Average-Komplexitätstheorie
und
Average-Analyse von Algorithmen

Vorlesungsskript

Wintersemenster 2003/2004

Christian Schindelhauer

Heinz Nixdorf Institut
Fakultät EIM, Institut für Informatik
Universität Paderborn

15. Mai 2004

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Organisation

1.1 Inhalt der Veranstaltung

Im ersten Teil der Veranstaltung werden Grundbegriffe der Average-Komplexitätstheorie vorgestellt. Diese Spezialisierung der Komplexitätstheorie charakterisiert Probleme hinsichtlich der durchschnittlichen Laufzeit, die zur Lösung notwendig sind. Hierfür ist zusätzlich zur Problembeschreibung auch die Wahl der betrachteten Wahrscheinlichkeitsverteilung notwendig.

Wir untersuchen hier zunächst das Boolesche Erfüllbarkeitsproblem (Satisfiability) und wie ein Parameter in der Wahrscheinlichkeitsverteilung, diese Komplexität dieses Problems verändert. Danach führen wir den Begriff der Average-NP-Vollständigkeit ein, der die Menge aller NP-vollständigen Problem beschreibt, die auch im Durchschnitt schwierig sind bezüglich natürlicher Wahrscheinlichkeitsverteilungen.

Im zweiten Teil stellen wir ein Zeitmodell für Boolesche Schaltkreise vor und untersuchen effiziente Schaltkreisentwürfe für die erwartete Zeit. Hierbei wenden wir uns dann komplexitätstheoretischen Überlegungen zu und fragen nach bösartigen Wahrscheinlichkeitsverteilungen für Schaltkreise. Ferner untersuchen wir, wie schwiereig es ist, das Laufzeitverhalten von Schaltkreisen zu unterschen.

Als Überleitung zum dritten Teil untersuchen wir einen im Durchschnitt effizeinten Vergleichsschaltkreis für das Sortieren. Danach wenden wir uns der Average-Analyse von $\mathcal{NP}$-schwierigen Problemen, nämlich dem Färbungsproblem und dem Rucksackproblem, zu und stellen Algorithmen vor, die diese in erwartet polynomieller Zeit lösen.

1.2 Durchführung

Im ersten und dritten Teil wird die Veranstaltung als klassische Tafelvorlesung in einem eher mathematischen Stil gehalten. Im mittleren Teil werden zumeist Folien verwendet. Die Übungen werden als drei ganztägige Blockveranstaltung gehalten.

Die Termine sind:

- Mittwoch, 17.03. 2004, 10-17 Uhr (F2.315)
- Dienstag, 23.03. 2004, 10-17 Uhr (F2.315)
- Montag, 05.04.2004, 10-17 Uhr (F2.315) **Achtung Terminänderung**

1.3 Prüfung

Im Anschluß kann der Stoffinhalt im Rahmen einer halbstündigen mündlichen Prüfung (nach DPO 3) abgelegt werden. Die Prüfungsfragen werden in den Blockübungen vorgestellt. An dieser Stelle werden sie noch einmal aufgeführt.

1.3.1 Prüfungsfragen

1. Teil: Average-Komplexitätstheorie

   - Geben Sie ein paar Beispiele für Probleme, die im worst-case schwierig sind und im Durchschnitt einfach.
   - Wie sind Zufallsgraphen definiert?
   - Warum folgt aus dem erwartet polynomiellen Zeitverhalten eines $\mathcal{NP}$-Problems nicht, dass alle $\mathcal{NP}$-Problem effizient berechnet werden können.
   - Welches Problem löst der Davis-Putnam-Algorithmus?
   - Beschreiben Sie, den DPLL-Algorithmus.
   - Untersuchen Sie, ob der DPLL-Algorithms 2-SAT in polynomieller Zeit berechnet.
   - Was weiß man über das Average-Verhalten des DPLL-Algorithmus.
   - Was versteht man unter der Schwellwertvermutung von SAT?
   - Was versteht man unter effizient im Average?
   - Was sind Dist$\mathcal{NP}$-Probleme in der Average-Komplexitätstheorie?
   - Wie funktionierte eine Average-Reduktion?
   - Nennen Sie ein Average-NP-vollständiges Problem.
   - Erklären Sie, warum Levin einen von dem Erwartungswert abweichenden Begriff eingeführt hat.
   - Welche Eigenschaften hat das Levinsche Maß?
   - Definieren Sie POL-computable und Dist$\mathcal{NP}$.
   - Beschreiben Sie die deterministische Average-Reduktion.
   - Wann dominiert eine Wahrscheinlichkeitsverteilung eine andere? Wozu wird dieser Begriff benötigt?
   - Beschreiben Sie eine Average-Reduktion.
   - Beschreiben Sie das generische vollständige Dist$\mathcal{NP}$-vollständige Problem.
   - Welche randomisierten Komplexitätsklassen kann man als effizient betrachten?
   - Beschreiben Sie die randomisierte Average-Reduktion.
   - Was sind bösartige Wahrscheinlichkeitsverteilungen?
   - Was versteht man unter POL-sampleable?
   - In wiefern unterscheidet sich $\mathcal{NP}$ $\times$ POL-sampleable von Dist$\mathcal{NP}$.

2. Teil: Average-Schaltkreistheorie

   - Was sind unifrom konstruierbare Schaltkreisfamilien?
   - Benennen und erläutern Sie Standard-Komplexitätsklassen für Schaltkreise!
   - Welche Beziehungen sind zwischen $\mathcal{NC}$, $\mathcal{AC}$, $\mathcal{L}$ und $\mathcal{P}$ bekannt?
   - Wie kann man Zeit für Schaltkreise definieren?
   - Ist der Begriff der Zeit in Schaltkreisen abhängig von der Wahl der Basis?
   - Vergleichen Sie die implizite und die explizite Definition der Zeit in Schaltkreisen!
   - Welche Average-Maße untersucht man für Schaltkreise?
   - Welche Schaltkreiskonstruktionen sind für die Disjunktion besonders zeiteffizient?
   - Was sind verteilungserzeugende Vorschaltkreise und wofür werden sie verwendet?
   - Geben Sie obere und untere Schranken für die Average-Schaltkreiszeit-Komplexität der Disjunktion an.

- Welche Funktionen in der Average-Schaltkreis-Komplexität können im Average nicht beschleunigt werden?
- Geben Sie die Definition der Dependancy an und erläutern Sie ihre Bedeutung!
- Wie schnell kann man im Durchschnitt mit Schaltkreisen addieren?
- Haben Schaltkreise bösartige Wahrscheinlichkeitsverteilungen? Was sind Worst-Case-Wahrscheinlichkeitsverteilung in diesem Zusammenhang?
- Wie komplex ist die Vorhersage der Schaltkreiszeit einer Eingabe, der worst-case-Zeit und der erwarteten Zeit eines Schaltkreises?

3. Teil: Effiziente Algorithmen im Average
   -

1.4 Literaturhinweise

1. Garey, Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness* [GJ78].

   Ein Standardwerk für die (Worst-Case)-Komplexitätstheorie und über den Umgang mit $\mathcal{NP}$-schwierigen Problemen. Siehe hierin auch den Beitrag zum Hamiltionschen Pfadproblem und Graphfärbung.

2. Wilf, *Algorithms and Complexity* [Wil87].

   Etwas angestaubt, aber immer noch gut zu lesen. Wilf gibt hier eine Schulbuchdarstellung seiner Average-Analyse des Graph-Färbungs-Algorithmus.

3. Cook, Mitchell, *Finding Hard Instances of the Satisfiability Problem: A Survey* [CM97].

   Der Erfinder der $\mathcal{NP}$-Vollständigkeit gibt hier mit Mitchell zusammen, eine sehr schönen Überblick über den Stand der Forschung im Bereich der Worst-Case-Eingaben für das SAT-Problem. Hier findet man auch eine akurate geschichtliche Entwicklung samt dem Davis-Putnam-Loveland-Logemann-Algorithmus.

4. Levin, *Problems complete in "average" instance*, [Lev84]

   Auf diesen zwei Seiten führt der Miterfinder NP-vollständigkeit dem Leser das gesamte Gebäude der Average-Komplexitätstheorie ein. Für den kryptographisch interessierten Leser sehr empfehlenswert (meint Gurevich). Erklärt wird dieser Artikel in [Gol97] und [Gur91]

5. Yamakami, *Average Case Computational Complexity Theory*, [Yam97]

   Diese ausführliche Dissertationsschrift ist ein Nachschlagewerk, das jedem der auf dem Gebiet der Average-Komplexitätstheorie arbeiten möchte nur wärmstens ans Herz gelegt werden kann.

6. Für eine Einführung in die Komplexitätstheore empfehler wir Reischuk, *Eine Einführung in die Komplexitätstheorie* [Rei99], sowie Papadimitriou, *Computaional Complexity*, [Pap94].

7. Zwichen der Average-Komplexitätung der Kolmogoroff-Komplexität bestehen enge Beziehungen. Leider konnte aus zeitlichen Gründen dieser Zusammenhang im Rahmen dieser Vorlesung nicht herausgearbeitet werden. Für einen Überblick zum Theme Kolmogoroff-Komplexitäte lese man das ausgezeichnete Standardwerk von Li und Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications* [LV93].

8. Für eine Einführung in das Gebiet der Average-Schaltkreis-Theorie wird der Konferenzartikel [JRS94] empfohlen.

9. Das effiziente Sortiernetzwerk beruhend auf einer kleinen Fehlerrate findet sich in [LP90]. Leider ist das Skript an dieser Stelle (nie) fertig geworden.

10. Bertholds Vöckings Analyse des Nemhauser-Ullmann-Algorithmus findet man in [BV03].

Kapitel 2

Motivation und Einführung

Warum beschäftigt man sich mit dem durchschnittlichen Zeitverhalten von Algorithmen, statt den schlimmsten Fall, sprich *worst-case*, zu betrachten? Wir werden hier zwei Beispiel dem dritten Kapitel vorgreifen, um eine Motivation zu bieten.

2.1 Zwei "leichte" $\mathcal{NP}$-schwierige Probleme

2.1.1 Der Hamiltonsche Pfad

Das Problem des Hamiltonschen Pfad (HP), ist für einen gegebenen ungerichteten Graph, wenn möglich einen Pfad zu finden, der alle Knoten des Graphen genau einmal beeinhaltet und dabei nur die Kanten des Graphs benutzt.

Dieses Problem wird allgemein als schwierig erachtet, was aus dem folgenden Satz gefolgert wird.

Theorem 1 *[Kar72] Das Entscheidungsproblem von HP ist $\mathcal{NP}$-vollständig.*

Unter dem Entscheidungsproblem eines solchen Suchproblems verstehen wir die Frage, ob eine Lösung gibt. Wir betrachten das Suchproblem für einen zufälligen Graph $G_{n,\frac{1}{2}}$. Die Definition dieses Zufallsgraphen ist wie folgt:

Definition 1 *Für $n \in \mathbb{N}$ und $p \in [0,1]$ bezeichnet $G_{n,p}$ eine Zufallsvariable über alle ungerichteten Graphen mit n Knoten, wobei die Wahrscheinlichkeit, dass eine Kante vorkommt, p ist. Und dieses Ereignis ist unabhängig von dem Vorkommen aller Kanten ist.*

Betrachtet man aber den Average-Case kann man einen Algorithmus mit folgender Eigenschaft finden.

Theorem 2 *Es gibt einen Algorithmus, der für einen Zufallsgraph $G_{n,\frac{1}{2}}$ HP in erwarteter linearer Laufzeit löst.*

Erwartete lineare Zeit bedeutet hier, dass ein Algorithmus A existiert der für einen Graph x die Laufzeit $\text{time}_A(x)$ besitzt. Die erwartete Laufzeit ist dann $\mathbf{E}_{x \leftarrow G_{n,p}}[\text{time}_A(x)]$, wobei der Erwartungswert einer Zufallsvariable X hierbei definiert ist durch

$$\mathbf{E}[X] := \sum_{x \in \text{ran}(X)} x \cdot \mathrm{P}[X = x] \; .$$

Wie kann man diese beiden Ergebnisse logisch vereinbaren? Eine mögliche Erklärung ist, dass die meisten Graphen hinsichtlich dieses Problems einfach sind. Zum Beispiel, dadurch dass der Graph nicht zusammenhängend ist oder dass sehr viele Hamiltonsche Pfade existieren und die Suche nach einer möglichen Lösung dadurch sehr einfach ist. Tatsächlich arbeitet der Average-effiziente Algorithmus nach diesem Prinzip [Kap90].

2.1.2 3-Färbung

Ein ungerichteter Graph $G = (V, E)$ hat eine gültige k-Färbung, wenn es eine Abbildung $f : V \to \{1, \ldots, k\}$ gibt, so dass für jede Kante die Knoten verschiedene Farben besitzen, d.h. falls $\{u, v\} \in E \Rightarrow f(u) \neq f(v)$.

Das 3-Färbungsproblem ist für einen gegebenen Graphen ein 3-Färbung zu bestimmen, falls diese existiert. Das entsprechende Entscheidungsproblem ist, die Frage zu beantworten, ob solche eine Färbung des Graphen prinzipiell möglich ist. Auch dieses Problem ist algorithmisch schwierig:

Theorem 3 *Das Entscheidungsproblem des 3-Färbungsproblem ist $\mathcal{NP}$-vollständig.*

Wir werden im dritten Kapitel folgenden Satz beweisen.

Theorem 4 *[BW85] 3-Färbung in $G_{n,\frac{1}{2}}$ kann erwartungsgemäß in konstanter Zeit gelöst werden.*

Um sich die Plausibilität dieses Ergebnis von Wilf und Bender [BW85] zu veranschaulichen, überlegt man sich folgendes. Ein Graph aus vier Knoten besitzt ganau dann keine 3-Färbung, wenn der Graph eine Clique beschreibt. Die Wahrscheinlichkeit, dass vier Knoten eine Clique bilden ist in $G_{n,\frac{1}{2}}$ genau 2^{-6}. Scannen wir den Graphen jetzt in vierer Gruppen nach solchen 4-Cliquen benötigen wir erwartete Zeit 128 bis diese Suche erfolgreich ist.

Die Effizienz ergibt sich also aus der Tatsache, dass praktisch jeder Graph aus $G_{n,\frac{1}{2}}$ keine 3-Färbung besitzt und dass diese sehr effizient nachgewiesen werden kann.

2.2 Vorausgesetzte Grundlagen

Es werden Grundkenntnisse im Bereich der (worst-case)-Komplexitätstheorie vorausgesetzt. Jedem Algorithmus liegt ein bestimmtes Maschinenmodell zugrunde. Wir werden die folgenden Modelle verwenden.

- DTM: Deterministische Turing-Maschine. Mit Hilfe einer Mehrband-Turing-Maschine werden die Komplexitätsklassen $\mathrm{DTIME}(T)$ und $\mathrm{DSPACE}(S)$ definiert und damit die Verallgemeinerungen

$$\mathcal{P} := \mathrm{DTIME}(\mathrm{POL}) , \qquad \mathrm{PSPACE} := \mathrm{DSPACE}(\mathrm{POL}) , \qquad \mathcal{L} := \mathrm{DSPACE}(\mathrm{LOG}) .$$

- NTM: Nichtdeterministische Turing-Maschine. Besonders interessierung uns bei diesem Modell die Komplexitätsklasse der in Zeit T lösbaren Probleme, genannt $\mathrm{NTIME}(T)$, und damit insbesondere

$$\mathcal{NP} := \mathrm{NTIME}(\mathrm{POL}) .$$

- RAM: Random Access Machine. Dieses moderne Maschinenmodell werden wir für die Beschreibung von effizienten Algorithmen verwenden. Im Gegensatz zur TM besitznen RAM einen festen Befehlssatz, wahlfrei zugreifbarer Speicher, in dem insbesondere frei indiziert werden kann.

- Schaltkreise. Das sind kreisfreie (!) gerichtete Graphen, deren Knoten Boolesche Funktionen beschreiben. An den Quellen wird die Eingabe einmalig als Bit eingelesen, jede Kante kann nur ein Bit weitergeben. Die Ausgabe ist an bestimmten ausgezeichneten Knoten erhältlich. Hierbei unterscheiden wir uniforme und nicht-uniforme Schaltkreisfamilien.

Diese genannten Komplexitätsklassen beschreiben die Worst-Case-Komplexität von Problemen. Wir werden sehen, dass die Verallgemeinerung auf in den Average-Case daraus nicht zwangsläufig folgt. Um eine Einführung in das Gebiet der Worst-Case-Komplexitätstheorie zu erhalten, sei aus [Rei99] oder [Pap94] verwiesen.

2.3 Erfüllbarkeit Boolescher Funktionen

Die Booleschen Konstanten bezeichnen wir mit $\mathcal{B} := \{0, 1\}$. Als Operationen betrachten wir die Negation $\neg x = \overline{x}$, die Disjunktion (x *oder* y) = $x \vee y$, die Konjunktion (x *und* y) = $x \wedge y$.

Für diese Operationen gelten das Kommutativgesetz, Distributivgesetz, das Assoziativgesezt und die DeMorganschen Umformungsregeln.

Funktionen können rekursiv aufbauend auf den Variablen x_i oder den Konstanten $0, 1$ wie folgt aufgebaut werden. Hierbei bezeichne $X := (x_1, \ldots, x_n)$.

$$F(X) \;\leftarrow\; \begin{cases} 0\,, \\ 1\,, \\ x_i\,, \text{für ein } i \text{ aus } \{1, \ldots, n\} \\ \neg F_1(X)\,, \\ F_1(X) \vee F_2(X)\,, \\ F_1(X) \wedge F_2(X)\,. \end{cases}$$

Als **Quantoren** kennt man den Existenzquantor $\exists$ und den All- oder Universalquantor $\forall$. Hierbei gilt

$$\exists x : F(x) :\Longleftrightarrow F(0) \vee F(1)\,,$$

$$\forall x : F(x) :\Longleftrightarrow F(0) \wedge F(1)\,.$$

Unter einer **quantifizierten Booleschen Funktion** (QBF) versteht man einen Term

$$Q_1 x_1 Q_2 x_2 \ldots Q_n x_n \;\; F(x_1, \ldots, x_n)$$

für $Q_i \in \{\exists, \forall\}$ und einer Booleschen Funktion F.

Davis und Putnam stellten 1960 [DP60] einen Algorithmus vor, der solche Formeln auswertet. Natürlich funktioniert dieser Algorithmus auch für Formeln, die nur Existenzquantorn umfassen. Diese Formeln beschreiben das Erfüllbarkeitsproblem Boolescher Funktionen.

2.3.1 Satisfiability

Gegeben ist hier eine Formel

$$Q = \exists x_1 \ldots \exists x_n \quad F(x_1, \ldots, x_n)\,,$$

wobei $F(X) = F(x_1, \ldots, x_n)$ eine Boolesche Funktion beschreibt. Wie aus dem Grundstudium bekannt, ist die Berechnung dieses Prädikats $\mathcal{NP}$-vollständig. Hierbei verringert sich die Komplexität des Problems nicht, wenn man statt allgemeiner Booleschen Funktion sich auf Funktionen in konjunktiver Normalform (CNF) beschränkt.

(CNF) Eine Formel $F(X)$ liegt in CNF vor, wenn F als Konjunktion von Klauseln, wie folgt, vorliegt

$$F = \bigwedge_{i=1}^{m} C_i = C_1 \wedge C_2 \wedge \ldots \wedge C_m\,,$$

wobei C_i Klauseln genannt werden. Der Wert der leeren Funktion in CNF ist daher gleich $F(X) = \bigwedge_{i \in \emptyset} C_i = 1$. Eine Klausel C ist die Disjunktion von Literalen, d.h.

$$C \;=\; \bigvee_{i=1}^{k} \ell_i \;=\; \ell_1 \vee \ell_2 \vee \ldots \vee \ell_k\,.$$

Hierbei ist der Wert der leeren Klausen demnach 0. Die Literale ℓ sind positive oder negierte Variablen:

$$\ell \;=\; (x_i)^b\,,$$

wobei $i \in \{1, \ldots, n\}$ und $b \in \{0, 1\}$. Hierbei bedeutet für eine binäre Variable x

$$x^0 := \neg x = \overline{x}\,, \quad \text{und} \quad x^1 := x\,.$$

```
Proc DPLL (φ ∈ CNF)
    begin
        if φ ist leer then return 1
        else if leere Klausel in φ vorhanden then return 0
        else if es existiert Klausel mit nur einem Literal ℓ = x_i^b then
            DPLL(φ, wobei x_i ← b)
        else if Literal ℓ = (x_i)^b kommt nicht negiert vor then
            DPLL(φ, wobei x_i ← b)
        else
            wähle x_i aus φ
            if DPLL(φ mit x_i = 1) then return 1
            else return DPLL(φ mit. x_i = 0)
            fi
        fi
    end
```

Abbildung 2.1: Der Davis-Putnam-Algorithmus nach Davis, Loveland und Longman

CNF-Formeln genügen Um einzusehen, dass die Erfüllbarkeit von CNF-Formeln genauso schwierig ist, wie die Erfüllbarkeit beliebiger Formeln werden auf eine gegebene beliebige Formel folgende Transformationen ausgführt.

1. Falls $F(X) = \neg F_1(X)$, wenden wir die DeMorgansche Regeln auf $F_1(X)$ an.
2. Falls $F(X) = F_1(X) \wedge F_2(X) \wedge \ldots \wedge F_m(X)$, wird dieser Prozess auf den Teilgliedern $F_i(X)$ rekursiv angewendet.
3. Falls $F(X) = F_1(X) \vee F_2(X) \vee \ldots \vee F_m(X)$ und einer der Teilterme $F_i(X)$ kein Literal ist, werden die Hilfsvariablen $h_1, \ldots h_m$ eingeführt und die Formel ersetzt durch

$$(F_1(X) \vee h_1) \wedge (F_2(X) \vee h_2) \wedge \ldots \wedge (F_m(X) \vee h_m) \wedge (\overline{h_1} \vee \overline{h_2} \vee \ldots \vee \overline{h_m}) \,.$$

Tatsächlich kann man sich sogar auf 3-CNF beschränken, d.h. dass in jeder Klausel nur drei Literale vorkommen. Die Transformation einer k-CNF auf eine 3-CNF erfolgt wieder durch die Hinzunahme von Hilfsvariaben. Für jeder Klausel C_i wird eine disjunkte Menge von k Hilfsvariablen genommen. Für die Klausel

$$C = (\ell_1 \vee \ell_2 \vee \ldots \vee \ell_k)$$

ersetzen wir dann

$$(\ell_1 \vee h_1) \wedge (\overline{h_2} \vee \ell_2 \vee h_3) \wedge (\overline{h_3} \vee \ell_3 \vee h_4) \wedge \ldots \wedge (\overline{h_k} \vee \ell_k) \,.$$

Das entsprechende Entscheidungsproblem wird 3-SAT genannt. Eine weitere Reduktion auf 2-SAT ist nicht bekannt. Nicht zuletzt deswegen, weil 2-SAT in polynomieller Zeit lösbar ist und aus solch einer Transformation $\mathcal{P} = \mathcal{NP}$ folgen würde.

2.3.2 Der DPLL-Algorithmus

Im Jahr 1962 stellten Davis, Logemann und Loveland [DLL62] eine Anpassung des allgemeineren Davis-Putnam-Algorithmus auf das SAT-Problem von CNF-Formeln vor.

