Longleftrightarrow \quad x < \frac{n}{e} \end{aligned}$$

Analog läßt sich $f'(x) < 0 \iff x > \frac{n}{e}$ zeigen. Die Funktion f ist also für $x < \frac{n}{e}$ monoton steigend und danach monoton fallend. Für $n \geq 2$ gilt

$$\begin{aligned} f(1) &= 0 \\ f(\frac{n}{2}) &= \frac{n}{2} - \log n \\ &\geq 0 \end{aligned}$$

Daraus folgt $f(k) \geq 0$ für $1 \leq k \leq \frac{n}{2}$ und somit die Behauptung. □

Anhang C

Symbolverzeichnis

$\bar{A}$	Komplement der Menge A
$\|A\|$	Größe der Menge A
$A \backslash B$	Mengendifferenz der Mengen A und B
$A \oplus B$	Symmetrische Mengendifferenz der Mengen A und B
$a + A$	Verschiebung der Menge A um a
$\mathcal{P}(A)$	Potenzmenge der Menge A
$\min A$	Kleinstes Element der Menge A
$\max A$	Größtes Element der Menge A
$\min(a_1, \ldots, a_l)$	Kleinstes Element der Zahlen $a_1, \ldots, a_l$
$\max(a_1, \ldots, a_l)$	Größtes Element der Zahlen $a_1, \ldots, a_l$
x_i	i-te Koordinate des Vektors x
$<x, y>$	Skalarprodukt der Vektoren x und y
$<x, y>_r$	Skalarprodukt der ersten r Koordinaten
$\|x\|_1$	Anzahl der Einsen im Vektor x
$\|x\|_0$	Anzahl der Nullen im Vektor x
$\log a$	Binärer Logarithmus von a
$\ln a$	Natürlicher Logarithmus von a
$\lceil a \rceil$	Kleinste ganze Zahl größer oder gleich a
$\lfloor a \rfloor$	Größte ganze Zahl kleiner oder gleich a
$\binom{n}{k}$	Binomialkoeffizient „n über k“
$\{B\}$	Boolesches Prädikat B
$P(X)$	Wahrscheinlichkeit für das Eintreten von Ereignis X
$f_{a,b}$	Schreibweise für $(f_a)_b$
$f = O(g)$	Funktion f wächst asymptotisch höchstens so schnell wie Funktion g
$f = \Omega(g)$	Funktion f wächst asymptotisch mindestens so schnell wie Funktion g
$f = \Theta(g)$	Funktion f wächst asymptotisch genauso schnell wie Funktion g
$[u, v]$	Abgeschlossenes Intervall zwischen u und v in $\mathbb{R}$
$]u, v[$	Offenes Intervall zwischen u und v in $\mathbb{R}$
$[n]$	Menge der natürlichen Zahlen von 1 bis n

Anhang D

Variablenverzeichnis

n	Variablenzahl
k	Anzahl der wesentlichen Variablen
K	Menge der wesentlichen Variablen
k'	Anzahl der unwesentlichen Variablen
v	Wertevektor von symmetrischen Funktionen
τ	Thresholdwert von Threshold-Funktionen
q	Modulus von Modulo-Funktionen
ρ	Ordnung von Anzahl-Funktionen
t	Thresholdwert von gewichteten Threshold-Funktionen
e	Anzahl der Einsvariablen bei gewichteten Threshold-Funktionen
f, g	Boolesche Funktionen
x	Vektor der Länge n
R	Zufällige Menge
$\mathcal{D}, \mathcal{B}$	Mengensysteme
Γ, Δ	Algorithmen
Φ, Π, Ψ	Problemstellung oder Invarianten
δ, γ	Translationen von Pflasterungen
P, Q, M	Pflasterungen
$\mathcal{M}, \mathcal{P}, \mathcal{Q}$	Systeme von Pflasterungen
ϵ	kleine Konstante

Literaturverzeichnis

[1] RUDOLF AHLSWEDE UND INGO WEGENER. „Suchprobleme“. Teubner Studienbücher Mathematik. B. G. Teubner, Stuttgart (1979).

[2] DANA ANGLUIN. „Queries and Concept Learning“. *Machine Learning* **2**, 319–342 (1988).

[3] DANA ANGLUIN, LISA HELLERSTEIN UND MAREK KARPINSKI. „Learning Read-Once Formulas with Queries“. *Journal of the Association for Computing Machinery* **40**(1), 185–210 (1993).

[4] AVRIM BLUM, LISA HELLERSTEIN UND NICK LITTLESTONE. „Learning in the Presence of Finitely or Infinitely Many Irrelevant Attributes“. *Journal of Computer and System Sciences* **50**, 32–40 (1995).