Obgleich dieser Algorithmus schon ziemlich lange bekannt ist, liegt er vielen heute eingesetzten Algorithmen zugrunde. Eine besondere Eigenschaft hierbei ist, dass sobald er eien Ausgabe trifft, er entweder konstruktiv eine Belegung der Formel ausgeben kann oder die Nichterfüllbarkeit beweisen kann.

Die Regel, die testet, ob eine Klausel nur aus einem Literal besteht wird hierbei **unit-clause**-Regel genannt, während die Regel, die nach dem gleichartigen Vorkommen eines Literals fragt, **pure-literal**-Regel heißt.

Die rekursiven Aufrufe des Algorithmus werden **DP-calls** genannt. Bei der Wahl der Variablen, die hierbei getestet wird, gibt es verschiedene Strategie. Die als momentan erfolgreichste anerkannte Regel ist die sogenannt MOMS-Technik (Maximum number of Occurrences in Minimum Size clauses) [DLL62, Gol79, Pre93]

2.4 Ist Satisfiability einfach?

Eine erste Average-Case-Analyse von SAT wurde von Goldberg [Gol79] vorgenommen. Hierbei betrachtete er zufällige CNF-Formeln aus n Variablen und m Formeln. In jede Klausel wird hierbei mit Wahrscheinlichkeit p eine der $2n$ Literale gewürfelt (als unabhängige Zufallsereignisse). Über die erwartete Laufzeit kann man folgenden Satz beweisen. Die Zufallsvariable, welche die Erzeugnis solcher CNF-Formeln beschreibt, wird hier mit $G_{p,m,n}$ bezeichnet.

Theorem 5 *[Gol79] Für SAT mit Formeln aus $G_{p,m,n}$ hat DPLL erwartete Laufzeit $\mathcal{O}(m \cdot n^2)$.*

Dieses Ergebnis legt nahe, dass das Satisfiability zwar im Worst-Case schwierig sein kann, dass es aber im Average eine machbare Aufgabe darstellt. Wir werden auf diese Analyse später zurückkommen.

Wir werden diesen Satz später beweisen. Als Intuition sei schon so viel verraten. Bei einer Fallunterscheidung betrachtet man zuerst den Fall, dass p groß ist. In diesem Fall kommen in einer Klausel sowohl ein Literal einfach als auch negiert vor, was die Klausel trivialerweise auf 1 setzt. Somit kann diese weggelassen werden. Auf diese Weise verschwinden ziemlich alle Klauseln und die Verblienbenen kann man dann effizient lösen.

Ist dagegen p sehr klein, so wird der Fall, dass ein Literal nur einfach vorkommt, sehr wahrscheinlich. Damit kann das Literal ersetzt werden und die Formel vereinfacht sich wiederum.

Dieses Ergebnis legt nahe, dass das Erfüllbarkeitsproblem zwar im Worst-case schwierig ist, aber im Average eine machbare Aufgabe darstellt. In einigen Forschergemeinden machte sich deswegen die Überzeugung breit, dass die Schwierigkeit des Erfüllbarkeitsproblems in der Praxis kein Problem darstellt. Um diesen Irrglauben im Bereich der künstlichen Intelligenz entgegenzuwirken veröffentlichteh Selman, Mitchell und Levesque eine richtungsweisende Arbeit in [MSL92].

Als erstes beschreiben sie eine andere Art von Wahrscheinlichkeitsverteilungen. Hierzu betrachten wir die folgenden drei Parameter n, k und m:

- n beschreibt die Anzahl von Variablen, die in der Booleschen Formel vorkommen.
- k beschreibt die Anzahl der Literale in einer Klausel der konjunktiven Normalform.
- m beschreibt die Anzahl der Klauseln der KNF. In jede dieser Klauseln werden jetzt k-mal hintereinander eines aus $2n$ Literale ausgewählt.

Der Parameter k wird festgehalten, während die Formeln für wachsende Variablenzahl n betrachtet werden. Wir bezeichnen mit k-CNF$_{n,c\cdot n}$ die Zufallsveriable, die dieses Zufallsexperiment in Abhängigkeit von k, n und c erzeut.

Hierbei wird der Quotient $c = \frac{m}{n}$ fest gewählt. Nun beschreiben wir die Komplexität einer solch erzeugen CNF-Formel durch die Anzahl der DP-Calls, die zur Lösung einer solchen Funktion notwendig. Abbildung 2.2 beschreibt das Ergebnis dieser experimentellen Untersuchung.

Man erkennt das für eine mittlere Wahl von c sich ein Maximum in der so definierten Komplexität der Booleschen Formeln ergibt. Worauf läßt sich das zurückführen? Hierfür betrachtet man zusätzlich das Ergebnis der DPLL-Algorithmen und trägt den Anteil erfüllbarer Formeln im Vergleich zu allen gegebenen Formeln an. Abbildung 2.3 zeigt, dass dieser Anteil bei $\frac{1}{2}$ liegt, wenn das Maximum der Laufzeit von DPLL erreicht wird. Daneben scheint die Stelle weitgehend unabhängig von der Größe n der Formel zu sein. Diese empirischen Betrachtungen veranlassten die Autoren zu den folgenden beiden Vermutungen.

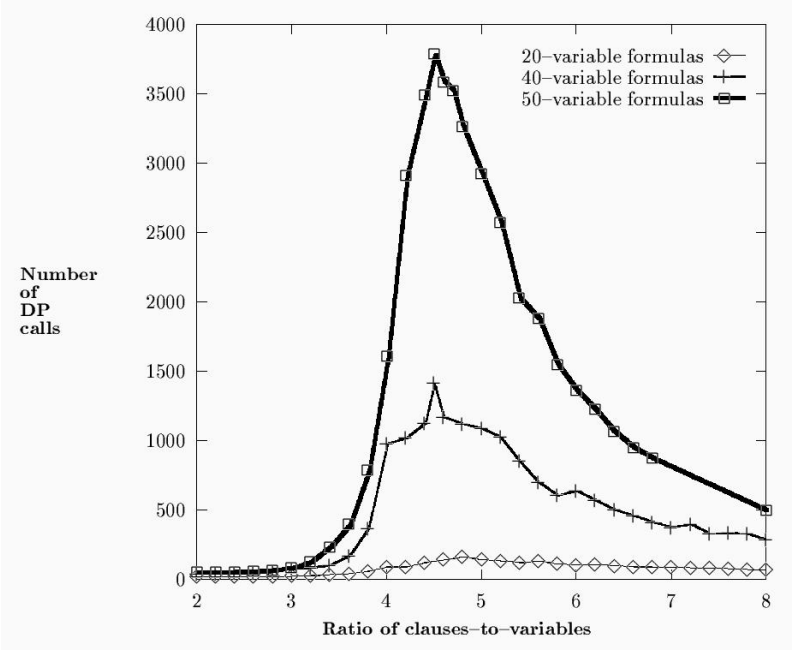


Abbildung 2.2: Durchschnittliche Anzahl rekursiver DP-Aufrufe für zufällige 3-SAT-Formen als Funktion über dem Verhältnis von Klauseln zu Variablen. [MSL92]

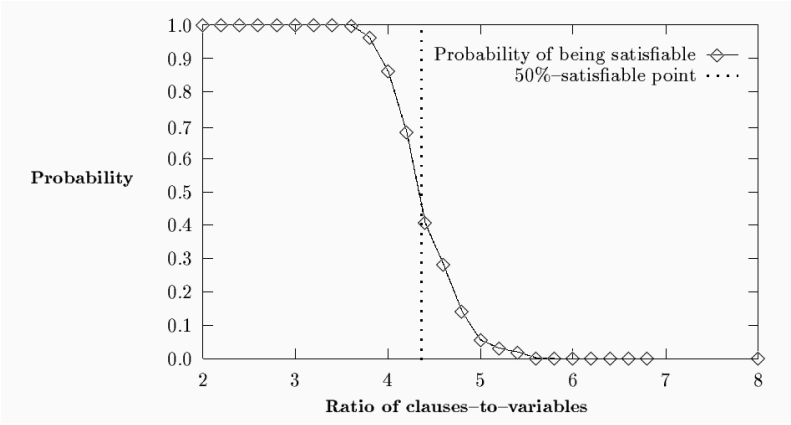


Abbildung 2.3: Wahrscheinlichkeit der Erfüllbarkeit Formel mit 50 Variablen als Funktion der Verhältnis von Klauseln zu Variablen[MSL92]

Schwellwertvermutung für k-SAT Sei m die Anzahl der Klauseln und n die Anzahl der Variablen Booleschen Formel in konjunktiver Normalform.

1. Es gibt ein Verhältnis $c_k^* = \frac{m}{n}$, so dass

$$\lim_{n\to\infty} \mathrm{P}[\phi \in (k, n, c_k^* n)\text{-CNF }, \text{ und } \phi \text{ ist erfüllbar}] = \frac{1}{2} .$$

2. Für dieses c_k^* ist $k - SAT$ prinzipiell im Average schwierig.

Seitdem diese Vermutungen aufgestellt worden sind, wurden gewisse Fortschritte erzielt.

Von Chvatal, Reed und gleichzeitig unabhängig von Goerdt wurde 1992 [CR92, Goe92] bewiesen, dass die Schwellwertvermutung für 2-SAT gilt. Hierfür ist $c_2^* = 1$. Natürlich erhält man in diesem Fall keine schwierigen 2-SAT-Instanzen, da solche Formeln in polynomieller Zeit insbesondere durch den DPLL-Algorithmus gelöst werden können.

Die besten Schranken für 3-SAT sind momentan

$$3,003 \ < \ c_3^* \ < \ 4,598$$

Die untere Schranke wurde 1992 von Frieze und Suen [FS96] bewiesen[1], die obere von Kirousis, Kranakis und Krizanc [KKK96] nachgeweisen. Davor wurden die Schranken sukzessive von 5,19 von verschieden Autoren über 4,78 [KMPS94] und 4,64 [DB97] auf diesen Wert reduziert.

Der beste bekannte Schätzwert ist

$$c_3^* \sim 4,24$$

Dieser wurde 1994 [SKC94] für $n = 2000$ bestimmt. Der hierbei verwendete Algorithmus ist eine Modifikation des Davis-Putnam-Algorithmus. Diesen hat man mit einer Eingabe-Wahrscheinlichkeitsverteilung getestet, die als besonders schwierig vermutet wird, (nämlich Instanzen am Schwellwert). Tatsächlich hat der Algorithmu 50% der Eingaben auf der damaligen Personal-Computer-Generation höchstens eine Stunde benötigt hat.

Die Frage inwieweit solche Wahrscheinlichkeitsvertilungen für jeden denkbaren Algorithmus schwierige Eingaben erzeugen, ist dagegen noch weit offen. Wir wenden uns im nächsten Kapitel der Frage zu, wie überhaupt schwierige Wahrscheinlichkeitsverteilungen für NP-vollständige Probleme identifiziert und nachgewiesen werden können.

[1] Der Journal-Artikel erschien erst später

Kapitel 3

Average-Komplexitätstheorie

Wir haben gesehen, dass es $\mathcal{NP}$-vollständige Probleme gibt, die im Erwartungswert effizient sind. Daneben scheint es auch eine Frage der Wahrscheinlichkeitsverteilung zu sein, wann ein $\mathcal{NP}$-Problem im Average schwierig wird. Wir werden in diesem Kapitel eine Theorie vorstellen, die versucht den Begriff der $\mathcal{NP}$-Vollständigkeit vom Worst-Case in Average zu übertragen. Hierfür gab es verschiedene Ansätze. Wir stellen hier den ersten Ansatz vor.

3.1 Levins verteiltes $\mathcal{NP}$ im Average

Wir gliedern diesen Abschnitt in vier fundamentale Fragestellungen:

- Was heißt effizient im Average?
- Was sind $\mathcal{NP}$-Probleme?
- Was ist eine Average-Reduktion?
- Welche Probleme sind Average-$\mathcal{NP}$-vollständig?

3.1.1 Levins Begriff von Effizienz

Ein Grund, warum die Entwicklung der Average-Komplexitätstheorie relativ lange dauerte, ist, dass die herkömmlichen Begriffe aus der Wahrscheinlichkeitstheorie nicht elementaren komplexitätstheoretischen Eigenschaften genügen.

In einem ersten Versuch untersuchen wir den Erwartungswert des Zeitverhaltens eines Algorithmus. Mit $\text{time}_M(x)$ bezeichnen wir die Laufzeit der Maschine M auf Eingabe $x \in \Sigma^*$.

Dann hat die Maschine M **erwartet polynomielle Laufzeit**, wenn

$$\exists c, c' \in \mathbb{N}\, \forall n \in \mathbb{N}\, \mathbf{E}_{\mu_n}\left[\text{time}_M(x)\right] \;\leq\; c' \cdot n^c \;.$$

Hierbei bezeichnet μ_n eine Wahrscheinlichkeitsverteilung über Σ^n, wobei wir mit $\mu'_n(x) = \mathrm{P}_{\mu_n}[x]$ die Wahrscheinlichkeit beschreiben, dass die Zeichenkette x erscheint. Damit ist die obere Bedingung äquivalent zu der Folgenden.

$$\exists c, c' \in \mathbb{N}\, \forall n \in \mathbb{N}\; \sum_{x \in \Sigma^*} \text{time}_M(x)\mu'_n(x) \;\leq\; c' \cdot n^c \;.$$

Leider gibt es eine Reihe von unschönen Eigenschaften dieses Begriffs:

- Keine Abgeschlossenheit gegenüber Polynomialzeittransformationen
- Keine Durchschnittsbildung bei unären Mengen
- Abhängigkeit von der Kodierung der Eingabe

Polynomialzeittransformationen Wir werden die ersten beiden dieser Probleme an Beispielen deutlich machen.

Hierfür betrachten wir ein 2-Band-TM M_2. Solche eine 2-Band-TM kann von einer 1-Band-TM M_1 in quadratischer Zeit simuliert werden, d.h.

$$\text{time}_{M_1}(x) \ \leq \ (\text{time}_{M_2}(x))^2 \ .$$

Betrachten wir folgendes Zeitverhalten der 2-Band-TM M_2:

$$\text{time}_{M_2}(x) \ = \ \begin{cases} 2^n \ , & \text{falls } x = 0^n = \underbrace{0 \ldots 0}_{n}, \\ n \ , & \text{sonst .} \end{cases}$$

Ferner betrachten wir das binäre Eingabealphabet $\Sigma = \{0, 1\}$ und die uniforme Wahrscheinlichkeitsverteilung $\mu'_n(x) = \frac{1}{|\Sigma|^n}$ also $\mu'_n(x) = \frac{1}{|\Sigma|^n} = 2^{-n}$. Damit ist die erwartete Laufzeit

$$\begin{aligned} \mathbf{E}_{\mu_n}\left[\text{time}_{M_2}(x)\right] &= \sum_{x \in \Sigma^n} \text{time}_{M_2}(x)\mu_n(x) \\ &= \sum_{x \in \Sigma^n} \text{time}_{M_2}(x)\frac{1}{2^n} \\ &= \sum_{x \in \Sigma^n \setminus \{0^n\}} n\frac{1}{2^n} + 2^n \cdot \frac{1}{2^n} \\ &= \left(1 - \frac{1}{2^n}\right) n + 1 \ \leq \ n + 1 \ . \end{aligned}$$

Also ist M_2 sogar erwartet linear zeitbeschränkt. Wir wissen, dass M_1 die Maschine M_2 in quadratischer Zeit simuliert. Damit erhalten wir folgende erwartete Laufzeit.

$$\begin{aligned} \mathbf{E}_{\mu_n}\left[\text{time}_{M_1}(x)\right] &= \mathbf{E}_{\mu_n}\left[(\text{time}_{M_2}(x))^2\right] \\ &= \sum_{x \in \Sigma^n} \text{time}_{M_2}(x)^2\frac{1}{2^n} \\ &= \sum_{x \in \Sigma^n \setminus \{0^n\}} n^2\frac{1}{2^n} + (2^n)^2 \cdot \frac{1}{2^n} \\ &= \left(1 - \frac{1}{2^n}\right) n^2 + 2^n \ \geq \ 2^n \end{aligned}$$

Also benötigt M_1 exponentielle Zeit, obwohl M_1 eine quadratische Simulation eines erwartet linear zeitbeschränkten Algorithmus darstellt.

Ein analoges Beispiel lässt sich die Kodierung eines Graphen als Adjazenzliste oder Adjazenzmatrix. Auch hier kann der selbe Algorithmus entweder erwartet linear oder erwartet exponentielle Zeit benötigen.

Unäre Mengen sind Mengen definiert über einem Alphabet $\Sigma = \{1\}$. Eine Menge L ist unär oder **tally**, gdw. $L \subseteq \{1^n \mid n \in \mathbb{N}\}$.

Obgleich solche Tally-Mengen in der Praxis kaum eine Rolle spielen, haben Solche Menge eine wichtige Bedeutung im Bereich der Komplexitätstheorie. Zum Beispiel gilt folgendes Theorem nach Berman 1978, zitiert in [For79]

Theorem 6 *Falls es eine $\mathcal{NP}$-vollständige Tally-Menge gibt, dann ist $\mathcal{NP} = \mathcal{P}$.*

Wenn eine Maschine eine Tally-Menge untersucht gibt es in jeder Eingabelänge nur eine Eingabe. Damit muss also $\mu_n(x) = 1$ sein und damit ist der Erwartungswert $\mathbf{E}_{\mu_n}\left[\text{time}_M(x)\right] = \text{time}_M(x)$ für alle $x \in 1^n$. Damit ergibt sich hieraus direkt eine Worst-case-Schranke und eine Mittelung über verschiedene Laufzeit findet nicht statt.

Levins Lösung Um das zuletzt aufgezeigte Problem zu lösen mitteln wir über alle Eingabelängen. D.h. wir betrachten eine Wahrscheinlichkeitsverteilung über alle Zeichenkette Σ^*, wie zum Beispiel die **uniforme Wahrscheinlichkeitsverteilung** μ_{uni}:

$$\mu'_{\text{uni}}(x) = \frac{1}{|x|^2} \cdot \frac{6}{\pi^2} \cdot \frac{1}{|\Sigma|^n} \ .$$

Die erwartete Laufzeit bezüglich einer solchen Wahrscheinlichkeitsverteilung

$$\mathbf{E}_\mu \left[\text{time}_M(x)\right] \ = \ \sum_{x \in \Sigma^*} \mu'(x) \text{time}_M(x) \ .$$

Dieser Erwartungswert konvergiert schon für quadratische Laufzeitfunktion nicht. Um also eine erwartete Laufzeitbeschränkung durch eine Funktion $T : \mathbb{N} \to \mathbb{N}$ zu beschreiben, kann man folgenden Term verwenden.

$$\mathbf{E}_\mu \left[\frac{\text{time}_M(x)}{T(|x|)} \right] \ = \ \sum_{x \in \Sigma^*} \frac{\text{time}_M(x)}{T(|x|)} \mu'(x) \ \leq \ 1.$$

Dieser Term wird wahr, wenn M erwartet T-zeitbeschränkt wird. Damit wäre das Problem der Mittelung über verschiedene Eingabelängen gelöst. Jedoch bleibt die mangelnde Abgeschlossenheit gegenüber polynomiellen Transformationen bestehen. Daher wandte Leonid Levin die Funktion T^{-1} sowohl auf den Zähler und den Nenner des Bruchs $\frac{\text{time}_M(x)}{T(|x|)}$ an und erhielt so $\frac{T^{-1}(\text{time}_M(x))}{|x|}$.

Wir definieren daher.

Definition 2 *Für $f : \Sigma^* \to \mathbb{N}$, einer Wahrscheinlichkeitsverteilung μ und $T : \mathbb{N} \to \mathbb{N}$ monoton zunehmend ist das Paar (f, μ) im Levin-Average in Zeit T genau dann, wenn:*

$$(f, \mu) \in \text{Av}(T) \ :\Longleftrightarrow \mathbf{E}_\mu \left[\frac{T^{-1}(f(x))}{|x|} \right] \ \leq \ 1 \ .$$

Besonders interessiert uns der Levinsche Begriff von **polynomiell im Average**, $\text{Av}(\text{POL})$:

$$(f, \mu) \in \text{Av}(\text{POL}) \iff \exists c, k \in \mathbb{N} \ \sum_{x \in \Sigma^*} \frac{f(x)^{1/k}}{|x|} \mu'(x) \ \leq \ c \ .$$

Damit ist eine TM M im Average polynomiell zeitbeschränkt bezüglich μ, wenn

$$(f, \mu) \in \text{Av}(\text{POL}) \iff \exists c, k \in \mathbb{N} \ \sum_{x \in \Sigma^*} \frac{\left(\text{time}_M(x)\right)^{1/k}}{|x|} \mu'(x) \ \leq \ c \ .$$

Wichtige Eigenschaften dieses Average-Begriffs sind in dem folgenden Lemma zusammengefasst:

Lemma 1
1. $\forall x : \ f(x) \leq T(|x|) \implies \forall \mu \ : \ (f, \mu) \in \text{Av}(T)$.
2. $\forall n : \ \mathbf{E}_{\mu(\Sigma^n)} \left[f(x)\right] \leq T(|x|) \implies (f, \mu) \in \text{Av}T(2\mathcal{N})$.
3. $(f, \mu), (g, \mu) \in \text{Av}(\text{POL}) \implies \{(f + g, \mu), (f \cdot g, \mu), (f - g, \mu)\} \in \text{Av}(\text{POL})$

Die Verifikation dieses Lemmas sei den Besuchern der Veranstaltung als Übung herzlichst empfohlen. Eine ganz wichtige Eigenschaft von Levins Average-Maß beschreibt das folgende Lemma.

Lemma 2 *Seien $f, T : \mathbb{N} \to \mathbb{N}$ monoton zunehmenden Funktionen. Dann gilt für alle Laufzeitfunktionen $g : \Sigma^* \to \mathbb{N}$:*

$$(g, \mu) \in \text{Av}(T) \ \Rightarrow \ (g \circ f, \mu) \in \text{Av}(g \circ T) \ .$$

Wiederum sei die Verifikation dieses Lemmas dem geneigten Leser als Übung empfohlen.

3.1.2 Was sind geeignete NP-Probleme für Average-Komplexitätsklassen

Vorab: Wir werden vom Begriff $\mathcal{NP}$ nicht abweichen. Wir beschäftigung uns mit $\mathcal{NP}$-Entscheidungsproblemen im originären Sinne[1]. Die Frage, die sich vielmehr stellt, ist welche Wahrscheinlichkeitsverteilungen soll man in Kombination mit $\mathcal{NP}$-vollständigen Problemen betrachten. Li und Vitanye [LV92] beschreiben eine Wahrscheinlichkeitsverteilung, für die jedes Problem sein Worst-case Verhalten auch im Erwartungswert annimmt. Es zeigt sich aber, dass diese Wahrscheinlichkeitsverteilung nicht Turing-berechenbar ist. Solche Wahrscheinlichkeitsverteilung möchte man aber gerne im vorhinein ausschließen.

Wir betrachten daher Wahrscheinlichkeitsverteilungen, die in polynomieller Zeit berechenbar sind. Hierbei ergibt sich als nächste Frage, was dies ist. Man muss sich vergegenwärtigen, dass die Wahrscheinlichkeiten reelle Zahlen darstellen. Damit müssen wir einen Begriff der effizienten Berechenbarkeit von reellen Zahlen zugrunde legen. Levin führte hierzu folgenden Begriff ein:

Definition 3 *Eine Wahrscheinlichkeitsverteilung μ ist polynomialzeitberechenbar, d.h. $\mu \in$ POL-computable, gdw. es eine Turingmaschine gibt, die auf Eingabe $x \in \Sigma^*$ und $m \in \mathbb{N}$ folgende Eigenschaften hat.*

1. *M produziert die Ausgabe $M(\#x\#1^m\#)$ in Zeit $\mathcal{O}(m^c|x|^c)$ für ein c.*
2. *$M(x)$ approximiert $\mu(x) = \sum_{z \leq x} \mu'(z) = \sum_{z \leq x} \mathrm{P}_\mu[z]$ bis auf die m-te Binärstelle:*

$$|M(x) - \mu(x)| \ \leq \ 2^{-m} \ .$$

Um Untersuchungen im Bereich der Average-Komplexitätstheorie vorzunehmen, müssen wir also Paare von Problemen und Wahrscheinlichkeitsproblemen betrachten. Wir nennen dies ein **(wahrscheinlichkeits) verteiltes Problem (distributed problem)**. Die zu $\mathcal{NP}$ alternative Klasse lautet daher verteiltes $\mathcal{NP}$, oder distributed $\mathcal{NP}$.

Definition 4

$$\mathit{Dist}\mathcal{NP} \ := \ \mathcal{NP} \times \text{POL-}\mathit{computable} \ .$$

An dieser Stelle sei darauf hingewiesen, dass die Wahrscheinlichkeitsverteilung mit $\mu(x)$ bezeichnet wird, während die konkrete Wahrscheinlichkeit $\mathrm{P}_\mu[x]$ mit $\mu'(x)$ bezeichnet wird. Der Grund ist, dass $\mu'(x)$ als Ableitung der Wahrscheinlichkeitsverteilung gesehen werden kann, denn es gilt für nicht-leere Zeichenketten x:

$$\mu'(x) = \mu(x) - \mu(x-1) \ .$$

Daraus folgt unmittelbar, dass bei einer Wahrscheinlichkeitsverteilung aus POL-computable auch die Wahrscheinlichkeit eines Eingabestrings in polynomieller Zeit berechnet werden kann (oder vielmehr approximiert werden kann).

3.1.3 Reduktionen von verteilten Problemen

In der Komplexitätstheorie verwendet man Reduktionen um zu zeigen, dass ein Problem L_1 einfacher ist als ein anderes L_2 oder zumindestens höchstens so schwierig. Darum verwendet man die Notation $L_1 \leq_{\text{pol}} L_2$, wobei man erlaubt, dass diese Ungleichung um bis zu einen polynomiellen Laufzeitunterschied verletzt werden kann.

Das Grunprinzip der Reduktion ist, dass L_1 höchstens so schwierig ist wie L_2, weil ich jede Instanz von L_1 mit einem Algorithmus der L_2 lösen kann auch berechnen kann. Hierbei ist zumeist eine Abbildung der Instanzen notwendig, die man mit einer Reduktionsfunktion R beschreiben kann.

Wir wiederholen also noch einmal den Polynomialzeitreduktionsbegriff aus der Worst-Case-Komplexitätstheorie:

Definition 5 Karp-Reduktion *Das Problem L_1 kann auf das Problem L_2 reduziert werden, wenn es eine Funktion $R : \Sigma^* \to \Sigma^*$ existiert, für die gilt*

1. *Gültigkeit: $x \in L_1 \iff R(x) \in L_2$*

[1] wobei zu erwähnen ist, dass es auch Untersuchungen von nichtdeterministischen Turingmaschinen und deren Average-Zeitverhalten existiert

2. *Effizienz: Es existiert eine polynomialzeitbeschränkte Turingmaschine M mit $M(x) = R(x)$.*

Wir werden diesen Begriff der Reduktion nun so übertragen, dass sie auch für verteilte Probleme angewendet werden kann. Hierfür muss man als erstes die Wahrscheinlichkeitsverteilung in den Reduktionsbegriff mit einbeziehen. Schließlich kann man schwierige Eingaben mit hohen Gewicht nicht auf Instanzen mit niedrigen Gewicht reduzieren. Ist nun dieses Problem effizient zu lösen, so kann man keine Rückschlüsse auf das ursprüngliche Problem machen. Daneben wollen wir auch der Reduktionsfunktion erlauben, nur im Average effizient zu sein. Diese Überlegungen führen zu der folgenden Definition gemäß den Ausführungen in [Gol97].

Definition 6 *Ein verteiltes Problem (L_1, μ_1) kann auf ein anderes verteiltes Problem im Average reduziert werden, d.h.*

$$(L_1, \mu_1) \leq_{\text{av-pol}} (L_2, \mu_2)$$

genau dann wenn eine Reduktionsfunktion $f : \Sigma^* \to \Sigma^*$ gibt mit den folgenden Eigenschaften.

1. *Gültigkeit:* $\forall x \in \Sigma^* \ : \ x \in L_1 \iff x \in L_2$
2. *Effizienz: Es existiert eine Turingmaschine M, so dass $M(x) = R(x)$ und*

$$(\text{time}_M, \mu_1) \in \text{Av(POL)} \ .$$

3. *Dominanz:* $\exists c > 0 \, \forall y \in \text{Bild}(R) \ : \ \mu_2'(y) \ \geq \ \sum_{x:R(x)=y} \frac{\mu_1'(x)}{|x|^c}$.

Bevor wir zeigen werden, dass diese Average-Reduktion überhaupt sinnvoll definiert ist, werden wir uns mit dem Begriff der Dominanz einer Wahrscheinlichkeitsverteilung beschäftigen.

Definition 7 *Die Wahrscheinlichkeitsverteilung μ_2 dominiert die Wahrscheinlichkeitsverteilung μ_1, gdw.*

$$\exists c > 0 \, \forall x \in \Sigma^* \ : \ \mu_2'(x) \ \geq \ \frac{\mu_1'(x)}{|x|^c} \ .$$

Wenn sich Wahrscheinlichkeitsverteilungen dominieren, so bleiben die zugehörigen verteilten Probleme in der gleichen Komplexitätsklasse.

Lemma 3 *Für Wahrscheinlichkeitsverteilungen μ_1, μ_2 und Funktion $f : \Sigma^* \to \mathbb{N}$ gilt:*

$$\mu_2 \text{ dominiert } \mu_1 \text{ und } (f, \mu_2) \in \text{Av(POL)} \implies (f, \mu_2) \in AvPOL \ .$$

Beweis: Es gilt also

$$\forall x \in \Sigma^* \ : \ \mu_2'(x) \ \geq \ \frac{\mu_1'(x)}{|x|^c}$$

und

$$\sum_x \frac{f(x)^{1/k}}{|x|} \mu_2'(x)) \leq c' \ ,$$

für geeignete Konstanten $c, c', k > 0$. Sei nun $\ell = (k+1)c$ und $X := \{x \in \Sigma^* \mid f(x)^{1/\ell} \cdot |x|^c \leq f(x)^{1/k}\}$. Aufgrund von X führen wir die folgende Fallunterscheidung durch.

1. $x \in X$:

$$\begin{aligned} \sum_{x \in X} \frac{f(x)^{1/\ell}}{|x|} \mu_1'(x) &= \sum_{x \in X} \underbrace{f(x)^{1/\ell} |x|^c}_{\leq f(x)^{1/k}} |x|^{-1} \mu_1'(x) \\ &\leq \sum_{x \in X} f(x)^{1/k} |x|^{-1} \mu_1'(x) \ \leq \ c' \ . \end{aligned}$$

2. $x \notin X$

$$\begin{aligned} f(x)^{1/k} < |x|^c f(x)^{1/\ell} &\implies f(x)^{1/k-1/\ell} < |x|^c \\ &\implies f(x)^{\ell-k} < |x|^{c\cdot\ell\cdot k} \\ &\implies f(x)^{1/\ell} < |x|^{c\cdot k/(\ell-k)} = |x| \; . \end{aligned}$$

Die Funktion f ist also durch $|x|^k$ beschränkt. Hieraus folgt aus der Tatsache, dass μ_1 eine Wahrscheinlichkeitsverteilung ist das Folgende.

$$\sum_{x\notin X} \frac{f(x)^{1/\ell}}{|x|}\mu_1'(x) \leq \sum_{x\notin X} \mu_1'(x) \leq 1 \; .$$

Wenn wir die beiden Fälle kombinieren, erhalten wir also

$$\sum_{x} \frac{f(x)^{1/\ell}}{|x|}\mu_1'(x) \leq c' + 1 \; .$$

Damit ist also $(f,\mu) \in \text{Av}(\text{POL})$. □

In der Average-Reduktion wurde die Dominanzeigenschaft kombiniert mit der durch die Reduktion induzierte Abbildung der Wahrscheinlichkeitsverteilungen. Wir werden im folgenden Lemma beschreiben, dass dadurch das Laufzeitverhalten im Average noch polynomiell bleibt.

Lemma 4 *Für Funktionen $g, f : \Sigma^* \to \Sigma^*$ und $T : \Sigma^* \to \mathbb{N}$ sei*

$$\mu_2'(y) = \sum_{x:f(x)=y} \mu_1'(x)$$

für alle $y \in \text{Bild}(f)$. Dann gilt:

1. *Falls $(|f|,\mu_1) \in \text{Av}(\text{POL})$ und $(T,\mu_2) \in AvPOL$, dann gilt $(T\circ f,\mu_1) \in \text{Av}(\text{POL})$.*
2. *Falls $(f,\mu_1) \in \text{Av}(\mathcal{FP})$ und $(g,\mu_2) \in \text{Av}(\mathcal{FP})$ dann folgt $(g\circ f,\mu_1) \in \text{Av}(\mathcal{FP})$.*

Beweis:

1. Seien $c', c'', k, m > 1$ geeignete Konstanten, so dass gilt

$$\sum_{y\in\Sigma^*} \frac{T(y)^{1/k}}{|y|} \;\leq\; c' \quad \text{und} \quad \sum_{x\in\Sigma^*} \frac{f(x)^{1/m}}{|x|} \;\leq\; c'' \; .$$

Dann folgt aus der ersten Ungleichung für jedes $m > 1$ durch geeignete Fallunterscheidung ($T(y) \overset{\leq}{>} |y|^m$):

$$\sum_{y\in\Sigma^*} \frac{T(y)^{1/(km)}}{|y|^{1/m}} \;\leq\; c' + 1 \; .$$

Jetzt schränken wir die Menge der Summanden ein auf den Wertebereich von f ein. Damit gilt also

$$\sum_{y\in\text{Bild}(f)} \frac{T(y)^{1/(km)}}{|y|^{1/m}} \;\leq\; c' + 1 \; .$$

Nun möchten wir y durch $f(x)$ substituieren. Hierbei müssen wir beachten, wie häufig es Werte x gibt, die auf ein y abgebildet werden. Von diesen müssen wir denselben Anteil besitzen wie vorher.

Dies wird durch die Gleichung $\mu_2'(y) = \sum_{x:f(x)=y} \mu_1'(x)$ gewährleistet. Damit ist also die folgende Ungleichung mit zuletzt genannten äquivalent.

$$\sum_y \sum_{x:f(x)=y} \frac{T(f(x))^{1/(km)}}{|f(x)|^{1/m}} \;\leq\; c+1$$

Ferner sei $h(x) := T(f(x))$. Hieraus folgt

$$\sum_x \frac{h(x)^{1/(km)}}{|f(x)|^{1/m}} \;\leq\; c'' \,.$$

Wir definieren nun die beiden folgenden Hilfsterme

$$\begin{aligned} \alpha(x) &:= \frac{h(x)^{\frac{1}{2k\cdot m}}}{f(x)^{\frac{1}{2m}}} \,, \\ \beta(x) &:= \frac{f(x)^{\frac{1}{2k\cdot m}}}{|x|^{\frac{1}{2}}} \,. \end{aligned}$$

Damit sind die Erwartungswerte $\mathbf{E}_{\mu_1}[\alpha^2]$ und $\mathbf{E}_{\mu_1}[\beta^2]$ endlich. Nun betrachten wir den Erwartungswert von $\alpha \cdot \beta$ und erhalten:

$$\begin{aligned} \mathbf{E}_{\mu_1}[\alpha\cdot\beta] &= \frac{1}{2}\mathbf{E}_{\mu_1}[(\alpha+\beta)^2 - \alpha^2 - \beta^2] \\ &\leq \frac{1}{2}\mathbf{E}_{\mu_1}[\max\{2\alpha, 2\beta\}^2 - \alpha^2 - \beta^2] \\ &\leq \frac{1}{2}\mathbf{E}_{\mu_1}[4\alpha^2 + 4\beta^2 - \alpha^2 - \beta^2] \\ &= \frac{3}{2}\mathbf{E}_{\mu_1}[\alpha^2] + \frac{3}{2}\mathbf{E}_{\mu_1}[\beta^2] \;\leq\; \frac{3}{2}(c' + c'' + 1) \,. \end{aligned}$$

Nun ist aber

$$\alpha(x)\beta(x) = \frac{h(x)^{\frac{1}{2k\cdot m}}}{f(x)^{\frac{1}{2m}}} \cdot \frac{f(x)^{\frac{1}{2k\cdot m}}}{|x|^{\frac{1}{2}}} = \frac{h(x)^{\frac{1}{2k\cdot m}}}{|x|^{\frac{1}{2}}} \leq \frac{h(x)^{\frac{1}{2k\cdot m}}}{|x|} \,.$$

2. Nun sei M_g eine Turing-Maschine für g mit $(\text{time}_{M_g}, \mu_2) \in \text{Av}(\text{POL})$ und M_f eine Turing-Maschine für f mit $(\text{time}_{M_f}, \mu_1) \in \text{Av}(\text{POL})$.

   Eine Turing-Maschine M bereche nun $g(f(x))$ wie folgt.

   - Zuerst berechnet man $y \leftarrow M_f(x)$.
   - Dann wird auf y die Maschine M_g gestartet und das Ergebnis ausgegeben.

   Damit gilt für die Laufzeit von M:

   $$\text{time}_M(x) = \text{time}_{M_f}(x) + \text{time}_{M_g}(f(x)) \,.$$

   Wir beweisen für jeden der Summanden separat, dass sie im Average polynomiell sind und wenden dann Lemma 1 an. Für den ersten Summanden folgt die effizient im Durchschnitt aus der Annahme. Für den zweiten Summanden setzen wir $T := \text{time}_{M_g}$ und wenden wir den ersten Teil dieses Lemmas an, wobei wir beachten, dass $|f(x)| \leq \text{time}_{M_f}(x)$ und damit $(|f(x)|, \mu_1) \in AvPOL$.

□

Durch die Kombination dieser beiden Lemmata folgt sofort die Gültigkeit der Average-Reduktion, die durch das folgende Theorem beschrieben wird.

Theorem 7 *Für verteilte Probleme $(L_1, \mu_1), (L_2, \mu_2)$ gilt*

$$(L_1, \mu_1) \leq_{\text{av-pol}} (L_2, \mu_2) \;\wedge\; (L_2, \mu_2) \in \text{Av-}\mathcal{P} \;\Longrightarrow\; (L_2, \mu_2) \in \text{Av-}\mathcal{P} \,.$$

```
Proc M_2(x, 1^k)
    begin
        i ← 1
        while M_1(x+1, 1^i) − M_1(x, 1^i) > 4 · 2^i ∧
              M_1(x+1, 1^i) − M_1(x, 1^i) > 4 · 2^i) ∧
              i ≤ k do
              i ← i + 1
        od
        if i < k then
            return ε2^{−i} ⌊2^i ε^{−1} M_1(x, 1^{i−log ε^{−1}}) + 1/2⌋
        else
            return M_1(x, 2^i)
        fi
    end
```

Abbildung 3.1: Mit diesem Programm wird eine rationale Approximation einer polynomialzeitberechenbaren Wahrscheinlichkeitsverteilung berechnet

Beweis: folgt aus Lemma 3 und Lemma 4. □

Als nächstes gilt es also zu zeigen, dass es ein verteilte Dist-NP-Problem gibt, welches vollständig für diese verteilte Problemklasse ist. Bevor wir dazu kommen, müssen wir folgendes technische Lemma beweisen. Es besagt, dass man statt reellwertiger Wahrscheinlichkeitsverteilungen zu betrachten, sich auf rationale Zahlen mit einer Zweierpotenz im Nenner einschränken kann.

Lemma 5 *Für alle $\epsilon > 0$, für alle $\mu_1 \in$ POL-computable gibt es eine Wahrscheinlichkeitsverteilung $\mu_2 \in$ POL-computable, so dass*

1. $|\mu_1'(x) - \mu_2'(x)| \leq \epsilon \mu_1'(x)$
   (was gleichbedeutend ist mit $\mu_1'(x) = \mu_2'(x)(1 \pm \epsilon)$)

2. $\exists a, b \in \mathbb{N} \; : \; \mu_2'(x) = \frac{a}{2^b}$

Beweis: In Abbildung 3.1 ist ein Algorithmus dargestellt, der die Wahrscheinlichkeitsverteilung μ_2' berechnet. Hierbei sei $M_1(x, 1^k)$ eine Turingmaschine, die gemäß unserer Definition in polynomieller Zeit die Wahrscheinlichkeitsverteilung μ_1 berechnet.

Wir werden nachweisen, dass die dadurch definerte Wahrscheinlichkeitsverteilung die geforderten Eigenschaften besitzt.

1. Laufzeit: Im schlimmsten Fall werden k Schleifendurchläufe ausgeführt. Dies hat $3k$ Berechnungen von M_1 mit zunehmendem zweiten Parameter $i \leq k$ zur Folge. Danach wird M_1 noch mit Paramter x und $1^{k - \log \epsilon}$ aufgerufen, wobei $\epsilon > 0$ eine Konstante darstellt. Es ist klar, dass damit M_2 noch polynomielle Laufzeit aufweist.

2. Zur Approximationseigenschaft betrachten wir das i_x für das die Schleife termininert:

   $$i_x := \min\{i \mid M_1(x+1, 1^i) - M(x, 1^i) > 4 \cdot 2^i \; \wedge \; M_2(x, 1^i) - M(x-1, 1^i) > 4 \cdot 2^i\}$$

   Damit gilt

   $$\mu_1'(x) \geq 2 \cdot 2^{i_x}$$

   Hieraus folgt die gesuchte Eigenschaft

□

3.1.4 Ein verteiltes $\mathcal{NP}$-vollständiges Problem

Wir betrachten ein sogenannte generisches vollständiges Problem. Solche Probleme kann man zumeist am einfachsten als vollständig charakterisieren, da sie als Instanz aus einer Eingabe, einer Gödelnummer einer zur Komplexitätsklasse angepassten Aufzählung und einer Laufzeitschranke bestehen.

Das generische Problem zu $\mathcal{NP}$ ist das **nichtdeterministische beschränke Halteproblem** NBH.

Für eine Aufzählung M_i von nichtdeterministischen Turing-Maschinen ist es folgendermaßen definiert.

$$\text{NBH} \ := \ \{(w01^i0^n \mid \text{für NTM } M_i \ : \ M_i \text{ akzeptiert } w \text{ und time}_{M_i}(w) \leq n\} \ .$$

Als passende Wahrscheinlichkeitsverteilung betrachten wir die uniforme Wahrscheinlichkeitsverteilung

$$\mu_{\text{NBH}}(w01^i0^n) \ = \ \frac{6}{\pi^2} \cdot \frac{1}{2^{|w|}} \cdot \frac{1}{i^2 \cdot n^2} \ .$$

In dieser Kombination erhalten wir das erste Dist$\mathcal{NP}$-vollständige Problem.

Theorem 8 $(\text{NBH}, \mu_{\text{NBH}})$ *ist Dist$\mathcal{NP}$-vollständig.*

Beweis:

Die Vollständigkeit dieses verteilten Problems setzt zwei Bedingungen voraus:

1. $(\text{NBH}, \mu_{\text{NBH}}) \in \text{Dist}\mathcal{NP}$,
2. $\forall (L, \mu) \in \text{Dist}\mathcal{NP} \ : \ (L, \mu) \leq_{\text{av-pol}} (\text{NBH}, \mu_{\text{NBH}})$.

Für die erste Bedingung muss man zum einen überprüfen, ob $\text{NBH} \in \mathcal{NP}$ ist. Dies ist der Fall, da es eine universelle nichtdeterministische Turing-Maschine für die Berechnung von $M_i(w)$ gibt, mit Laufzeit $\mathcal{O}(\text{POL}(i)\text{time}_{M_i}(w)) \subseteq \text{POL}$. Es bleibt noch zu zeigen, dass $\mu_{\text{NBH}} \in$ POL-computable. Hierzu beobachtet man zuerst, dass alle Eingaben gleicher Länge gleiches Gewicht erhalten. Es verbleibt das Problem die Summe der Wahrscheinlichkeiten kürzerer Zeichenketten zu berechnen. Dies kann in polynomieller Zeit durch einfache Addition aller Möglichkeiten von i, n und $|w|$ geschehen.

Es verbleibt das Problem, die Härte des Problems zu zeigen. Wir betrachten daher ein verteiltes Problem $(L, \mu) \in \text{Dist}\mathcal{NP}$. Sei M_i eine nichtdeterministische Turing-Maschine für L mit Laufzeit $\text{time}_{M_i}(x) \leq |x|^c$ für eine geeignete Konstante c.

Für diese Sprache werde für ein geeignet gewähltes j folgende Reduktionsfunktion gewählt:

$$R(x) \ := \ \begin{cases} 0x01^j0^{|x|^c} \ , & \text{falls } \mu'(x) < 2^{-|x|} \ , \\ 0x'01^j0^{|x|^c} \ , & \text{falls } \mu'(x) < 2^{-|x|}, \text{ wobei (*) } r = 2^{-\lceil \log \mu'(x) \rceil} \\ & \text{und } x' = \lfloor \mu(x) \cdot r + \frac{1}{2} \rfloor . \end{cases}$$

Die Wahl von x' in Bedingung (*) kann man auch ersetzen durch die kürzeste Binärzahl x', so dass die Ungleichung

$$\mu(x) \leq 0, x'1 < \mu(x+1) \text{(**)}$$

gilt. Hier bezeichnet $x+1$ den lexikographischen Nachfolger von x und $0, x'1$ beschreibt einen binäre Darstellung einer Zahl aus $[0, 1]$ durch die Ziffern der Binärzahl x'.

Lemma 6 *Sowohl aus (*) als auch aus (**) folgt*

$$|x'| \leq -\log \mu'(x) \ .$$

Beweis: Für (*) erhält man

$$|x'| \ \leq \ \log(\mu(x) \cdot r) = \underbrace{\log \mu(x)}_{<0} + \log r \leq \log r \leq -\log \mu'(x) \ .$$

Für (**) gilt nun

$$1 \geq 0, x'1 \geq \mu(x) \cdot 2^{|x'|} \ .$$

```
M_j(w)
    begin
        b ← w_1
        x̃ ← w_2, ..., w_{|w|}
        if b = 0 then
            return M_i(x̃)
        else
            Suche per binärer Suche x,
                so dass r = 2^{⌈− log μ'(x̃)⌉} und
                x̃ = ⌊μ(x) · r + 1/2⌋
            if Suche misslingt then return 0
            else return M_i(x)
            fi
        fi
    end
```

Abbildung 3.2: Die nicht-deterministische Turing-Maschine M_j ist Bestandteil der Reduktion auf NBH.