[5] IL‘JA N. BRONSTEIN UND KONSTANTIN A. SEMENDJAJEW. „Taschenbuch der Mathematik“. Nauka, Leipzig, 24. Auflage (1989).

[6] ZHIXIANG CHEN UND STEVEN HOMER. „Learning Counting Functions with Queries“. *Theoretical Computer Science* **180**, 155–168 (1997).

[7] CHARLES J. COLBOURN. „Winning the Lottery“. In C. J. COLBOURN UND J. H. DINITZ (Herausgeber), *The CRC Handbook of Combinatorial Designs*, Kapitel 8, 578 – 584. CRC Press (1996).

[8] PETER DAMASCHKE. „Adaptive versus Nonadaptive Attribute-Efficient Learning“. In *Proc. of 30th ACM Symposium on Theory of Computing*, 590–596 (1998).

[9] PAUL FISCHER, NORBERT KLASNER UND INGO WEGENER. „On the Cut-off Point for Combinatorial Group Testing“. *Discrete Applied Mathematics* **91**, 83–92 (1998).

[10] PHILIPPE FLAJOLET UND ROBERT SEDGEWICK. „An Introduction to the Analysis of Algorithms“. Addison-Wesley, New York (1996).

[11] HENRY W. GOULD. „Combinatorial Identities“. Morgantown Printing and Publishing Co., Morgantown (1972).

[12] Ronald L. Graham, Donald E. Knuth und Oren Patashnik. „Concrete Mathemathics“. Addison-Wesley, New York (1990).

[13] Branko Grünbaum und Geoffrey C. Shepard. „Tilings & Patterns - An Introduction“. Freeman and Company, New York (1989).

[14] Ralf Hartmut Güting. „Datenstrukturen und Algorithmen“. Leitfäden und Monographien der Informatik. B. G. Teubner, Stuttgart (1992).

[15] Joachim Hartung, Bärbel Elpelt und Karl-Heinz Klösener. „Statistik“. Oldenburg-Verlag, München (1982).

[16] Tibor Hegedüs und Piotr Indyk. „On Learning Disjunctions of Zero-One Threshold Functions with Queries“. In *Algorithmic Learning Theory*, 446–460. Springer, Berlin (1997).

[17] Harry Joe. „An Ordering of Dependence for Distribution of k-Tuples, with Applications to Lotto Games“. *The Canadian Journal of Statistics* **15**(3), 227–238 (1987).

[18] Alexander M. Mood, Franklin A. Graybill und Duane C. Boes. „Introduction to the Theory of Statistics“. McGraw-Hill, New York, Dritte Auflage (1974).

[19] M. Morley und G. H. John van Rees. „Lottery Schemes and Covers“. *Utilitas Mathematica* **37**, 159–166 (1990).

[20] Rajeev Motwani und Prabhakar Raghavan. „Randomized Algorithms“. Camebridge Univ. Press, Camebridge (1995).

[21] Walter Oberschelt. „Lotto-Garantiesysteme und Blockpläne“. *Mathematisch-Physikalische Semesterberichte* **19**, 55–67 (1972).

[22] Christine Ritzmann. „Die selbstabstoßende Irrfahrt mit Drift“. Diplomarbeit, Universität Zürich, Philosophische Fakultät II (1996).

[23] Lothar Sachs. „Angewandte Statistik“. Springer, Berlin, Fünfte Auflage (1978).

[24] Ryuhei Uehara, Kensei Tsuchida und Ingo Wegener. „Optimal Attribute-Efficient Learning of Disjunction, Parity and Threshold Functions“. In *Proc. of 3rd European Conference on Computational Learning Theory, EuroColt*, 171–184 (1997).

[25] Ingo Wegener. „Effiziente Algorithmen für grundlegende Funktionen“. Leitfäden und Monographien der Informatik. B. G. Teubner, Stuttgart (1989).

Formalien

Hiermit erkläre ich, daß ich die Diplomarbeit mit dem Titel

> Attribut-effizientes Lernen in Klassen von symmetrischen Funktionen und Threshold-Funktionen

selbstständig und nur unter Verwendung der angegebenen Hilfsmittel angefertigt habe. Zitate wurden kenntlich gemacht.

Dortmund, den 25. Februar 1999 Mark Cieliebak

Ich erkläre mich einverstanden, daß meine Diplomarbeit nach § 6 (1) des URG der Öffentlichkeit durch die Übernahme in die Bereichsbibliothek zugänglich gemacht wird. Damit können Leser der Bibliothek die Arbeit einsehen und zu persönlichen wissenschaftlichen Zwecken Kopien aus der Arbeit anfertigen. Weitere Urheberrechte werden nicht berührt.

Dortmund, den 25. Februar 1999 Mark Cieliebak