Damit ist $2^{|x'|}\mu(x) \leq 1$, wobei $\mu(x) \geq \mu'(x)$ woraus folgt $2^{|x'|} \cdot \mu'(x) \leq 1$. □

Für die Reduktionsfunktion haben wir die Wahl von j noch offengelassen. Wir wählen j implizit in Abhängigkeit von i und μ, wobei M_j die nichtdeterministische Turing-Maschine beschreibt mit der folgenden Funktionalität:

Die Korrektheit der Reduktion folgt nun direkt aus der Definition von M_j. Für die Zeitkomplexität der Reduktionsfunktion R gilt sogar

$$\text{time}_R(x) \;\leq\; \mathcal{O}(|x|^c)\,,$$

da die Berechnung von $\mu'(x) = \mu(x) - \mu(x-1)$ in polynomieller Zeit möglich ist wegen $\mu'(x) \geq 2^{-|x|}$. Damit liegt hier sogar eine worst-case-Reduktion vor.

Es bleibt nur noch zu zeigen, dass μ_{NBH} die Wahrscheinlichkeitsverteilung μ dominert, d.h.

$$\exists c > 0 \;:\; \mu_{\text{NBH}}(w) \;\geq \sum_{x:R(x)=w} \frac{\mu'(x)}{|x|^c}\,.$$

Wir betrachten hierzu die beiden folgenden Fälle:

1. Fall: $\mu'(x) < 2^{|x|}$

   Daraus folgt $R(x) = 0x01^j0^{|x|^c} = w$ und damit

   $$\mu'_{\text{NBH}}(w) = 2^{-|x|-1} \cdot \frac{1}{j^2} \cdot \frac{1}{|x|^c} > \mu'(x) \cdot \frac{1}{2j^2|x|^c}\,.$$

2. Fall: $\mu'(x) \geq 2^{-|x|}$

   Damit ist $R(x) = 1x'01^j0^{|x|^c} = w$ und es gilt nach Lemma 6:

   $$\mu'_{\text{NBH}}(x) = 2^{-|x'|-1} \cdot \frac{1}{j^2} \cdot \frac{1}{|x|^c} \geq \mu'(x) \cdot \frac{c}{2j^2|x|^c}\,.$$

□

3.2 Randomisierte Average-Reduktionen

Es zeigt sich, dass die Menge der Dist$\mathcal{NP}$-vollständigen Problem mit natürlichen Wahrscheinlichkeitsverteilungen sehr klein ist. Bis heute sind vielleicht ein oder zwei Dutzend solcher Probleme bekannt und diese

Anzahl würde sich noch drastisch reduzieren, wenn man sich auf die oben vorgestellten deterministischen Average-Reduktionen beschränken würde.

Daher werden wir in diesem Abschnitt den Reduktionsbegriff aufweichen (also generalisieren), indem wir in der Reduktion die Verwendung von Zufallsereignissen erlauben.

Hierzu müssen wir zuerst geeignete randomisierte Maschinenmodelle betrachten. Die **probabilistische Turing-Maschine (PTM)** kann auf verschiedene Weise beschrieben werden.

- Beschreibung durch Berechnungsbaum:

  Eine probabilistische Turing-Maschine kann in jedem Schritt eine Verzweigung des Berechnungsbaumes mit Ausgrad zwei vornehmen. Blätter des Baumes akzeptieren oder verwerfen. Jeder Ast einer Verzweigung wird unabhängig mit Wahrscheinlichkeit $\frac{1}{2}$ beschritten. Die Akzeptanzwahrscheinlichkeit ergibt sich durch die Summe der Wahrscheinlichkeiten in akzeptierende Blätter des Berechnungsbaums zu geraten.

- Orakelband mit Zufallsbits

  In dieser Beschreibung wird die probabilistische Turing-Maschine durch eine deterministische Turing-Maschine beschrieben, die über ein zusätzliches einseitig unbeschränktes Band verfügt. Auf diesem Band stehen die Zeichen 0 oder 1 mit jeweils Wahrscheinlichkeit $\frac{1}{2}$. Die probabilistische Turing-Maschine akzeptiert, wenn sie in einem akzeptierenden Endzustand anhält.

Wie man sich leicht überlegt, sind diese beiden Beschreibungen äquivalent. Ohne Auswirkung auf die Laufzeit, Speicherverbrauch oder Akzeptanzwahrscheinlichkeit lässt sich ein Modell in das andere überführen.

Als probabilistische *worst-case*-Komplexitätsklassen werden im allgemeinen die folgenden Notationen verwendet.

$$\begin{aligned}
\mathcal{RP} := \{L \mid \exists \text{PTM} M \ : \quad & x \in L \implies \mathrm{P}[M(x)=1] \geq \tfrac{1}{2} \ \wedge \\
& x \notin L \implies \mathrm{P}[M(x)=0] = 1 \ \wedge \\
& \mathrm{time}_M(x) \leq \mathrm{POL}(|x|) \ \}
\end{aligned}$$

$$\mathcal{ZPP} := \{L \mid \exists \text{PTM} M \ : \ L(M) = L \ \wedge \ \mathbf{E}[\mathrm{time}_M(x)] \leq \mathrm{POL}(|x|)\} \, ,$$

$$\mathcal{PP} := \{L \mid \exists \text{PTM} M \ : \ \mathrm{P}[M(x) = \chi_L(x)] > \frac{1}{2} \ \wedge \ time_M(x) \leq \mathrm{POL}(|x|)\} \, ,$$

$$\mathcal{BPP} := \{L \mid \exists \text{PTM} M \ : \ \mathrm{P}[M(x) = \chi_L(x)] > \frac{2}{3} \ \wedge \ time_M(x) \leq \mathrm{POL}(|x|)\} \, ,$$

$$\mathcal{XYZ} := \{L \mid \exists \text{PTM} M \ : \ \mathrm{P}[M(x) = \chi_L(x)] > 1 - 2^{-\mathrm{POL}(x)} \ \wedge \ time_M(x) \leq \mathrm{POL}(|x|)\} \, .$$

Hierbei werden die Probleme aus $\mathcal{RP}$ (randomized $\mathcal{P}$) durch sogenannte **Monte-Carlo-Algorithmen** gelöst, während die Probleme aus $\mathcal{ZPP}$ (zero error probability $\mathcal{P}$) durch **Las-Vegas-Algorithmen** beschrieben werden. Im allgemeinen wird die Klasse $\mathcal{BPP}$ (bounded probability $\mathcal{P}$) als die Klasse effizient berechenbarer Probleme angesehen, während die Klasse $\mathcal{PP}$ als zu mächtig angesehen wird (wie wir gleich sehen werden).

Die Klasse $\mathcal{XYZ}$ ist äquivalent zu einer der obigen randomisierten Komplexitätsklassen, wie wir gleich sehen werden.

Eine wichtige Methode im Bereich der Analyse probabilistischer Prozesse ist die Chernoff-Schranke. Diese beschreibt die Wahrscheinlichkeit der Abweichung der Summe von n unabhängigen Bernoulli-Experimenten von dem Mittelwert.

Lemma 7 Chernoff-Schranke
Seien $x_1, \ldots, x_n$ unabhängige Bernoulli-Experimente mit $\mathrm{P}[x_i = 1] = p$ und $\mathrm{P}[x_i = 0] = 1 - p$. Dann gilt für jedes $c \in [0,1]$:

$$\mathrm{P}[\sum_{i=1}^{n} x_i \ \geq \ (1+c)pn] \ \leq \ e^{-\frac{c^2}{3}pn} \, .$$

Beweis: Sei $X := \sum_{i=1}^{n} x_i$. Dann gilt für jedes $t \geq 0$:

$$\mathrm{P}[X \geq (1+c)pn] \;=\; \mathrm{P}[e^{tX} \geq e^{t(1+x)pn}] \,.$$

Die Markov-Ungleichung liefert nun

$$\mathrm{P}[e^{tX} \geq k\mathbf{E}[e^{tX}] \;\leq\; \frac{1}{k} \,.$$

Wir wählen nun $k = e^{t(1+c)pn}/\mathbf{E}[e^{t\cdot X}]$, wobei

$$\mathbf{E}[e^{tX}] = \mathbf{E}[e^{tx_1} \cdot e^{tx_2} \cdot \ldots \cdot e^{tx_n}] = \left(\mathbf{E}[e^{t\cdot x_1}]\right)^n \leq (1+p(e^t-1))^n \,.$$

Damit ist nun

$$\begin{aligned} \mathrm{P}[X \geq (1+c)pn] &\leq e^{-t(1+c)pn} \cdot (1+p(e^t-1))^n \\ &\leq e^{-t(1+c)pn} \cdot e^{pn(e^t-1)} \,, \end{aligned}$$

weil für alle $a \geq 0$ gilt $(1+a)^n \leq e^{an}$. Wir wählen $t = \ln(1+c)$ und erhalten dann

$$\mathrm{P}[X \geq (1+c)pn] \leq e^{pn(c-(1+c)\ln(1+c))} \cdot e^{pn(-\frac{1}{2}c^2+\frac{1}{6}c^3-\frac{1}{22}c^4+\ldots)} \;\leq\; e^{-\frac{1}{3}pnc^2}$$

□

Mit Hilfe dieser Schranke kann man die Fehlerschranke von $\mathcal{BPP}$ von $\frac{2}{3}$ bis auf einen exponentiell kleinen Fehler verringern, was durch folgendes Theorem beschrieben wird.

Theorem 9 Boosting

$$\mathcal{XYZ} = \mathcal{BPP} \,.$$

Beweis: Die Berechnung der probabilistischen $\mathcal{BPP}$-Turing-Maschine wird $36|x|^c$ wiederholt. Wenn die Mehrzahl der Berechnungen akzeptiert, wird die Eingabe akzeptiert sonst verworfen.

Seien $x_1, \ldots, x_m$ für $m = 36|x|^c$ die Ausgaben der Teilberechnungen. Die Wahrscheinlichkeit, dass eine zu verwerfende Eingabe fälschlicherweise akzeptiert wird und damit

$$\sum_{i=1}^{n} x_i \leq \frac{n}{2} \quad \text{für} \quad p = \mathrm{P}[x_i = 0] = \frac{1}{3} \,.$$

Damit ist die Wahrscheinlichkeit für diesen Fehlertyp

$$\begin{aligned} \mathrm{P}[\sum_{i=1}^{m} x_i \leq \frac{m}{2}] &= \mathrm{P}[\sum_{i=1}^{m} x_i \leq pm(1+\frac{1}{2})] \\ &\leq e^{-pm\frac{1}{3}(\frac{1}{2})^2)} \\ &= e^{-m\frac{1}{36}} \;=\; e^{-|x|^c} \,. \end{aligned}$$

□

Als Relationen zwischen den Komplexitätsklassen kennt man

Theorem 10

$$\mathcal{RP} \cap \text{co-}\mathcal{RP} = \mathcal{ZPP} \,.$$

dessen Beweis wir hier nicht ansprechen und

Theorem 11

$$\mathcal{RP} \subseteq \mathcal{NP} \subseteq \mathcal{PP} \,.$$

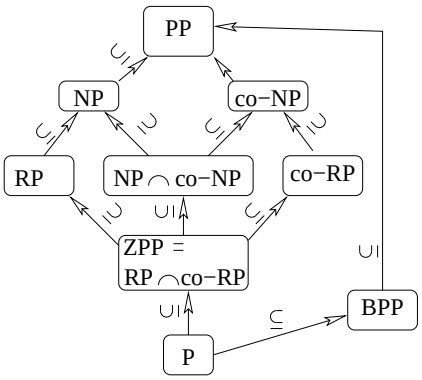


Abbildung 3.3: Die bekannten Relationen zwischen den randomisierten Worst-Case-Komplexitätsklassen.

Beweis: Die erste Inkluision folgt aus der Beschreibung der PTM durch einen Berechnungsbaum. Interpretieren wir die PTM als NTM, so erhalten wir schon die gesuchte Zielmaschine.

Für die zweite Inklusion betrachten wir eine NTM, die einen vollständigen binären Berechnungsbaum besitzt und deren linkester Ast immer verwirft. Man kann sich ein einfaches Verfahren überlegen, dass aus jeder NTM eine solche NTM mit nur konstantem Zeitverlust baut.

Nun ersetzt man jede verwerfende Berechnung durch einen weiteren Teilbaum mit vier Blättern, von der zwei 0 sind und die anderen 1. Akzeptierende Berechnungen werden durch vier Blätter ersetzt in der beide auf 1 gesetzt werden. Für einen verwerfenden Berechnungsast, werden drei 0er und eine 1 eingefügt.

Akzeptiert die ursprüngliche NTM, so erhält man mindestens ein akzeptierendes Blatt mehr als verwerfende in der PTM. Verwirft die NTM, so erhält man genau zwei verwerfende Blätter mehr als Akzeptierende. □

In Abbildung 3.3 sind alle bislang bekannten Relationen in diesen randomisierten Worst-case-Komplexitätsklassen dargestellt.

Wie kann man aber ein vernünftiges randomisiertes Average-Maß definieren? Den randomisierten Worst-case-Komplexitätsklassen liegt mitunter der Erwartungswert zu Grunde. Wir haben bereits gesehen, dass hinsichtlich von Simulation und anderen Transformationen dieser gewisse Nachteile birgt.

Daher schreiten wir einen ähnlichen Weg und erweitern das deterministische Average-Maß

$$\sum_x \frac{\text{time}_M(x)^{1/k}}{|x|} \cdot \mu'(x) \ \leq \ c$$

durch das folgende randomisierte

$$\sum_x \frac{\text{time}_M(x,r)^{1/k}}{|x|} \cdot \mu'(x) \cdot \mu'_{\text{uni}}(r) \;\leq\; c\,,$$

wobei $\text{time}_M(x,r)$ die Ausgabe beschreibt die M auf Eingabe x unter Verwendung der binären Zufallszeichenkette r benötigt und

$$\mu'_{\text{uni}}(x) = \frac{6}{\pi^2} \cdot \frac{1}{2^{|x|}} \cdot \frac{1}{|x|^2}\,.$$

Mit Hilfe dieses Begriffs definieren wir die randomisierte Average-Reduktion.

Definition 8 *Ein verteiltes Problem (L_1, μ_1) kann auf ein verteiltes Problem (L_2, μ_2) reduziert werden, notiert durch*

$$(L_1, \mu_1) \;\leq_{\text{r-av-pol}}\; (L_2, \mu_2)$$

wenn es eine probabilistische Turing-Maschine M gibt mit den folgenden Eigenschaften:

1. **Effizienz**

$$\exists c, k > 0 \;:\quad \sum_{x,r} \frac{\text{time}_M(x,r)^{1/k}}{|x|} \cdot \mu'_{\text{uni}}(x) \cdot \mu'_1(x) \;\leq\; c\,,$$

   wobei $\text{time}_M(x,r)$ *die Laufzeit von M mit verwendeten Zufallsbits $r \in \{0,1\}^*$ bezeichnet.*

2. **Gültigkeit**

$$\forall x \in \{0,1\}^* \;:\; \text{P}[M^{L_2}(x) = L_1(x)] \;\geq\; \frac{2}{3}\,,$$

   wobei M^{L_2} die Turingmaschine M mit Orakel L_2 beschreibt.

3. **Dominanz**

$$\exists c > 0\; \forall y \in \{0,1\}^* \;:\; \mu'_2(y) \;\geq\; \frac{1}{|y|^c} \cdot \sum_x \text{Ask}_M(x,y) \cdot \mu'_1(x)\,,$$

   wobei $\text{Ask}_M(x,y)$ *die Wahrscheinlichkeit beschreibt, dass M auf Eingabe x die Orakelfrage $y \in L_2$ stellt.*

Die passende randomisierte Klasse zu Av-$\mathcal{P}$ ist Av$\mathcal{BPP}$, welche Eigenschaften aus Av-$\mathcal{P}$ und $\mathcal{BPP}$ vereinigt.

$$\text{Av}\mathcal{BPP} \;:=\; \{(L,\mu) \mid \exists \text{PTM}\, M \;:\; \text{P}[M(x) = L(M)] \geq \frac{2}{3} \wedge \exists c, k > 0 \sum_{r,x} \frac{\text{time}_M(x,r)^{1/k}}{|x|} \mu'(x)\mu'_{\text{uni}}(r) \leq c\,\}\,.$$

Diese Klasse ist unter der randomisierten Average-Reduktion abgeschlossen.

Theorem 12

$$P_1 \leq_{\text{r-av-pol}} P_2 \;\wedge\; P_2 \in \text{Av}\mathcal{BPP} \;\Longrightarrow\; \text{Av}\mathcal{BPP}\,.$$

Der Beweis dieses Theorems sei dem Leser zur Übung überlassen.

3.3 Ein natürliches Dist$\mathcal{NP}$-vollständiges Problem

Wir stellen nun ein natürliches Dist$\mathcal{NP}$-vollständiges Problem vor.

Das **Repli-Domino-Problem (Tiling Problem)** ist gegeben durch ein quadratischen Spielfeld der Größe $n \times n$ und m Kopiervorlagen für Puzzle-Teile der Größe 1×1. Die Ränder der i-ten Puzzle-Teile werden durch die Zeichenketten s_i^j für $j \in \{N, S, O, W\}$ beschrieben.

Die Aufgabenstellung ist nun das Spielfeld mit Kopien dieser Vorlagen auszulegen, dass die Nordseite mit benachbarten Südseite und die Ostseite einer benachbarten Westseite der Puzzle-Kopien übereinstimmen.

Als Wahrscheinlichkeitsverteilung verwenden wir

$$\mu'_{\text{RT}}(n, m, (s_i^j)_{i\in[m],j\in[4]}) \;=\; c \cdot \frac{1}{n^2} \cdot \frac{1}{m^2} \cdot \prod_{i\in[m],j\in[4]} \mu'_{\text{uni}}(s_i^j) \; .$$

Theorem 13 *Randomized Tiling ist Dist$\mathcal{NP}$-vollständig unter randomisierter Average-Reduktion.*

Im Rahmen dieser Vorlesung wurde der Beweis aus Zeitgründen nicht behandelt. Man beachte, dass keine deterministische Reduktion bekannt ist, die das gleiche leisten kann.

Es sind nur zwei Dutzend Dist$\mathcal{NP}$-vollständige Probleme bekannt. Damit ist diese Zahl erheblich kleiner als zum Beispiel die Menge der $\mathcal{NP}$-vollständigen Probleme. Ein Grund ist sicherlich, dass es ungleich komplexer ist eine Average-Reduktion zu finden als eine Worst-Case-Reduktion. Solche Probleme findet man in [WB92, Gur91, BG95].

3.4 Bösartige Wahrscheinlichkeitsverteilungen

Wir haben mit POL-computable die Menge der Wahrscheinlichkeitsverteilungen eingeschränkt. Tatsächlich gibt es eine Wahrscheinlichkeitsverteilung bezüglichen der jedes Problem im Erwartungswert das asymptotische Worst-case-Verhalten annimmt [Kob92, LV92, Mil93].

Theorem 14 *Es gibt eine Wahrscheinlichkeitsverteilung μ, so dass für alle Turing-Maschinen M gilt*

$$\exists c > 0 \, \forall n \;:\quad \mathbf{E}_\mu[\text{time}_M(x) \mid x \in \Sigma^n] \;\geq\; c \cdot t_M(|x|) \; ,$$

wobei $t_M(n) := \max_{x\in\Sigma^n} \text{time}_M(x)$.

Beweis: Wir betrachten für jede Turing-Maschine M_i die folgende Wahrscheinlichkeitsverteilung μ_i:

$$\mu'_i(x) \;:=\; \begin{cases} 0 \; , & \text{falls } \text{time}_{M_i}(x) < t_{M_i}(|x|) \vee M_i(x) \uparrow \; , \\ \frac{1}{|\{z\in\Sigma^{|x|} \mid \text{time}_{M_i}(z)=t_M(|x|)\}|} \; , & \text{sonst .} \end{cases}$$

Nun definieren wir

$$\mu'(x) = \sum_{i=1}^{\infty} \alpha(i)\mu'_i(x) \; ,$$

wobei $\alpha(i) > 0$ und $\sum_{i=1}^{\infty} \alpha(i) = 1$. Dann ist

$$\mathbf{E}_\mu[\text{time}_{M_i}(x) \mid x \in \Sigma^n] \;\geq\; \alpha(i) \cdot \mathbf{E}_{\mu_i}[\text{time}_{M_i}(x)] = \alpha(i) t_{M_i}(x) \; .$$

□

Für die Berechnung dieser bösartigen Wahrscheinlichkeitsverteilung ist es notwendig zu entscheiden, ob die Turing-Maschine M_i auf Eingabe x hält. Dieses Halte-Problem ist nicht Turing-berechenbar und damit kann die Wahrscheinlichkeitsverteilung auch nicht berechnet werden.

Dennoch gelingt es diese Technik für polynomiell beschränkte Wahrscheinlichkeitsverteilungskomplexitätsklassen zu übernehmen. In [BDCGL92] wurde folgende Alternative zu Computable vorgestellt.

Definition 9

$$\text{POL-}\textit{sampleable} := \{\mu \mid \exists \textit{PTM } M : \mathrm{P}[M(\lambda) = x] = \mu'(x) \wedge \mathrm{time}_M(M(\lambda) \to x) \leq \mathrm{POL}(|x|) .$$

Es gibt also eine randomisierte Turing-Maschine, die auf der leeren Eingabe λ die Ausgabe x mit der Wahrscheinlichkeit $\mu'(x)$ in Zeit $|x|^c$ ausgibt. Dieser Berechenbarkeitsbegriff veralgemeinert den Begriff der POL-computable:

Theorem 15

$$\forall \epsilon > 0 : \forall \mu_1 \in \text{POL-}\textit{computable } \exists \mu_2 \in \text{POL-}\textit{sampleable} : \forall x : \mu_2'(x) = (1 \pm \epsilon)\mu_1'(x) .$$

Beweis: Die Beweisidee ist eine Zufallszahl $r \in [0, 1]$ zu bestimmen und per binärer Suche $\mu^{-1}(r)$ zu berechnen. Die Laufzeit ist linear in der Länge von r und der Berechungszeit der Wahrscheinlichkeitsverteilung. Um die Approximation genügend genau vorzunehmen wird r um $-\log \epsilon$ zusätzliche Stellen, als zur binären Suche notwendig, ausgeführt. □

Interessanterweise gibt es in POL-sampleable eine universelle Wahrscheinlichkeitsverteilung, die alle anderen Wahrscheinlichkeiten in POL-sampleable dominiert. Bezüglich dieser Wahrscheinlichkeitsverteilung wird jedes $\mathcal{NP}$-vollständige Problem auch $\mathcal{NP} \times$ POL-sampleable-vollständig.

Theorem 16 *Es existiert eine universelle Wahrscheinlichkeitsverteilung $\mu_2 \in$ POL-sampleable, so dass für alle $\mathcal{NP}$-vollständigen Problem L_2 gilt:*

$$\forall (L_1, \mu_1) \in \mathcal{NP} \times \text{POL-}\textit{sampleable} : (L_1, \mu_1) \leq_{\textit{av-pol}} (L_2, \mu_2) .$$

Beweis: Zuerst beobachten wir, dass es eine Aufzählung $\mu_1, \mu_2, \ldots$ aller POL-sampleable Wahrscheinlichkeitsverteilungen gibt. Sei nun

$$\mu_2'(x) = \sum_{i=1}^{\infty} \mu_i'(x)$$

dann folgt das Theorem durch eine worst-case Reduktion in Kombination der Dominanz der Wahrscheinlichkeitsverteilungen. [2] □

[2] Ausführung der Beweisdetails als Übungsaufgabe

Kapitel 4

Average-Schaltkreiskomplexitätstheorie

4.1 Worst-Case-Schaltkreis-Klassen

Definition 10 *Eine Schaltkreisfamilie $\mathcal{C} = (C_0, C_1, C_2, \ldots)$ ist uniform konstruierbar, gdw. es eine log-space-DTM existiert, die auf Eingabe $\$\underbrace{1\ldots1}_{n}\$$ eine Kodierung des Schaltkreis C_n (z.B. in Adjazenslisten-darstellung) ausgibt.*

Wir definieren nun für Funktionen $D, S : \mathbb{N} \to \mathbb{N}$ uniforme Schaltkreiskomplexitätsklassen:

Definition 11

$$\begin{aligned}
\text{CirDepth}(D) &:= \{P \subseteq B^* \mid \exists \textit{ uniforme } B_2\textit{-Schaltkreisfamilie} \\
&\quad \mathcal{C} = (C_0, C_1, \ldots) \textit{ mit } \forall n \,:\, \text{depth}(C_n) \leq D(n)\} \\
\text{CirSize}(S) &:= \{P \subseteq B^* \mid \exists \textit{ uniforme } B_2\textit{-Schaltkreisfamilie} \\
&\quad \mathcal{C} = (C_0, C_1, \ldots) \textit{ mit } \forall n \,:\, \text{size}(C_n) \leq S(n)\} \\
\text{CirDepthSize}(D, S) &:= \{P \subseteq B^* \mid \exists \textit{ uniforme } B_2\textit{-Schaltkreisfamilie} \\
&\quad \mathcal{C} = (C_0, C_1, \ldots) \textit{ mit} \\
&\quad \forall n \,:\, \text{size}(C_n) \leq S(n) \,\wedge\, \text{depth}(C_n) \leq D(n)\}
\end{aligned}$$

Die Klasse der hochgradig parallelisierbaren Probleme wird $\mathcal{NC}$ (Nick's Class nach Nicholas Pippenger) genannt. Sie ist wie folgt untergliedert:

$$\mathcal{NC}^1 := \text{CirDepthSize}(O(\log n), \text{POL}) .$$

In $\mathcal{NC}^1$ sind Probleme wie Addition, Vergleich, Maximumsbildung zweier Binärzahlen, Vermengen zweier sortierter Zahlenfolgen (*Merge*), Multiplikation, Summe von n n-stelligen Bitzahlen und Matrixmultiplikation enthalten.

$$\mathcal{NC}^2 := \text{CirDepthSize}(O(\log^2 n), POL) .$$

In dieser Komplexitätsklasse liegt die Berechnung der Determinante, transitiver Abschluß einer Booleschen Matrix, Matrix-Invertierung und Graphzusammenhang (hierzu später mehr).

Die allgemeine Klasse $\mathcal{NC}^k$ ist definiert für $k \geq 1$ als

$$\mathcal{NC}^k := \text{CirDepthSize}(O(\log^k n), POL) .$$

Die Vereinigung ergibt

$$\mathcal{NC} := \bigcup_{k \geq 1} \mathcal{NC}^k .$$

4.1.1 Unbeschränkte Schaltkreise

Die Boolesche Funktionenfamilie Disjunktion $\vee^*$ und Konjunktion $\wedge^*$ sind definiert als die Vereinigung aller mehrstelligen Oder-Funktionen, bzw. Und-Funktionen:

$$\begin{aligned} \vee^* &:= \{\vee, (x_1, x_2) \mapsto x_1 \vee x_2, (x_1, x_2, x_3) \mapsto x_1 \vee x_2 \vee x_3, \ldots\} \\ \wedge^* &:= \{\wedge, (x_1, x_2) \mapsto x_1 \wedge x_2, (x_1, x_2, x_3) \mapsto x_1 \wedge x_2 \wedge x_3, \ldots\} \end{aligned}$$

Ein $\vee^* \cup \wedge^*$-Schaltkreis mit Eingaben $x_1, \ldots, x_n$ und den zusätzlich zur Verfügung gestellten Eingaben $\overline{x_1}, \ldots, \overline{x_n}$ wird **unbeschränkter Schaltkreis** genannt. Wir definieren für unbeschränkte Schaltkreise C die Tiefe **depth**(C) genauso wie bei beschränkten Schaltkreisen. Für die Größe eines unbeschränkten Schaltkreises C bezeichnen wir abweichend von beschränkten:

$$\textbf{size}(C) \ := \ \text{Summe der Eingrade aller Gatter.}$$

Wir definieren Schaltkreisfamilien und logarithmische Platzkonstruierbarkeit analog zu den beschränkten Schaltkreisen. Die unbeschränkten Schaltkreiskomplexitätsklassen sind:

Definition 12

$$\begin{aligned} \text{UbCirDepth}(D) &:= \{P \subseteq B^* \mid \exists \textit{ uniforme unbeschr. Schaltkreisfamilie} \\ &\quad \mathcal{C} = (C_0, C_1, \ldots) \textit{ mit } \forall n \ : \ \text{depth}(C_n) \leq D(n)\} \\ \text{UbCirSize}(S) &:= \{P \subseteq B^* \mid \exists \textit{ uniforme unbeschr. Schaltkreisfamilie} \\ &\quad \mathcal{C} = (C_0, C_1, \ldots) \textit{ mit } \forall n \ : \ \text{size}(C_n) \leq S(n)\} \\ \text{UbCirDepthSize}(D, S) &:= \{P \subseteq B^* \mid \exists \textit{ uniforme unbeschr. Schaltkreisfamilie} \\ &\quad \mathcal{C} = (C_0, C_1, \ldots) \textit{ mit} \\ &\quad \forall n \ : \ \text{size}(C_n) \leq S(n) \ \wedge \ \text{depth}(C_n) \leq D(n)\} \end{aligned}$$

Die $\mathcal{NC}$ entsprechenden Komplexitätsklassen heißen $\mathcal{AC}$ (*alternating class*). Für jede Boolesche Funktion kann ein unbeschränkter Schaltkreis der Tiefe 2 angegeben werden. Somit macht es Sinn, konstante Tiefe zu betrachten. Nun muß aber berücksichtigt werden, daß die Größe des unbeschränkten Schaltkreises polynomielle Tiefe nicht überschreitet und die logarithmische Platzkonstruierbarkeit gewahrt bleibt.

$$\mathcal{AC}^0 \ = \ \text{UbCirDepthSize}(O(1), \text{POL}) \ .$$

In dieser Komplexitätsklasse liegen beispielsweise Addition, Vergleich und Maximum zweier n-Bit-Zahlen, sowie *Merge*.

$$\mathcal{AC}^1 \ = \ \text{CirDepthSize}(O(\log n), \text{POL}) \ .$$

Nachweislich nicht in $\mathcal{AC}^0$, aber in $\mathcal{AC}^1$ liegen die Probleme Parität, sowie Multiplikation, Summe von n Zahlen der Länge n; Sortieren von n-Bitzahlen, Majorität und transitiver Abschluß.

Entsprechend definieren wir $\mathcal{AC}^k$ für $k \geq 1$ als

$$\mathcal{AC}^k \ = \ \text{CirDepthSize}(O(\log^k n), \text{POL})$$

und die allgemeine Klasse $\mathcal{AC}$ als

$$\mathcal{AC} \ := \ \bigcup_{k \geq 0} \mathcal{AC}^k \ .$$

Für den Vergleich von beschränkten und unbeschränkten Komplexitätsklassen gilt folgendes Lemma:

Lemma 8

$$\begin{aligned} \text{CirDepthSize}(D, C) &\subseteq \text{UbCirDepthSize}(D, 2C) \\ \text{UbCirDepthSize}(D, C) &\subseteq \text{CirDepthSize}(O(D \log C), O(C)) \end{aligned}$$

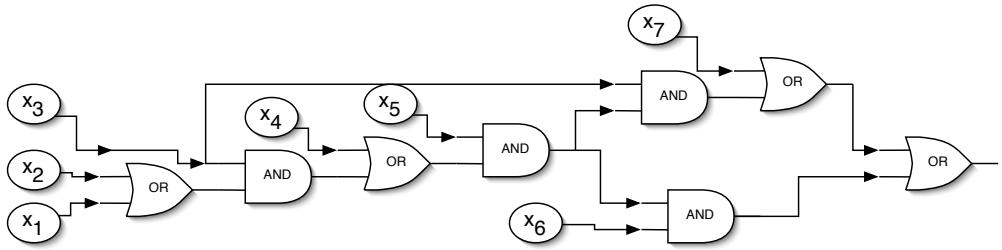


Abbildung 4.1: Dieser Schaltkreis mit mininmaler Größe berechnet für alle Eingaben die Ausgabe schneller als die Tiefe erwarten lässt.

Sei nun $\mathcal{L}$ die Menge aller Probleme, die in logarithmischen Platz von einer DTM berechnet werden können, sowie $\mathcal{NL}$ die Menge aller Probleme die in logarithmischen Platz von einer NTM berechnet werden können. Dann gilt

Theorem 17

$$\mathcal{AC}^0 \subseteq \mathcal{NC}^1 \subseteq \mathcal{L} \subseteq \mathcal{NL} \subseteq \quad \mathcal{AC}^1 \subseteq NC^2 \subseteq \mathcal{AC}^2 \quad \subseteq \ldots$$
$$\mathcal{AC}^k \subseteq \mathcal{NC}^{k+1} \subseteq \mathcal{AC}^{k+1} \quad \subseteq \ldots \mathcal{AC} = \mathcal{NC} \subseteq \mathcal{P} \; .$$

Bevor wir den Durchschnittsbegriff im Average klären, einigen wir uns noch auf die folgenden elementaren Definitionen.

Wir bezeichnen mit B_n^m die Menge der Booleschen Funktion $f : \{0,1\}^n \to \{0,1\}^m$, wobei $B_n := B_n^1$. Wir bezeichnen hier mit $\mathcal{D}_n$ die Menge aller (diskreten) Wahrscheinlichkeitsfunktionen μ über $\{0,1\}^n$. Die uniforme Wahrscheinlichkeitsfunktion über $\{0,1\}^n$, die jede der 2^n möglichen Eingabevektoren gleichgewichtet, schreiben wir hier als $\mu_n^{\mathbf{uni}}$. Für eine Verteilung $\mu \in \mathcal{D}_n$ beschreibe supp(μ) die Menge aller Eingabevektorens $x \in \{0,1\}^n$ mit positiver Wahrscheinlichkeit $\mu(x)$ (also nicht Null). Eine Wahrscheinlichkeitsverteilung in $\mathcal{D}_n$ ist positiv falls alle Werte nicht Null sind, also supp$(\mu) = \{0,1\}^n$.

4.2 Zeit in Schaltkreisen

In Schaltkreisen, dargestellt als gerichtete azyklische Graphen, wird üblicherweise die Tiefe als Maß für den Berechnungsverzögerung (delay) verwendet. In der Tat gibt es einen engen Zusammenhang zwischen Schaltkreistiefe under Zeit in verschieden parallelen Maschinenmodellen. Letztlich ist aber die Tiefe eines Schaltkreise im besten Fall eine Beschreibung des Worst-case.

Krapchenko [Kra78] hat einen Schaltkreis mit minimaler Größe vorgestellt, für welchen die Schaltkreiszeit für jede Eingabe kleiner ist als die Tiefe des Schaltkreises, siehe Abbildung 4.1. Somit besteht die berechtigte Hoffnung, dass mit der Betrachtung der Zeit im Schaltkreis statt der Tiefe eine Verbesserung eintreten wird.

Wir haben bereits gesehen, dass neben dem Erwartungswert auch ein anderes Konzept der Average-Komplexität Sinn macht, nämlich der Ansatz von Levin [Lev84, Gur91, BDCGL92, RS93]. Natürlich macht es auch wenig Sinn für Schaltkreise nur uniformen Wahrscheinlichkeitsverteilungen zu betrachten. Gibt es aber auch hier bösartige Wahrscheinlichkeitsverteilungen? Kann man überhaupt in den zumeist logarithmisch oder polylogarithmisch tiefenbeschränkten Schaltkreisen signifikante Verbesserungen im Average erwarten? Diese und andere Fragen werden wir hier erörtern.

Als erstes stellt sich die Frage, ob es überhaupt Sinn macht ein so rigides Berechnungsmodell wie Boolesche Schaltkreise im Average zu betrachten. Der Schlüssel hierzu liegt in einer geeigneten Version der Zeit in einem Schaltkreis. Wir betrachten daher zwei (scheinbar) verschiedene Modelle.

4.2.1 Eine implizite Definition der Zeit

Schaltkreise können für eine beliebige Basis von Grundfunktionen gegeben sein. Bezeichne Cir(f) die Menge aller Schaltkreise über der gegebenen Basis, welche die Funktion f berechnen. Betrachten wir diese

Funktion bezüglich einer Wahrscheinlichkeitsverteilung μ, so genügt es für einen Schaltkreis in $\text{Cir}_\mu(f)$, dass nur die positiv gewichteten Eingaben $x \in \text{supp}(\mu)$ mit $f(x)$ korrekt berechnet werden.

Informationen können in einem Schaltkreis schneller propagiert werden, wenn zum Beispiel eine der Eingaben eines OR-Gatters früher verfügbar ist als die andere und den Wert 1 besitzt. Dadurch wird der Ausgabewert des Gatters durch 1 vorzeitig verfügbar. Formal definieren wir eine Funktion time : $\{0,1\}^n \to \mathbb{N}$ für jedes Gatter v eines Schaltkreises C. Es bezeichnet für jede Eingabe x den Zeitschritt, wenn der Ausgabewert von v aus den anliegenden Eingaben berechnet werden kann. Mit $\text{res}_v(x)$ bezeichnen wir den Ausgabewert des Gatters v auf Eingabe x.

Definition 13 *Sei C ein Schaltkreis und v ein Gatter von C. Für die Eingabe und Konstantengatter v definieren wir* $\text{time}_v(x) := 0$. *Für interne nichtkonstante Gatter v mit k direkten Vorgängern $v_1, \ldots, v_k$ definieren wir* $\text{time}_v(x) := 1 + \min\{t \mid$ *die Werte* $\text{res}_{v_i}(x)$ *mit* $\text{time}_{v_i}(x) \leq t$ *bestimmen eindeutig* $\text{res}_v(x)\}$. *Für den Gesamtschaltkreis C mit Ausgabegattern $y_1, \ldots, y_m$ ergibt sich die Gesamtlaufzeit durch*

$$\text{time}_C(x) := \max_i \text{time}_{y_i}(x) .$$

Gemäß dieser Definition ist beispielsweise die Verzögerung eines Gatters v mit Vorgängergattern v_1,v_2 gegeben durch

$$\text{time}_v(x) := 1 + \begin{cases} \max\{\text{time}_{v_1}(x), \text{time}_{v_2}(x)\} & \text{if } \text{res}_{v_1}(x) = \text{res}_{v_2}(x) = 0, \\ \min\{\text{time}_{v_i}(x) \mid \text{res}(v_i) = 1\} & \text{else.} \end{cases}$$

Dieses Zeitmodell definiert die Berechnungszeit nur implizit. Das Gatter selbst "weiss" nicht, ob es mit der Berechnung fertig ist. Daher nennen wir solche Schaltkreise **implizite Zeit-Schaltkreise (implicitly timed circuits), IT-Schaltkreise**. Im folgenden betrachten wir die Boolesche Standardbasis mit mit AND, OR and NOT Gattern.

Jede vollständige Basis muss Gatter enthalten, die eine beschleunigte Berechnung zulassen. Somit ist die Beschleunigungseigenschaft unabhängig von der Basiswahl.

Theorem 18 *Für alle vollständigen endlichen Bases B_1, B_2 gibt es eine Konstante k mit der folgenden Eigenschaft. Jeder Schaltkreis C_1 aus Gatter aus B_1 hat einen äquivalenten Schaltrkeis C_2 mit Gattern aus B_2, wobei*

$$\text{time}_{C_2} \leq k\text{time}_{C_1} .$$

Beweis: Es genügt zu zeigen, dass eine beliebiige Basis B_1 in konstanter Zeit durch die Standardbasis simuliert werden kann und umgekehrt.

Die erste Eigenschaft kann wie folgt eingesehen werden: Für eine Boolesche Funktion f die zu einem Gatter aus B_1 korrespondiert, wird jeder Primimplikant durch einen Baum von AND-Gatter beschrieben. Diese werden durch einen Baum von OR-Gatter vereinigt. So erhält man den Schaltkreis C_f.

Gibt es eine Teileingabe, die f mit 1 bestimmt, dann gibt es einen Primimplikanten von f, für den alle Eingaben verfügbar sind. Dieser AND-Baum kann damit mit mit 1 ausgewertet werden und dieses Ergebnis kann im OR-Baum vorzeitig ausgewertet werden.

Ist das Ergebnis von f durch Teileingaben mit 0 bestimmt, so muss jeder Primimplikant von f ein Literal mit Wert 0 besitzen. Somit kann der AND-Baum vorzeit ausgewertet werden und damit sind alle Eingaben des OR-Baums gegeben und der Schaltkreis C_f liefert ebenfalls vorzeitig sein Resultat.

Ersetzt man für jede Funktion f in B_1 alle Gatter durch solch einen Typ CC_f erhält man höchstens einen konstanten Faktor Zeitverlust, wobei diese Konstante duch die Tiefe der C_f-Schaltkreise beschränkt ist.

Wir betrachten nun eine (Nicht-Standard)-Basis B_2, welche die Standard-Basis zeiteffizient simulieren soll. Die 16 Booleschen Funktion f mit zwei Eingaben lassen sich in drei verschieden Grundtypen klassifizieren:

1. Der AND-Typ: ($f(x_1, x_2) = (x_1^\alpha \wedge x_2^\beta)^\gamma$ für $\alpha, \beta, \gamma \in \{0,1\}$),
2. DerPARITY-Typ ($f(x_1, x_2) = x_1 \otimes x_2 \otimes \alpha$ mit $\alpha \in \{0,1\}$),

3. und die elementaren Funktionen (0, 1, x_1, x_2, $\overline{x_1}$, $\overline{x_2}$).

B_2 muss eine Funktion $b : \{0,1\}^q \to \{0,1\}$ enthalten, so dass $b(p_1(x_1,x_2),\ldots,p_q(x_1,x_2)) = a(x_1,x_2)$, wobei p_i Funktionen elementaren Typs oder vom PARITY-Typ sind und a eine AND-type Funktion ist. Dies ist zwingend notwendig, da elementare und PARITY-Funktion bezüglich Komposition abgeschlossen sind.

Für solch ein b muss sogar eine elementare Funktion e_i existieren $b(e_1(x_1,x_2),\ldots,e_q(x_1,x_2)) = a'(x_1,x_2)$, so dass a' vom Typ AND ist.

Nach der (impliziten) Definition der Zeit in einem Schaltkreis hat die Funktion $b \circ (e_1,\ldots,e_q)$ das gleiche Zeitverhalten wie a'. Die Anwendung der De-Morganschen Regeln ergibt Konstruktionen für AND- und OR-Gatter. □

Wird ein Schaltkreis mit einer Wahrscheinlichkeitsverteilung μ an den Eingaben bedient, wobei nicht alle Eingaben mit positiver Wahrscheinlichkeit vorkommen, ergeben sich weitere Möglichkeiten das Zeitverhalten zu verbessern. Wird zum Beispiel $\bigvee_{i=1}^n x_i$ für eine Wahrscheinlichkeitsverteilung mit $\mathrm{P}[X = 0^n] = 0$ betrachtet, genügt ein Schaltkreis mit konstanter Ausgabe 1. Diese Effekte werden wir hier nicht näher untersuchen.

4.2.2 Eine explizite Definition der Zeit in Schaltkreisen

Um die Zeit in einem Schaltkreis explizit auszudrücken, kodiert man nicht ausgewertete Ergebnisse durch ein zusätzliches Symbol "?", welches "keine Ahnung" bedeutet. So erhält man aus der zweiwertigen Booleschen Logik eine Dreiwertige.

Im Fall der OR and AND-Funktionen erhält man beispielsweise die folgenden Funktionstabellen.

AND	?	0	1
?	?	0	?
0	0	0	0
1	?	0	1

OR	?	0	1
?	?	?	1
0	?	0	1
1	1	1	1

Alle nichtkonstanten Gatter werden mit "?" initialisiert. Sobald die anliegenden Eingaben eines Gatters die Ausgabe bestimmen wird diese auf 0 oder 1 gesetzt. Solche Schaltkreise nennen wir Schaltkreise mit expliziter Zeitdarstellung (auf Englisch: **explicitly timed circuits, ET-circuits**). Ein ähnliches Konzept wurde unter dem Namen *self-timed circuits* in [DGY90, LBSV93] vorgestellt.

Offensichtlich kann man durch eine einfache Kodierung diese Darstellung durch einen Booleschen Schaltkreis darstellen, der nur doppelt so groß ist, aber das gleiche Zeitverhalten hat. Hierfür kodiert man gemäß des Eisenbahn-Codes (railway code) $\mathrm{code}(?) = \{(0,0)\}$, $\mathrm{code}(0) = \{(0,1)\}$, $\mathrm{code}(1) = \{(1,0)\}$) [DGY90].

4.2.3 Äquivalenz der Zeitmodelle

Folgendes Lemma zeigt die Äquivalenz der verschiedenen Zeitmodelle.

Lemma 9 *Für jeden impliziten Zeitschaltkries I gibt es einen expliziten Zeitschaltkreis E' und für jeden expliziten Zeischaltkreis E gibt es einen impliziten I', so dass die Funkionalität äquivalent ist und es gilt:*

$$\begin{aligned}
\mathrm{size}(E') &\leq 2\mathrm{size}(I)\,, \\
\mathrm{depth}(E') &\leq \mathrm{depth}(I)\,, \\
\mathrm{time}(E') &\leq \mathrm{time}(I)\,, \\
\mathrm{size}(I') &\leq \mathrm{size}(E)\,, \\
\mathrm{depth}(I') &\leq \mathrm{depth}(E)\,, \\
\mathrm{time}(I') &\leq \mathrm{time}(E)\,.
\end{aligned}$$

Diese Lemma folgt aus der Definition von impliziter und expliziter Zeit in Schaltkreisen und der Anwendung des Eisenbahn-Codes.

4.3 Average-Maße für Schaltkreise

Abweichend vom vorherigen Kapitel bezeichne $\mu(x)$ die Wahrscheinlichkeit eines Ereignisses und nicht die Summe von Wahrscheinlichkeiten. Damit ist für eine Funktion $t : \{0,1\}^n \to \mathbb{N}$ und solch einer Wahrscheinlichkeitsverteilung $\mu : \{0,1\}^n \to [0;1]$ der Erwartungswert gegeben durch

$$\mathbf{E}_\mu(t) \ := \sum_{x \in \{0,1\}^n} t(x)\ \mu(x)\ .$$

Für eine Klasse von Wahrscheinlichkeitsfunktionen definieren wir

$$\text{ECirTime}(f, D) \ := \max_{\mu \in D} \quad \min_{C \in Cir_\mu(f)} \quad \mathbf{E}_\mu(\text{time}_C)$$

als die optimale erwartete Schaltkreiszeit von f bezüglich D.

Der klassische Fall eines Booleschen Schaltkreis der erheblich schneller im Average als im Worst-Case berechnet werden kann, ist die Disjunktion.

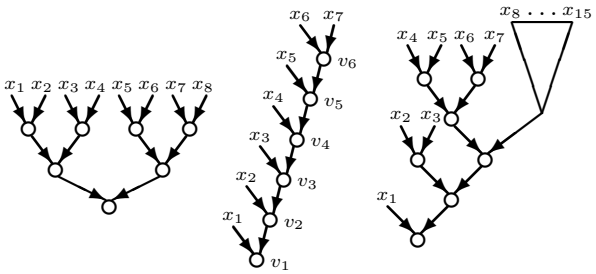


Abbildung 4.2: Drei verschiede Schaltkreisentwürfe für die Disjunktion

Theorem 19 $\text{ECirTime}(\texttt{OR}_n, \mu_{\text{uni}_n}) \ := \ 2 - 2^{-(n-2)}$.

Beweis: Man betrachte den Schaltkreis C der in der Mitte der Abbildung 4.2 dargestellt ist. Dann folgt

$$\text{P}[\text{time}_C(x) = t] \quad = \quad \begin{cases} 2^{-t} & \text{for } t \leq n-2\ , \\ 2^{-(t-1)} & \text{for } t = n-1\ . \end{cases}$$

Damit gilt

$$\begin{aligned} \mathbf{E}(\text{time}_C) \quad &= \quad \sum_{t=1}^{n-2} \frac{t}{2^t} + \frac{n-1}{2^{n-2}} \ := \ \sum_{t=1}^{n-2} \left(\frac{2}{2^t} - \frac{1}{2^{n-1}} \right) + \frac{n-1}{2^{n-2}} \\ &= \quad 2 - \frac{2}{2^{n-2}} - \frac{n-2}{2^{n-2}} + \frac{n-1}{2^{n-2}} \ \leq \ 2 - 2^{-(n-2)}\ . \end{aligned}$$

Für die untere Schranke kann gezeigt werden, dass jeder strukturell verschiedene Schaltkreis im Erwartungswert länger rechnet [JRS94]. □

Natürlich ist diese Schaltkreiskonstruktion nicht wünschenswert, weil während das erwartete Zeitverhalten optimiert wird, die Tiefe maximiert wird. Eine Lösung ergibt sich durch die Adaption des Levinschen Average-Maß. Da die Zeitschranken in Schaltkreisen zumeist langsamer als linear wachsen, müssen wir zuerst definieren was man unter Umkehrfunktion im Levinschen Sinne zu verstehen hat.

Definition 14 *Für eine monotone Komplexitätsschranke $T : \mathbb{N} \to \mathbb{N}$ definieren wir die Inverse durch*

$$T^{-1}(\tau) \ := \ \min\{m \mid T(m) \geq \tau\}\ .$$

Eine Funktion $t : \{0,1\}^n \to \mathbb{N}$ ist Average T beschränkt bezüglich der Wahrscheinlichkeitsverteilung $\mu : \{0,1\}^n \to [0;1]$, beschrieben durch $(t,\mu) \in \mathrm{Av}(T)$, falls

$$\sum_{x\in\{0,1\}^n} T^{-1}(t(x))\,\mu(x) \;\le\; n\;.$$

Für Boolesche funktionen $f \in B_n^m$, einer Klasse von Wahrscheinlichkeitsverteilungen $D \subseteq \mathcal{D}_n$ und einer Komplexitätsschranke T definieren wir die Relation

$$\mathrm{AvCirTime}(f,D) \le T \quad := \quad \forall \mu \in D \quad \exists C \in \mathit{Cir}_\mu(f)\ (\mathrm{time}_C,\mu) \in \mathrm{Av}(T)\;,$$

was bedeutet, dass für jede Verteilung aus D für die Funktion f ein T-Average effizienter Schaltkreis existiert.

Technisch gesehen, ist der Umgang mit dem Average-Maß komplizierter als mit dem Erwartungswert. Aber schon das Beispiel der Disjunktion zeigt, dass dieses Maß bessere Schaltkreiskonstruktionen notwendig macht.

Sei nun $\mathrm{llog} : n \mapsto \log\log n$ der zweifach iterierte Logarithmus zur Basis 2. Ferner beschreibe $\exp n := 2^n$.

Theorem 20

$$\mathrm{llog} - 1 \;<\; \mathrm{AvCirTime}(\mathtt{OR}_n, \mu_{\mathrm{uni}\,n}) \;\le\; 2\mathrm{llog} + 1\;.$$

Beweis: In Abbildung 4.2 auf der rechten Seite kann man die entsprechende Schaltkreiskonstruktion sehen. Das Average-Zeitverhalten ergibt sich aus den folgenden Wahrscheinlichkeiten.

$$\begin{aligned}\mathrm{P}[\mathrm{time}_C(x) = 2t-1] &= 2^{-2^t+1}\;,\\ \mathrm{P}[\mathrm{time}_C(x) = 2\log n] &\le 2^{-(n-1)}\;.\end{aligned}$$

Um $(\mathrm{time}_C,\mu_{\mathrm{uni}}) \in \mathrm{Av}(2\mathrm{llog}+1)$ zu bekommen, muss für $T = 2\mathrm{llog}+1$

$$\sum_{t\ge 1} \frac{T^{-1}(t)}{n}\,\mathrm{P}[\mathrm{time}_C(x) = t] \;\le\; 1$$

gezeigt werden. Dies folgt für $n \ge 4$.

$$\sum_{t=1}^{\log n} \frac{T^{-1}(2t-1)}{n}\frac{2}{2^{2^t}} + \frac{T^{-1}(2\log n)}{n}\frac{2}{2^n} \;\le\; \sum_{t=1}^{\log n} \frac{2^{2^t}}{n}\frac{2}{2^{2^t}} + \frac{2^{2^{\log n}}}{n}\frac{2}{2^n} \;\le\; \sum_{t=1}^{\log n} \frac{2}{n} + \frac{2}{n} \;\le\; 1\;.$$

Die untere Schranke kann wie folgt gezeigt werden. Für jeden Schaltkreis C und jeder Eingabe x mit $\mathrm{time}_C(x) = t$ ist das Ergebnis von höchstens 2^t Eingabevariablen abhängig. Für $t < \log n$ impliziert dies, das wenigstens eine Eingabe den Wert 1 haben muss, welches mit Wahrscheinlichkeit $1 - 2^{-2^t}$ geschieht. Also ist

$$\mathrm{P}[\mathrm{time}_C(x) < \log n] \;\le\; 1 - 2^{-n}\;.$$

Für $T = \mathrm{llog} - 1$ ist die inverse Funktion $T^{-1}(t) = 2^{2^{t+1}}$. Somit können wir die folgende Abschätzung angeben.

$$\begin{aligned}\sum_{t\ge 1} \frac{T^{-1}(t)}{n}\quad \mathrm{P}[\mathrm{time}_C(x) = t] \;&\ge\; \frac{T^{-1}(\log n)}{n}\,\mathrm{P}[\mathrm{time}_C(x) \ge \log n]\\ &\ge\; \frac{2^{2^{\log n+1}}}{n}\,2^{-n} \;:=\; \exp(2n - \log n - n) > 1\;.\end{aligned}$$

□

Der rechte Schaltkreis in der Abbildung 4.2 hat fast optimale erwartete Zeit $\le 2,3$, logarithmische Tiefe und eine $\mathcal{O}(\mathrm{llog}n)$-Average-Zeit. Damit vereinigt er alle wünschenswerten Zeiteigenschaften.

Interessanterweise dreht sich für sublineare Zeitschranken die Beziehung zwischen Average und Erwartungswert um:

Lemma 10 *Sei T eine monotone Komplexitätsschranke so dass die Funktion $n \mapsto \frac{n}{T(n)}$ monoton zunimmt. Dann gilt*

$$(t, \mu) \in \text{Av}(T) \Longrightarrow \mathbf{E}_\mu(t) \leq 2T(n) \; .$$

Damit folgt

$$\text{AvCirTime}(f, D) \leq T \Longrightarrow \text{ECirTime}(f, D) \leq 2T(n) \; .$$

Beweis: Nach der Definition von $(f, \mu_n) \in \text{Av}(T)$ gilt

$$\sum_{|x|=n} \mu_n(x) \frac{T^{-1}(f(x))}{n} \; \leq \; 1 \; .$$

Wir betrachten eine Partition I_1, I_2 der Eingabemenge $\{0,1\}^n$

$$\begin{aligned} I_1 &:= \{x \in \{0,1\}^n | \; f(x) \; > \; T(n)\} \; , \\ I_2 &:= \{x \in \{0,1\}^n | \; f(x) \; \leq \; T(n)\} \; . \end{aligned}$$

I_1: Falls $f(z) > T(n)$ dann gilt $T^{-1}(f(z)) \geq T^{-1}(T(n)+1) \geq n$. Für die monoton zunehmende Funktion $g(n) := n/T(n)$ kann man schlussfolgern

$$g(T^{-1}(f(z))) \; \geq \; g(n) \; \Longrightarrow \; \frac{T^{-1}(f(z))}{n} \; \geq \; \frac{f(z)}{T(n)} \; .$$

Dann gilt

$$\sum_{x \in I_1} \mu_n(x) \frac{f(x)}{T(n)} \; \leq \; \sum_{x \in I_1} \mu_n(x) \frac{T^{-1}(f(x))}{n} \; \leq \; 1 \; .$$

I_2: $f(x) \leq T(n)$ für $x \in I_2$ impliziert

$$\sum_{x \in I_2} \mu_n(x) \frac{f(x)}{T(n)} \; \leq \; 1$$

und daher

$$\sum_{|x|=n} \mu_n(x) \frac{f(x)}{T(n)} \; \leq \; 2 \; .$$

□

4.4 Die Komplexität von Verteilungen

Für Schaltkreise müssen wir analog zu Sampleable und Computable eigene Komplexitätsklassen für Wahrscheinlichkeitsverteilungen definieren. Hierfür wird ein schaltkreis-basiertes Modell verwendet werden, die der Klasse Sampleable nachempfunden ist.

Definition 15 *Ein Schaltkreis C mit r Eingabebits und n Ausgabebits transformiere die uniforme Zufallsvariable Z aus $\{0,1\}^r$ zur Zufallsvariable X aus $\{0,1\}^n$ wie folgt. Der Eingabevektor wird uniform gemäß Z gewählt. Dann entspricht X den Ausgaben des Schaltkreises C.*

So ein Schaltkreis werde verteilungserzeugender Vorschaltkreis **distribution generating circuit, DG-circuit** *genannt.*

Man erhält keine interessanten Resultate, wenn man nicht auch eine Ausgradbeschränkung der DG-Schaltkreise fordert, siehe hierzu Abbildung 4.3. Dort sieht man einen Vorschaltkreis (mit $d(n) = 0$) konstanter Tiefe, der für die Disjunktion zu einer Worst-Case-Eingabe verteilung führt.

Somit klassifizieren wir mit Vorschaltkreisen wie folgt.

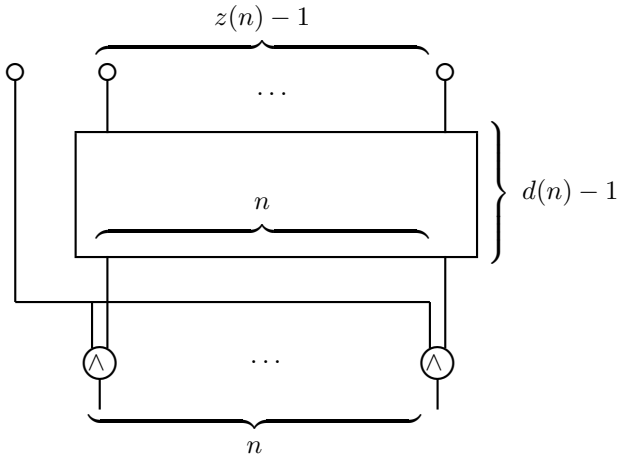


Abbildung 4.3: Ein Vorschaltkreis mit unbeschränkten Grad erzeugt zu schwierige Eingaben

Definition 16

$\mathcal{D}Depth_n(d) := \{\ \mu \in \mathcal{D}_n \mid \exists$ *ein* r*-Eingabe und* n*-Ausgabe DG-Vorschaltkreis* C *der Tiefe* d*, der die Verteilung* X *aus* $\{0,1\}^n$ *mit Hilfe von Zufallsbits erzeugt* $\}$.

Das folgende Lemma zeigt die beiden wesentlichen Techniken zum Beweis von oberen Schranken für solche Klassen auf.

Lemma 11 *Sei* X *aus* $\mathcal{D}Depth_n(d)$ *und definiere*

$$I_X := \{i \mid 0 < \ \mathrm{P}[X_i = 1] \ < 1\} .$$

Dann gilt

1. *Für alle* $i \in I_X$ *gilt*
$$2^{-2^d} \ \leq \ \mathrm{P}[X_i = 1] \ \leq \ 1 - 2^{-2^d} ,$$
2. *Es gibt eine Menge von* $k \geq |I_X| 2^{-2d}$ *unabhängigen Zufallsbits* $X_{i_1}, \ldots, X_{i_k}$.

Beweis: Der erste Teil folgt über eine Induktion über die Vorschaltkreistiefe. Für den zweiten Teil überlege man sich, dass jedes Ausgabebit von höchstens 2^d Zufallsbits abhängt. Jedes Zufallsbit kann aber aufgrund der Ausgradbeschränkung nur 2^d Ausgabebits beeinflussen.

Damit muss es $|I_X| 2^{-2d}$ unabhängige Bitpositionen in I_X geben. □

4.5 Obere und untere Schranken für Grundlegende Boolesche Funktionen

4.5.1 Funktionen mit sublogarithmischer Average-Zeit

An dieser Stelle zeigen wir eine Reihe von Funktionen, die sowohl im Erwartungswert als auch im Average effizienter als im Worst-Case realisiert werden können.

Theorem 21 *Für* $0 \leq d < \mathrm{llog} n$ *gilt*

$$2^{d-1} - 1 \ \leq \quad \mathrm{ECirTime}(\mathtt{OR}_n, \mathcal{D}Depth(d)) \quad \leq \ 2^d + 3 ,$$

$$\mathrm{llog} n + 2^d - 2 \ \leq \quad \mathrm{AvCirTime}(\mathtt{OR}_n, \mathcal{D}Depth(d)) \quad \leq \ 2\mathrm{llog} n + 2^{d+1} + 4d + 2 .$$

Beweis: Für die untere Schrank betrachte die Wahrscheinlichkeitsverteilung $\mathrm{P}[X_i = 1] := 2^{-2^d}$ mit unabhängigen Bits X_i. Ein einfacher Vorschaltkreis erzeugt dies Verteilung in Zeit d. Die Wahrscheinlichkeit dass eine beliebige Eingabe der Länge 2^{2^d-1} nur aus Nullen besteht ist dann mindestens $\frac{1}{2}$.

Um aber eine Ausgabe zu generieren muss man all diese Eingaben berücksichtigen. Wegen der Eingradbeschränkung der Schaltkreise folgt dann sofort die untere Schranke

$$\mathrm{ECirTime}(\mathtt{OR}_n, \mathcal{D}\mathrm{Depth}(d)) \;\geq\; \frac{1}{2}(2^d - 1) \;\geq\; 2^{d-1} - 1.$$

Die folgende Betrachtung für die gleiche Zufallsvariable X ergibt die untere Schranke für das Average-Maß. Note, that

$$\mathrm{P}[X = 0^n] \;:=\; (1 - 2^{-2^d})^n \;\geq\; \exp\left(-2\frac{n}{2^{2^d}}\right) .$$

Sei C ein Schaltkreis der $\mathtt{OR}_n$ berechnet. Angenommen, $\mathrm{AvCirTime}(\mathtt{OR}_n, \mathcal{D}\mathrm{Depth}(d)) \leq T = \mathrm{llog}n + 2^d - 2$. Dann gilt

$$\begin{aligned}\sum_t \frac{T^{-1}(t)}{n}\,\mathrm{P}[\mathrm{time}_C(X) = t] \quad &\geq \quad \frac{T^{-1}(\log n)}{n}\,\mathrm{P}[\mathrm{time}_C(X) = \log n] \\ &\geq \quad \frac{\exp\exp(\log n - 2^d + 2)}{n\,\exp\left(2\frac{n}{2^{2^d}}\right)} \;>\; 1\,,\end{aligned}$$

da $n/2^{2^d} \geq \log n$ für $d < \mathrm{llog}n$.

Für die obere Schranken müssen wir eine beliebige Zufallsvariable X über $\{0,1\}^n$ in $\mathcal{D}\mathrm{Depth}(d)$ betrachten. Angenommen, $X \in \mathcal{D}\mathrm{Depth}_n(d)$ ist positive, d.h. keine Eingabe wird mit Wahrscheinlichkeit 0 erzeugt.

Nach Lemma 11 gibt es dann mindestens $n2^{-2d}$ unabhängige Eingabebits X_i. Wir unterteilen sie in Gruppen Y_j der Größe 2^{2^d} und berechnen für jede Y_j die Disjunktion $\mathtt{OR}(Y_j)$ aller Bits innerhalb von Tiefe 2^d. Man beachte, dass $\mathrm{P}[Y_i = 0^{2^d}] \leq (1 - \exp(-2^d)^{2^d} < 2^{-1}$.

Seien nun W die übrig gebliebenen Bits. Die Disjunktion über diese wird durch einen balancierten Binärbaum der Tiefe $\log n$ berechnet. Somit realisieren wir die Oder-Funktion durch

$$\mathtt{OR}(X) \;:=\; \mathtt{OR}(Y_1) \vee (\mathtt{OR}(Y_2) \vee (\cdots(\mathtt{OR}(Y_p) \vee \mathtt{OR}(W))\cdots)) \,,$$

wobei $p \geq n2^{-2d-2^d} \geq 1 + \mathrm{llog}n$ für genügend große n.

Sei $\mathrm{time}(X)$ die Zufallsvariable, welche das Zeitverhalten des Schaltkreises$C$ beschreibt. Wegen Lemma 11 gilt $\mathrm{P}[X_i = 1] \geq 2^{-2^d}$ und somit

$$\mathrm{P}[\mathtt{OR}(Y_j) = 1] \geq 1 - (1 - 2^{-2^d})^{2^{2^d}} \;\geq\; \frac{1}{2}\,.$$

Darus folgt für $1 \leq t \leq \mathrm{llog}n + 1$

$$\mathrm{P}[\mathrm{time}(X) = 2^d + t] \;\geq\; 2^{-t}\,.$$

Hieraus folgt sofort, dass der Erwartungswert von $\mathrm{time}_C(X)$ beschränkt ist durch $2^d + 3$.

$$E(\mathrm{time}_C) \;\leq\; \sum_{t=1}^{\mathrm{llog}_{n+1}} \frac{2^d + t}{2^t} + \frac{\log n + \mathrm{llog}n + 1}{2^{\mathrm{llog}_{n+1}}} \;\leq\; 2^d + \sum_{t=1}^{\mathrm{llog}_{n+1}} \frac{t}{2^t} + 1 \;\leq\; 2^d + 3.$$

Verbinden wir nun die Schaltkreise $\mathtt{OR}(Y_i)$ und $\mathtt{OR}(W)$ als Eingaben des rechten Schaltkreises der Abbildunge 4.2 ($\mathtt{OR}(\mathtt{OR}(Y_1), \ldots, \mathtt{OR}(Y_p), \mathtt{OR}(W))$) ergibt die folgende Rechnung

$$\sum_{i=1}^{p} \frac{T^{-1}(2\log i + 1 + 2^d)}{n2^{i-1}} + \frac{T^{-1}(2\log n + 1)}{n2^p} \;\leq\; 1.$$

Somit ist eine obere Schranke für das Average-Maß gegeben durch

$$T \;\leq\; 2(\text{llog}n + 2d + 2^d + 1).$$

Falls X nicht positiv ist, gibt es entweder eine Eingabe X_i with $\mathrm{P}[X_i = 1] = 1$, womit das Ergebnis immer 1 ist, oder die Eingabe X_i ist 0 mit Wahrscheinlichkeit 1 und somit können wir sie einfach weglassen. □

4.5.2 Die Parität als schwierige Funktion

Neben der Disjunktion gibt es ein Reihe anderer Funktionen, die man mit Schaltkreisen im Average schneller berechnen kann als im Worst-case. Jedoch gibt es auch sehr elementare Funktion, für die die Best-Case-Zeit der Worst-Case-Zeit entspricht. Ein prominentes Beispiel ist die Paritätsfunktion PARITY. Intuitiv ist das einfach einzusehen. Schließlich hängt die Ausgabe von jedem Eingabebit ab und aufgrund der Eingradbeschränkung lässt sich diese Funktion deshalb nicht schneller berechnen.

Definition 17 *Um den Grad der Eingabeabhängigkeit zu messen definieren wir* depen(f) *von* input bit dependency *einer n-stelligen Booleschen Funktion als*

$$\text{depen}(f) \;:=\; \min\{k \mid \exists\, a_{i_1}, \ldots, a_{i_k} \textit{ so dass } f_{|\, x_{i_1}=a_{i_1},\ldots,x_{i_k}=a_{i_k}} \equiv \text{const}\}$$

das ist die minimale Länge eines Primimplikanden oder einer Primklausel von f.

Funktionen in diesem Zusammenhang sind die Schwellwertfunktion $\texttt{THRESHOLD}^a_n$, welche genau dann 1 ist, wenn mindestens a Einser an der Eingabe vorliegen. Die Mehrheitsfunktion $\texttt{MAJORITY}_n$ ist der Spezialfall $a = \lfloor \frac{n}{2} \rfloor$. $\texttt{BITSORT}_n$ beschreibt die Sortierfuntion von n Bits und $\texttt{MULTIPLY}_n$ die Multiplikation von zwei n-Bit-Zahlen. Für diese Funktionen läßt sich depen sehr genau bestimmen.

Lemma 12

$$\begin{aligned}
\text{depen}(\texttt{OR}_n) &= 1\,, \\
\text{depen}(\texttt{PARITY}_n) &= n\,, \\
\text{depen}(\texttt{MULTIPLY}_n) &\geq \sqrt{n}\,, \\
\text{depen}(\texttt{THRESHOLD}^a_n) &= \min(a, n-a+1)\,, \\
\text{depen}(\texttt{MAJORITY}_n) &= \lceil \tfrac{n}{2} \rceil\,, \\
\text{depen}(\texttt{BITSORT}_n) &= n\,.
\end{aligned}$$

Der Zusammenhang zur Zeit in Schaltkreisen ergibt sich durch das folgende Theorem.

Lemma 13 *Für $f \in B_n$ sei $C \in \mathit{Cir}(f)$. Dann gilt* $\forall x \in \{0,1\}^n \quad \text{time}_C(x) \;\geq\; \log\, \text{depen}(f)\,.$

Beweis: Sei $x \in \{0,1\}^n$ beliebig. Nach der Definition von depen(f) bestimmt keine Teilmenge von weniger als depen(f) viele Eingabebits nicht die Ausgabe von $f(x)$.

Da der Eingrad der Schaltkreise mit 2 beschränkt ist, muss jede Berechung von C in weniger als $\log \text{depen}(f)$ Schritten Eingaben unberücksichtigt lassen, die notwendig zur korrekten Berechnung waren. □

Daraus folgen die folgenden asymptotisch engen unteren Schranken.

Theorem 22

$$\begin{aligned}
\text{ECirTime}(\texttt{PARITY}_n, \mu_{\text{uni}_n}) &\geq \log n\,, \\
\text{ECirTime}(\texttt{MAJORITY}_n, \mu_{\text{uni}_n}) &\geq \log n - 1\,, \\
\text{ECirTime}(\texttt{BITSORT}_n, \mu_{\text{uni}_n}) &\geq \log n\,, \\
\text{ECirTime}(\texttt{MULTIPLY}_n, \mu_{\text{uni}_n}) &\geq \tfrac{1}{2}\log n\,, \\
\text{AvCirTime}(\texttt{PARITY}_n, \mu_{\text{uni}_n}) &> \log n - 1\,, \\
\text{AvCirTime}(\texttt{MAJORITY}_n, \mu_{\text{uni}_n}) &> \log n - 2\,, \\
\text{AvCirTime}(\texttt{BITSORT}_n, \mu_{\text{uni}_n}) &> \log n - 1\,, \\
\text{AvCirTime}(\texttt{MULTIPLY}_n, \mu_{\text{uni}_n}) &> \tfrac{1}{2}\log n - 1\,.
\end{aligned}$$

4.5.3 Die Schwellwertfunktion

Diese Funktion ist wegen ihres Schwellwertparameters $a \in \{0, \ldots, n\}$ besonders interessant. Ist dieser nämlich an den Randstellen bei 0 oder n, so können wir diese Funktion sehr effizient im Durchschnitt berechnen. Ist er in der Mitte, dann ist diese Funktion die Mehrheitsfunktion und damit genauso schwer. Tatsächlich kann man folgende Aussage treffen:

Theorem 23 *Für $d \leq \text{llog}n - 1$ und $a \leq n2^{-(2^d+2d+O(1))}$ gilt*

$$\tfrac{1}{2}\,(\log a + 2^d) - O(1) \qquad \leq \text{ECirTime}(\texttt{THRESHOLD}^a_n, \mathcal{D}\textit{Depth}(d)) \leq\ 2(\log a + 2^d) + O(1)\,,$$
$$\tfrac{1}{2}(\text{llog}n + \log a + 2^d) - O(1) \leq \text{AvCirTime}(\texttt{THRESHOLD}^a_n, \mathcal{D}\textit{Depth}(d)) \leq 2\,(\text{llog}n + \log a + 2^d) + O(1).$$

4.5.4 Addition zweier Zahlen

Addition
Geg.: zwei n-stellige Binärzahlen x, y
Ges.: $z = x + y$

Folgendes Beispiel zeigt die Addition von zwei 10-stelligen Zahlen x und y. Zuerst wird der Übertrag c berechnet. Mit Hilfe des Übertrags c berechnet sich die Summe für alle k als

$$z[k] = x[k] \oplus y[k] \oplus c[k]\,,$$

wobei $a \oplus b$ die Paritätsfunktion (exklusive Oder-Funktion, *Xor*) $a + b \bmod 2$ darstellt.

Binäre Addition

k	10	9	8	7	6	5	4	3	2	1	0
$x[k]$	–	1	0	1	0	1	0	0	1	1	0
$y[k]$	–	1	0	0	1	1	0	1	0	0	0
$c[k]$	1	0	1	1	1	0	0	0	0	0	0
$z[k]$	1	0	1	0	0	0	0	1	1	1	0
$w[k]$	s	g	s	p	p	g	s	p	p	p	s

Wie das Beispiel zeigt, unterliegt der Übertrag folgenden Regeln:

- *generate*: $w[k] = \texttt{g}$
  Wenn $x[k] = y[k] = 1$, dann wird ein Übertrag $c[k+1] = 1$ erzeugt.
- *stop*: $w[k] = \texttt{s}$
  Ist $x[k] = y[k] = 0$, dann entsteht kein Übertrag: $c[k+1] = 0$.
- *propagate*: $w[k] = \texttt{p}$. Ist $x[k] \neq y[k]$, so entsteht ein Übertrag $c[k+1] = 1$ genau dann, wenn $\exists \ell : w[k-\ell] = \texttt{g}$ und $w[k] = w[k-1] = \cdots = w[k-\ell+1]$.

Wir definieren also die Funktionen $m : \{0,1\}^2 \to \{\texttt{s},\texttt{p},\texttt{g}\}$, $\odot : \{\texttt{s},\texttt{p},\texttt{g}\}^2 \to \{\texttt{s},\texttt{p},\texttt{g}\}$ und $t : \{\texttt{s},\texttt{p},\texttt{g}\} \to \{0,1\}$:

m	0	1
0	s	p
1	p	g

$\odot$	s	p	g
s	s	s	g
p	s	p	g
g	s	g	g

$t : \texttt{s} \mapsto 0$
$\texttt{p} \mapsto 0$
$\texttt{g} \mapsto 1$

Nach dieser Tabelle gilt also z.B. $\texttt{g} \odot \texttt{s} = \texttt{s}$. Wir stellen das niedriger wertige Element dem höher wertigen voran, also $w[0] \odot w[1]$. Diese Unterscheidung ist wichtig, da $\odot$ nicht kommutativ ist.

Fakt 1 *Die Operation $\odot$ ist assoziativ.*

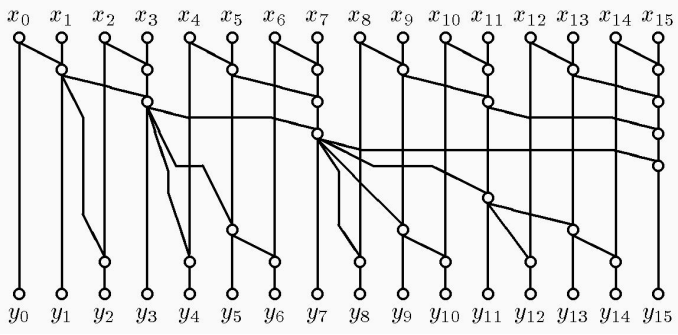


Abbildung 4.4: Ladner-Fischer-Schaltkreis für die parallele Präfixberechnung.

Der $k+1$-te Übertrag berechnet sich mittels dieser Operation nun durch

$$c(k+1) = t(w[0] \odot w[1] \odot \cdots \odot w[k]) \; .$$

Theorem 24 *[LF80] Durch den Parallel-Präfix-Schaltkreis nach Ladner und Fischer kann man Addition mit Schaltkreisen der Größe $\mathcal{O}(n)$ und der Tiefe $\mathcal{O}(\log n)$ berechnen.*

Beweis: Hierfür verwendet man den in Abbildung 4.4 dargestellten Schaltkreis nach Ladner und Fischer. □

Erwartet schnelle Addition zweier Zahlen

Nehmen wir nun an, daß jedes Bit von x und y mittels eines perfekten Münzwurf bestimmt worden sei. Die Laufzeit eines erwartet schnellen Algorithmus hängt maßgeblich von der Länge einer Kette von *propagate*-Zuständen ab.

Die Wahrscheinlichkeit einer Folge von ℓ *propagate*-Einträgen läßt sich wie folgt abschätzen:

$$\begin{aligned} \mathrm{P}[\exists k : w[k+1] = w[k+2] = \cdots = w[k+\ell] = \mathtt{p}] &\leq n \cdot \mathrm{P}[w[1] = w[2] = \cdots = w[\ell] = \mathtt{p}] \\ &= n \cdot 2^{-\ell} \; . \end{aligned}$$

Wählt man $\ell = \log n + \log\log n$, so gilt

$$\mathrm{P}[\exists k : w[k+1] = w[k+2] = \cdots = w[k+\ell] = \mathtt{p}] \; \leq \; \frac{1}{\log n} \; .$$

Theorem 25 *Für $d < \mathrm{llog}n$ gilt*

$$\begin{aligned} \frac{1}{4}(\mathrm{llog}n + 2^d) \; &\leq \; \mathrm{ECirTime}(\mathtt{ADDITION}_n, \mathcal{D}\mathit{Depth}(d)) \; \leq \; O(\mathrm{llog}n + 2^d) \; , \\ \mathrm{llog}n + 2^d - 2 \; &< \; \mathrm{AvCirTime}(\mathtt{ADDITION}, \mathcal{D}\mathit{Depth}(d)) \; \leq \; O(\mathrm{llog}n + 2^d) \; . \end{aligned}$$

Beweis: Wir beschränken uns hier auf den Beweis der oberen Schranke des Erwartungswerts. Für eine Analyse der anderen Maße sei auf [JRSW94] verwiesen.

Für die Berechnung betrachte man das Rechennetzwerk aus Abbildung 4.5, welches in Abbildung 4.6 als Schaltkreis dargestellt worden ist. Man erkennt, dass jedes Ausgabe durch einen Baum erzeugt wird, welcher an den Average-effizienten Oder-Schaltkreis erinnert. Insbesondere kann die Berechnung in Tiefe $2d$ vollendet werden, wenn keine p Kette der Länge d als Eingabe anliegt.

Nach obiger Wahrscheinlichkeitsabschätzung besteht ein Eingabe-Intervall der Länge $\ell = \log n + \log\log n$ aus Werten p mit Wahrscheinlichkeit $\leq \frac{1}{\log n}$.

Funktion	Menge von W'keitsverteilungen	Tiefe	ECirTime	AvCirTime
OR, AND,	uniform	$\log n$	$2 - \frac{1}{2^{n-2}}$	$\Theta(\log\log n)$
OR, AND, EQUAL	$\mathcal{D}$Depth(d)	$\log n$	$\Theta(\min(2^d, \log n))$	$\Theta(\min(\log\log n + 2^d, \log n))$
PARITY, MAJORITY	uniform	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$
ADDITION	$\mathcal{D}$Depth(d)	$\Theta(\log n)$	$\Theta(\min(\log\log n + 2^d, \log n))$	$\Theta(\min(\log\log n + 2^d, \log n))$
THRESHOLD$_a^n$	$\mathcal{D}$Depth(d)	$\Theta(\log n)$	$\Theta(\min(\log a + 2^d, \log n))$	$\Theta(\min(\log\log n + \log a + 2^d, \log n))$

Tabelle 4.1: Zusammenfassung der erwarteten und Average-Komplexität elementarer Funktionen.

Die erwartete Laufzeit ist also für eine geeignete Konstante c

$$\begin{aligned} E(\text{time}) &\leq c \cdot \sum_{i=1}^{\log n} i \cdot \mathrm{P}[\exists k : w[k+1] = \cdots = w[k+2^{i-2}] = \mathtt{p}] \\ &\leq c(2 + \log \ell) + c \cdot \log n \cdot \mathrm{P}[\exists k : w[k+1] = \cdots = w[k+\ell] = \mathtt{p}] \\ &\leq c(2 + \log \ell) + c \\ &= O(\log \ell) = O(\log\log n) \end{aligned}$$

□

Für die untere Schranke benutzt man das folgende Lemma.

Lemma 14 *Für jeden Schaltkreis C der das parallele Präfixproblem löst und jede Eingabe z gilt*

$$\text{time}_C(z) \;\geq\; \log \mathtt{pro}(z)\,.$$

Hierbei bezeichnet $\mathtt{pro}(z)$ die Länge des größten Teilstrings $\mathtt{p}^*$ in z. Da mit konstanter Wahrscheinlichkeit tatsächlich eine p-Kette der Länge $\log n$ auftaucht, folgt daraus die untere Schranke für den Erwartungswert der Schaltkreiszeit für die Addition.

Ähnlich wie bei den oben gezeigten Booleschen Funktionen läßt sich dieses Ergebnis auf mit Vorschaltkreisen generierten Wahrscheinlichkeitsverteilungen verallgemeinern. Verlangt man zusätzlich, dass alle Eingaben positive Wahrscheinlichkeit besitzen, so erreicht der oben dargestellte Schaltkreis für alle Wahrscheinlichkeitsverteilungen die asymptotisch optimale erwartete Laufzeit und Average-Laufzeit.

4.5.5 Überblick

In der folgenden Tabelle 4.1 sind die Ergebnisse noch einmal zusammengefasst.

4.5.6 Schaltkreise haben keine bösartigen Wahrscheinlichkeitsverteilungen

Bösartige Wahrscheinlichkeitsverteilungen haben wir schon weiter oben vorgestellt. Interessanterweise gibt es diese in Schaltkreisen nicht. Es gibt nämlich für jede Wahrscheinlichkeitsverteilung eine nichttriviale Disjunktion, die nur von wenigen Variablen abhängt, die mit großer Wahrscheinlichkeit den Wert 1 ergibt. Sei $x, \alpha \in \{0,1\}$, und $x^1 := x$ und $x^0 = \neg x$.

Lemma 15 *Sei $X = x_1, \ldots, x_n$ eine Zufallsvariable über $\{0,1\}^n$ beliebig verteilt und sei $\{i_1, i_2, \ldots, i_l\} \subseteq [1..n]$. Dann existieren Boolesche Konstanten $\alpha_1, \ldots, \alpha_l$ so dass $\mathrm{P}[\mathtt{OR}(x_1^{\alpha_1}, \ldots x_l^{\alpha_l}) = 1] \;\geq\; 1 - 2^{-l}$.*

Beweis: durch eine Induktion über die Eingabelänge: Sei $\alpha_1 = 1$ falls $\mathrm{P}_\mu[x_1 = 1] \geq \frac{1}{2}$, und ansonsten $\alpha_1 = 0$. Es ist einsichtig, dass für alle $\alpha_1, \ldots, \alpha_k$ es eine Konstante $\alpha_{k+1} \in \{0,1\}$ gibt so dass

$$\mathrm{P}_\mu[x_{k+1}^{\alpha_{k+1}} = 1 \mid x_1^{\alpha_1}, \ldots, x_k^{\alpha_k} = 0] \;\geq\; 1/2$$

oder

$$\mathrm{P}_\mu[x_1^{\alpha_1} \vee \ldots \vee x_k^{\alpha_k}] = \; 1$$

gilt.

Sei $\mathrm{P}_\mu[x_1^{\alpha_1} \vee \ldots \vee x_k^{\alpha_k}] \geq 1 - 2^{-k}$, dann ist

$$\begin{aligned} \mathrm{P}_\mu[x_1^{\alpha_1} \vee \ldots \vee x_{k+1}^{\alpha_l}] &= (1 - \mathrm{P}_\mu[x_1^{\alpha_1} \vee \ldots \vee x_k^{\alpha_k}])/2 \;+\; \mathrm{P}_\mu[x_1^{\alpha_1} \vee \ldots \vee x_k^{\alpha_k}] \\ &\geq 1 - 2^{-k-1} \,. \end{aligned}$$

□

Mit dieser Eigenschaft beweisen wir dieses Theorem.

Theorem 26 *Für jede Wahrscheinlichkeitsverteilung μ über $\{0,1\}^n$ existierte eine n-stellige Boolesche Funktion f mit*

$$\mathrm{depth}(f) \;\geq\; n - \log n - \mathrm{llog}n \quad \text{and} \quad \mathrm{ECirTime}(f, \mu) \;\leq\; 4 \,.$$

Beweis: Für ein gegebenes μ werde die Funktion f wie folgt definiert. Sei $\ell := \log n$. Bezüglich μ werden die Booleschen Werte $\alpha_1, \ldots, \alpha_l$ gemäß des obigen Lemmas gewählt.

Wir wählen nund $g \in B_{n-\log n}$, so dass $g \notin \mathrm{CirDepth}(n - \log n - \mathrm{llog}n - 1)$. Gemäß der Shannon-Schranke muss eine solche Funktion existieren (Übungsaufgabe!). Sei nun

$$f(x_1, \ldots, x_n) \;:=\; \mathtt{OR}(x_1^{\alpha_1}, \ldots, x_l^{\alpha_l}, g(x_{l+1}, \ldots, x_n)) \,.$$

Um die untere Schranke zu beweisen, nehmen wir an, dass $\mathrm{depth}(C) < n - \log n - \mathrm{llog}n$ and $C \in \mathrm{Cir}(f)$. Wir ersetzen den Eingabevektor $(x_1, \ldots, x_l)$ durch die Eingaben $(\neg\alpha_1, \ldots, \neg\alpha_l)$ und erhalten Schaltkreis C', für desen Tiefe gilt: $\mathrm{depth}(C') = \mathrm{depth}(C)$ und $C' \in \mathrm{Cir}(g)$. Dies widerspricht der Shannon-Schranke.

Für die obere Schranke wird die OR-Teilfunktion mit einen Average-optimalen Schaltkreis realisiert, deren Erwartungswert durch 2 beschränkt ist [JRS94]. Dieser ist gegeben durch

$$C_{\mathtt{OR},n}(x_1, \ldots, x_n) := \mathtt{OR}(x_n, C_{\mathtt{OR},n-1}(x_1, \ldots, x_{n-1})) \,.$$

Für g versenden den tiefenoptimalen Schaltkreis mit Tiefe $n - \log n - \mathrm{llog}n + O(1)$. Dann kann die erwartete Schaltkreiszeit abgeschätzt werden durch

$$\begin{aligned} E_\mu(\mathrm{time}_C) &\leq 1 \;+\; \mathrm{P}_\mu[\mathtt{OR}(x_1^{\alpha_1}, \ldots, x_l^{\alpha_l}) = 1]2 \\ &\quad +\quad \mathrm{P}_\mu[\mathtt{OR}(x_1^{\alpha_1}, \ldots, x_l^{\alpha_l}) = 0](\mathrm{depth}(g) + \log n) \\ &\leq 1 + 2 + n/n \;=\; 4 \,. \end{aligned}$$

□

Somit hat die Klasse aller n-stelligen Booleschen Funktionen keine bösartigenWahrscheinlichkeitsverteilungen.

4.5.7 Worst-Case Wahrscheinlichkeitsverteilungen

Zwar kann man keine universell schlechte Wahrscheinlichkeitsverteilung für die Schaltkreistiefe, jedoch kann man für jede Funktion eine finden, welche relativ lange erwartete Schaltkreiszeiten provoziert. In [JRS95] wurde folgendes Resultat hierzu bewiesen.

Theorem 27 *Für jedes $t \in \mathbb{N}$ gilt:*

$$\begin{aligned} &\mathrm{ECirTime}\left(t, \mathcal{D}\mathit{Depth}\left(\frac{t+n}{2} + \sqrt{n-t} + \log n + 3\right)\right) \\ &\subseteq \mathrm{CirDepth}(t + \log n + \log t + 3) \,. \end{aligned}$$

Das folgende Korollar ist äquivalent und verdeutlicht die Aussage dieses Theorems, dessen Beweis wir hier aus Zeitgründen weglassen müssen.

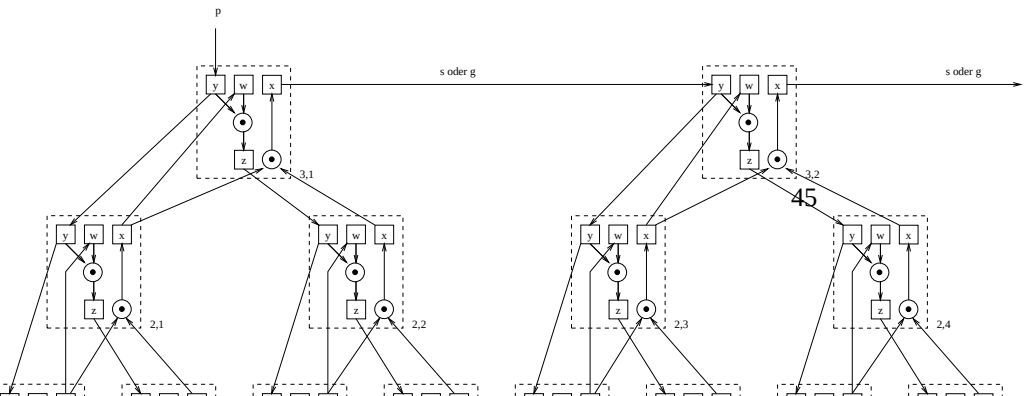
Korollar 1 *Für jede Boolesche Funktion $f \in \text{CirDepth}(d)$ gibt es eine Wahrscheinlichkeitsverteilung μ_f, die in Tiefe $\frac{1}{2}(n+d) + \sqrt{n-d} + O(\log n)$ berechnet werden kann, so dass alle Schaltkreise für f eine Schaltkreiszeit von mindestens $d - \log n - \log d - 3$ bezüglich μ_f besitzen.*

4.5.8 Die Komplexität der Zeitberechnung von Schaltkreisen

In Tabelle 4.2 findet man einen Überblick über die Komplexität verschiedener Fragestellungen im Zusammenhang mit Schaltkreiszeit. Als Übungsaufgabe mache man sich mit der Bedeutung dieser Komplexitätsklassen vertraut. Die dargestellten Ergebnisse finden sich in [JS96].

Gegeben	Gesucht	Obere Schranke	Untere Schranke
C, x, t	$\text{time}_C(x) \leq t$	$\mathcal{P}$-complete	
C, t	$\exists x\ \text{time}_C(x) \leq t$	$\mathcal{NP}$-complete	
C, t	$t_{\mathtt{wc}}(C) = t$	$\mathcal{BH}_2$-complete	
C	Finde x so, dass $\text{time}_C(x) = t_{\mathtt{wc}}(C)$	$\mathcal{FP}^{\mathcal{NP}}$	$\mathcal{FP}^{\mathcal{NP}}_{\mathtt{tt}}$-hard
C	lexikographisch maximales x mit $\text{time}_C(x) = t_{\mathtt{wc}}(C)$	$\mathcal{FP}^{\mathcal{NP}}$-complete	
C, D	$\mu_D =$ worst case W'keitsverteilung für C	co-$\mathcal{NP}$-complete	
C, D	$E_{\mu_D}(\text{time}_C)$	#$\mathcal{P}$-complete	

Tabelle 4.2: Obere und untere Schranken für die Komplexität von Problemen im Zusammenhang mit der Schaltkreiszeit. C beschreib einen Schaltrkreis, D einen Vorschaltkreis, x eine Eingabe und t eine Zeitschranke.


p
s oder g
s oder g
y
w
x
z
3,1
3,2
2,1
2,2
2,3
2,4

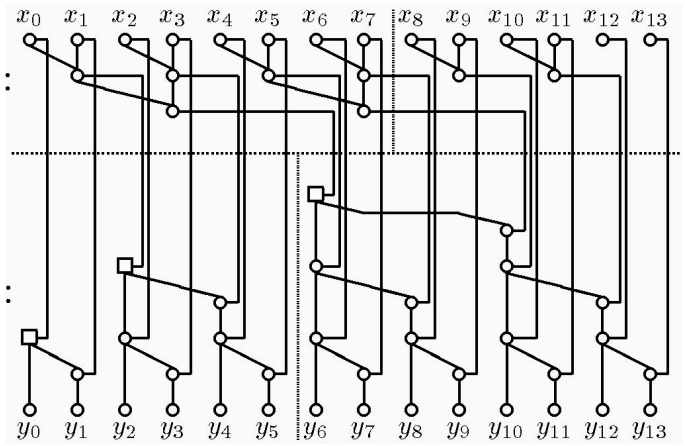


Abbildung 4.6: Average-effizienter Schaltkreis zur effizienten Berechnung der parallelen Präfixfunktion.

Kapitel 5

Ein erwartet schneller, fast korrekter Sortierer

In diesem Kapitel werden wir einen Schaltkreis vorstellen, der mit hoher Wahrscheinlichkeit Eingaben schnell sortieren kann. Aus den Überlegungen im letzten Kapitel folgt sofort, dass ein solcher Schaltkreis immer mindestens Zeit $\log n$ benötigt, da die Dependancy von Bit-Sortieren n ist. Andererseits hat das Sortiernetzwerk nach Ajtas, Komlos und Szegedy [AKS83] eine Tiefe von rund $1900 \log n$. Damit geht es in diesem Abschnit "nur" um konstante Faktoren, die aber nicht unerheblich sind.

In der Praxis werden diese asymptotisch optimalen Schaltkreise meist nicht verwendet, da man mit nur unwesentlich schlechterer Asymptotik sehr effiziente einfache Schaltkreise angeben kann. Ein Beispiel hierzu ist der bitonische Sortierer, den wir jetzt kurz vorstellen werden.

5.1 Der bitonische Sortierer

Die Aufgabenstellung ist also das Sortieren von Zahlen:

> **Sortieren von n Zahlen**
> Geg.: n Zahlen $x_1, \ldots, x_n$
> Ges.: Die aufsteigend sortierte Permutation $\Pi(x_1, \ldots, x_n) = (y_1, \ldots y_n)$.

5.1.1 Sortiernetzwerke

Im Gegensatz zu Schaltkreisen verwenden wir Bausteine, die in der Lage sind in einer Zeiteinheit zwei Zahlen zu sortieren.

Definition 18 *Eine* $\mathbf{Komparator}$ *ist ein Baustein mit zwei Eingängen x, y und zwei Ausgängen mit den Ergebnis* $\min(x, y)$ *und* $\max(x, y)$.

Ein $\mathbf{Vergleichsnetzwerk}$ *ist ein azyklischer Graph mit Komparatoren als Knoten. Ein- und Ausgrad aller Knoten ist zwei.*

Ein $\mathbf{Sortiernetzwerk}$ *ist ein Vergleichsnetzwerk, dessen Ausgabefolge monoton zunehmend ist für alle Eingabemöglichkeiten.*

Das 0-1-Prinzip besagt, daß wenn ein Sortiernetzwerk korrekt arbeitet für Eingaben aus $\{0, 1\}$, dann arbeitet es auch für beliebige Zahlen korrekt.

Lemma 16 *Sei f eine monoton zunehmende Funktion. Wenn ein Vergleichsnetzwerk auf Eingabe $x_1, \ldots, x_n$ die Ausgabe $y_1, \ldots, y_n$ produziert, so gibt das Netzwerk auf Eingabe $f(x_1), \ldots, f(x_n)$ die Folge $f(y_1), \ldots f(y_n)$ aus.*

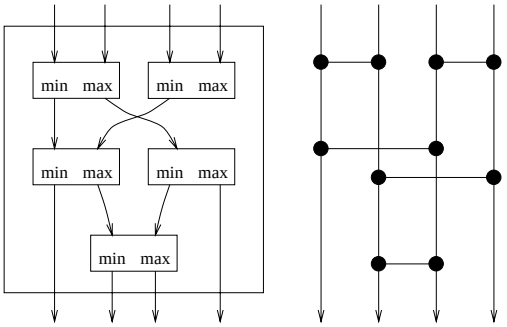


Abbildung 5.1: Ein Vergleichsnetzwerk für vier Eingaben mit alternativer Notation.

Theorem 28 *Wenn ein Vergleichsnetzwerk mit n Eingaben alle 2^n möglichen Folgen aus $\{0,1\}^n$ korrekt sortiert, dann sortiert es auch beliebige Zahlen korrekt.*

Beweis: Für einen Widerspruchsbeweis nehmen wir an, das Netzwerk sortiere alle 0-1-Folgen korrekt, während es eine Eingabe $a_1, \ldots, a_n$ gibt, die nicht korrekt sortiert wird.

Dann gibt es zwei Zahlen $a_i < a_j$, deren Werte falsch herum ausgegeben werden. Definiere f als

$$f(x) \;=\; \begin{cases} 0\,, & \text{falls } x \leq a_i \\ 1\,, & \text{sonst} \end{cases}$$

Dann ergibt obiges Lemma, daß $f(a_i) = 0$ und $f(a_j) = 1$ falsch herum ausgegeben werden, was den Widerspruch ergibt. □

Wir betrachten aus diesem Grund im folgenden nur noch 0-1-Folgen als Eingaben. Zusätzlich schränken wir uns auf Eingabegrößen n ein, die als Zweierpotenz darstellbar sind.

Bitonische Folge

Definition 19 *Eine Folge $a_1, a_2, \ldots, a_n$ heißt* ***bitonisch****, wenn*

1. *die Folge monoton zunehmend ist ($a_1 \leq a_2 \leq \ldots \leq a_n$) oder*
2. *die Folge monoton abnehmend ist ($a_1 \geq a_2 \geq \ldots \geq a_n$) oder*
3. *wenn sie zuerst monoton zunimmt und dann monoton abnimmt ($a_1 \leq a_2 \leq \ldots \leq a_k \geq a_{k+1} \geq \ldots \geq a_n$) oder*
4. *wenn sie zuerst monoton abnimmt und dann monoton zunimmt ($a_1 \geq a_2 \geq \ldots \geq a_k \leq a_{k+1} \leq \ldots \leq a_n$).*

Eine bitonische 0-1-Folge heißt **rein**, gdw. wenn sie konstant 0 oder konstant 1 ist.

Halbreiniger (*half-cleaner*)
Eingabe: Eine bitonische Folge $A = a_1, \ldots a_n$.
Ausgabe: Zwei bitonische Folgen $X = x_1, \ldots, x_{n/2}$ und $Y = y_1, \ldots, y_{n/2}$,
wobei XY eine Permutation von A ist,
X und Y bitonisch sind und eine der beiden Folgen rein ist.

Folgendes Netzwerk (siehe Abb.5.2) aus $n/2$ Komparatoren `comp` und der Tiefe 1 löst diese Aufgabe.

`half-cleaner`$[n]$
input: $a_1, \ldots, a_n$
for $i \in \{1, \ldots, n/2\}$ **in parallel do**
$(x_i, y_i) \leftarrow$ `comp`$(a_i, a_{n/2+i})$

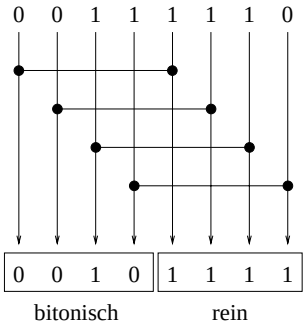


Abbildung 5.2: Ein Halbreiniger für 8-stellige Eingabefolgen.

od
output: $x_1, \ldots, x_{n/2}$
$y_1, \ldots, y_{n/2}$

Lemma 17 *Die Prozedur* `half-cleaner` *ist korrekt.*

Beweis:

1. Die Eingabe ist monoton zunehmend.
   Es findet keine Vertauschung statt. Die eine Hälfte der Eingabe ist schon rein.

2. Die Eingabe nimmt monoton ab.
   Alle Komparatoren führen eine Vertauschung durch. Wiederum war eine Hälfte der Eingabe schon rein.

3. Die Eingabe ist in der Form $\underbrace{0\ldots0}_{i}\underbrace{1\ldots1}_{j}\underbrace{0\ldots0}_{k}$.

   (a) $i \geq n/2$
   Keine Vertauschung findet statt. $X = 0\ldots0$ ist rein. Y bitonisch.

   (b) $i < n/2, i + j \geq n/2, j \geq n/2$
   Nun wird $Y = 1\ldots1$ rein. X bitonisch.

   (c) $i < n/2, i + j \geq n/2, j < n/2$
   $X = 0\ldots0$ ist rein. Y bitonisch.

   (d) $k \geq n/2$
   $X = 0\ldots0$. Y ist bitonisch.

4. Der Fall $1\ldots10\ldots01\ldots1$ ist analog.

□

Sortieren Bitonischer Folgen

Bitonisches Sortieren (*bitonic-sorter*)
Eingabe: Eine bitonische Folge
Ausgabe: Eine sortierte Permutation dieser Folge

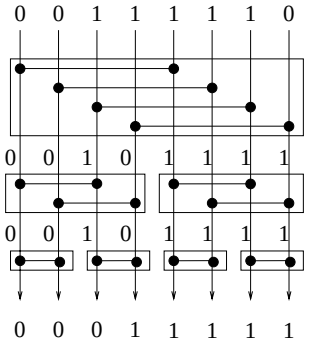


Abbildung 5.3: Ein Sortierer für bitonische Folgen der Länge 8.

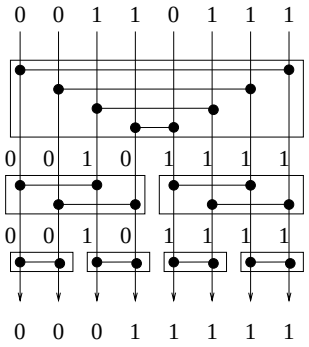


Abbildung 5.4: Ein *merging-network*.

`bitonic-sorter`(n)
input: $a_1, \ldots, a_n$
$m \leftarrow 2n$
for $i \leftarrow 1$ **to** $\log n$ **do**
  $m \leftarrow m/2$
  for $j \in \{1, \ldots, n/m\}$ **in parallel do**
    $(a_{(j-1)m+1}, \ldots, a_{jm}) \leftarrow$ `half-cleaner`$[m](a_{(j-1)m+1}, \ldots, a_{jm})$
  od
od
output: $a_1, \ldots, a_n$

Die resultierende Struktur wird auch **butterfly-network** genannt.

Lemma 18 `bitonic-sorter` *arbeitet korrekt.*

Beweis: Das Vergleichsnetzwerk `bitonic-sorter` hat eine baumähnliche Struktur. Jeder Halbreiniger liefert seine Ausgabe zwei Halbreiniger der halben Größe oder liefert die Ausgabe des gesamten Netzwerks.

Wie wir oben schon bewiesen haben, ist für eine bitonische Eingabe eine der beiden Ausgaben rein. Damit gibt es in jeder Stufe i des Netzwerks nur eine nicht reine Folge. Ferner sind alle reinen Folgen schon richtig sortiert. Da in der letzten Stufe nur noch Folgen der Länge 1 betrachtet werden, sind diese Folgen auch rein. □

Fakt 2 `bitonic-sorter`$[n]$ *hat Tiefe* $\log n$ *und Größe* $O(n \log n)$.

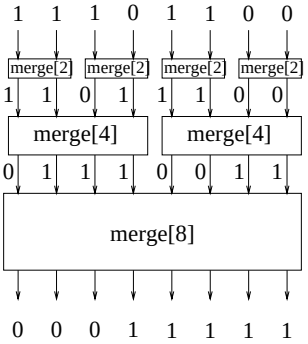


Abbildung 5.5: Ein Sortierer für Folgen der Länge 8.

Merging

Verschmelzen sortierter Teilfolgen ($merge$)
Eingabe: zwei sortierte Folgen X, Y gleicher Länge
Ausgabe: die sortierte Permutation von XY.

Für $X = x_1, \ldots, x_n$ sei $\text{rev}(X) = x_n, \ldots, x_1$ die umgedrehte (reverse) Teilfolge.

Fakt 3 *Sind die Folgen X und Y sortiert, so ist $X\text{rev}(Y)$ bitonisch.*

Damit steht uns mit `bitonic-sorter` schon ein effizienter `merge`-Schaltkreis zur Verfügung:

$$\texttt{merge}[n](X, Y) := \texttt{bitonic-sorter}[n](X, \text{rev}(Y))$$

Merging-Sortiernetzwerk

Das folgende Sortiernetzwerk löst das Sortierproblem, indem es baumförmig kleine sortierte Teilfolgen zu größeren vereint (siehe Abbildung 5.5)

Theorem 29 *Es gibt ein Sortiernetzwerk der Größe $O(n \log^2 n)$ und der Tiefe $\log^2 n$.*

Beweis: Folgendes Netzwerk hat diese Eigenschaften

`merging-network`$[n]$
input: $x_1, \ldots, x_n$
$m \leftarrow 1$
for $i \leftarrow 1$ **to** $\log n$ **do**
  $m \leftarrow 2m$
  for $j \in \{1, \ldots, n/m\}$ **in parallel do**
    $(a_{2(j-1)m+1}, \ldots, a_{2jm}) \leftarrow \texttt{merge}[m]((a_{(j-1)m+1}, \ldots, a_{jm}),$
      $(a_{(j-1)m+m+1}, \ldots, a_{2jm}))$
  od
od
output: $a_1, \ldots, a_n$

□

Die Frage, ob man ein Sortiernetzwerk der Tiefe $O(\log n)$ angeben kann, haben Ajtas, Komlós und Szemerédi positiv beantwortet. Leider hat dieses eine Größe von $\approx 2000 \log n$, so daß dieses Ergebnis für die Praxis irrelevant ist.

Ausgabeposition	1	2	3	4
W'keit, für Ausgabe der Zahl 1	1	0	0	0
2	0	2/3	1/3	0
3	0	1/3	2/3	0
4	0	0	0	1

Tabelle 5.1: Wahrscheinlichkeitsverteilung der vier Ausgaben eines Butterfly-Vergleichsnetzwerks bei gleichwahrscheinlichen Eingangspermutationen

Ausgabeposition	1	2	3	4	5	6	7	8
W'keit, für Ausgabe der Zahl 1	1	0	0	0	0	0	0	0
2	0	4/7	2/7	0	1/7	0	0	0
3	0	2/7	3/7	0	2/7	0	0	0
4	Übungsaufgabe							
5	Übungsaufgabe							
6	0	0	0	2/7	0	3/7	4/7	0
7	0	0	0	1/7	0	2/7	2/7	0
8	0	0	0	0	0	0	0	1

Tabelle 5.2: Wahrscheinlichkeitsverteilung der acht Ausgaben eines Butterfly-Vergleichsnetzwerks bei gleichwahrscheinlichen Eingangspermutationen

5.2 Sortieren durch einen Butterfly-Vergleicher

Kann man aber im Durchschnitt schneller sortieren als $\mathcal{O}(\log^2 n)$ und $2000 \log n$. Dieser Frage gehen Leighton und Plaxton in [LP90] nach.

5.2.1 Wie gut sortiert ein Butterfly-Vergleichsnetzwerk

Arrangiert man Komparatoren durch ein Butterfly-Netzwerk, bestünde die Hoffnung, dass die Ausgabe im wesentlichen schon sortiert ist. Aber schon für Eingabe der Größe vier kann ein Gegenbeispiel angegeben werden.

Man kann noch nicht einmal erwarten, dass eine korrekte Sortierung mit Wahrscheinlichkeit $\frac{1}{2}$ stattfinden. Hierzu betrachten wir die Wahrscheinlichkeit, dass in einer uniform permutierten Eingabe von $\{1, \ldots, n\}$ die Zahl i an der Stelle i erscheint.

Dies resultiert in der folgenden Wahrscheinlichkeitsverteilung aus Tabelle 5.1 und 5.1.

Wendet man aber auf die Ausgabe eine Permutation aus, so kann man zeigen, dass die Folge dann im wesentlichen sortiert ist. D.h. dass die Ausgaben nicht allzu weit von ihrer korrekten Position entfernt sind.

Theorem 30 *[LP90] Für das Butterfly-Vergleichsnetzwerk gibt es eine Permutation π, so dass*

1. *Es existieren feste Ausgabeposition Y mit $|Y| \leq n^\gamma$, die überhaupt nicht korrekt sortiert werden.*
2. *Für alle Ausgaben $i \in [n] \setminus Y$ wird an der Ausgabe i eine Zahl $j \in [i - n^\gamma, i + n^\gamma]$ mit Wahrscheinlichkeit $1 - \mathcal{O}(2^{-n^c})$ ausgegeben.*

Hierbei ist $\gamma = 0,822$.

Dem Beweis dieses Theorems werden wir uns den Rest dieses Abschnitts widmen.

Wir betrachten nun Eingaben aus k Nullen und $n-k$ Einser für zufällige Permutationen aus $\binom{n}{k}$. Hierzu sei

$$f_i(k) := \mathrm{P}[i\text{-te Ausgabe} = 0] \; .$$

Für diese Funktion gelten folgende Eigenschaften.

Fakt 4 *1.* $\forall k \in [n] \quad f_i(k) \leq f_i(k+1)$.

2. Der Sortierschaltkreis arbeitet korrekt, genau dann wenn

$$f_i(k) = \begin{cases} 1\,; & k > i \\ 0\,; & \textit{sonst?} \end{cases}$$

Der zweite Teil beschreibt eine Verallgemeinerung des 0/1-Prinzips. Aus diesen Aussagen kann man nun folgendes Lemma schlussfolgern.

Lemma 19 *Falls $f_i(u) \leq \epsilon$ und $f_i(v) \geq 1 - \epsilon'$ gilt. Dann kann bei einer zufälligen Permutation von $[n]$ als Eingabe die Ausgabe k mit Wahrscheinlichkeit $1 - \epsilon - \epsilon'$ innerhalb einer Position im Intervall $\{u, u+1, \ldots, v-1\}$ ausgegeben.*

Wir betrachten nun eine neue Eingabewahrscheinlichkeitsverteilung. Statt Permutationen, betrachten wir jedes Eingabebit gleichwahrscheinlich mit Wahrscheinlichkeit p auf 0 gesetzt und mit Wahrscheinlichkeit $1 - p$ auf 1 gesetzt. Ferner seien diese Eingabe-Zufallsereignisse voneinander unabhängig.

Für diese Eingabeverteilung sei

$$g_i(p) := \mathrm{P}[i\text{-te Ausgabe} = 0]\ .$$

Für diese Funktion gelten folgende Eigenschaften.

Fakt 5 *1.* $g_i(0) = 0$

2. $g_i(1) = 1$

3. $g_i'(p) := \frac{dg_i(p)}{dp} > 0$ *für* $p \in (0,1)$

Zwischen den beiden Wahrscheinlichkeitsverteilungen gilt folgende Beziehung:

$$g_i(p) = \sum_{0 \leq k \leq n} \binom{n}{k} p^k (1-p)^{n-k} f_i(k)\ .$$

Damit gilt folgendes Lemma.

Lemma 20 *Falls $g_i(u) \leq 2^{-n^\epsilon}$ und $g_i v \geq 1 - 2^{-n^{\epsilon'}}$ dann gilt für zufällige Eingabepermutationen über $[n]$, dass die i-te Ausgabe mit Wahrscheinlichkeit $\mathcal{O}(2^{-n^{c'}}$.*

Beweis: Sei $B(n,p,k) = \binom{n}{k} p^k (1-p)^{n-k}$. Durch Anwenden von Stirlings Formel kann man zeigen

$$B(n, k/n, k) = \Omega(\frac{1}{\sqrt{n}}\ .$$

Da nun $g_i(k/n) \geq B(n, k/n, k \cdot f_i(k))$ folgt $f_i(k) = \mathcal{O}(\sqrt{n} g_i(k/n))$. Damit ist

$$f_i(\lfloor u \cdot n \rfloor) = \mathcal{O}(\sqrt{n} g_i(\lfloor u \cdot n \rfloor / n)) = \mathcal{O}(\sqrt{n} g_i(u)) = \mathcal{O}(\sqrt{n} 2^{-n^\epsilon}) = \mathcal{O}(2^{-n^\epsilon + \frac{1}{2}\log n})\ .$$

Eine analoge Abschätzung gilt für $f_i(\lceil v/n \rceil)$. □

Wir definieren nun $a_{\mathrm{BIN}(i-1,k)} := g_i(k)$, wobei $\mathrm{bin}(i,k)$ die k-stellige Binärdastellung von i beschreibt. Mit Hilfe dieser Notation erhalten wir nun die folgenden Rekursionsgleichungen zur Beschreibung von $g_i(p)$ für $\alpha \in \{0,1\}^*$

$$\begin{aligned} a_{\alpha\,0}(p) &= 2a_\alpha(p) - a_\alpha(p)^2 \\ a_{\alpha\,1}(p) &= a_\alpha(p)^2 \end{aligned}$$

Wir betrachten nun die Umkehrfunktion $b_\alpha(z) := p$, so dass $a_\alpha(p) = z$ gilt. Für den leeren String Φ gilt:

$$b_\Phi(z) = z \ ,$$

weil $a_\Phi(z) = z$.

Für die Funktion $b_\alpha(z)$ kann nun ebenfalls eine Rekursionsgleichung angeben indem man die Umkehrfunktionen von $x \mapsto 2x - x^2$ und $x \mapsto x^2$ entsprechend einsetzt. Es gilt

$$\begin{aligned} b_{0\,\alpha}(p) &= 1 - \sqrt{1 - b_\alpha(z)} \\ b_{1\,\alpha}(p) &= \sqrt{b_{\alpha(z)}} \end{aligned}$$

Von besonderen Interesse sind die Werte

$$\begin{aligned} u_\alpha &= b_\alpha(2^{-n^\delta}) \\ p_\alpha &= b_\alpha(1/2) \\ v_\alpha &= b_\alpha(1 - 2^{-n^\delta}) \end{aligned}$$

Dies rührt daher, da die Ausgabe α zwischen den Positionen $\lfloor u_\alpha n \rfloor$ mit Wahrscheinlichkeit $1 - 2^{-n^\delta + 1}$ auftritt.

Wir werden zeigen, dass gilt $v_\alpha - u_\alpha = \mathcal{O}(n^{\delta - 1})$. Hierfür betrachten wir die "Distanz" zwischen u_α und v_α und deren Abnahme bei Zunahme von α.

Am Anfang sind p und q relativ weit von einander entferent.

Sei $\Delta(p, q) = \log \frac{q(1-p)}{(1-q)p}$ ein Abstandsmaß.

Für $\alpha = \Phi$ gilt:

$$\Delta(p, q) = \Delta(2^{-n^\delta}, 1 - 2^{-n^\delta}) = \log \frac{(1 - 2^{-n^\delta})^2}{2^{-n^{2\delta}}} = 2n^\delta + o(1) \ .$$

Es gilt:

$$y - x \leq \Delta(x, y) \ .$$

Zu zeigen ist $h_\alpha(2^{-n^\delta}, 1 - 2^{-n^\delta}) \leq n^{\gamma - 1 - \delta}$, wobei

$$h_\alpha(p, q) = \frac{\Delta(b_\alpha(p), b_\alpha(p))}{\Delta(p, q)} \ .$$

Als ersten Schritt beobachten wir

$$\begin{aligned} h_\Phi(p, q) &= 1 \\ h_{0\alpha}(p, q) &= h_0(b_\alpha(p), b_\alpha(q)) \cdot h_\alpha(p, q) \\ h_{1\alpha}(p, q) &= h_1(b_\alpha(p), b_\alpha(q)) \cdot h_\alpha(p, q) \end{aligned}$$

...

An dieser Stellen ist das Skript noch unvollständig.

...

Damit ist das Theorem bewiesen. Wie man damit einen $\mathcal{O}(\log n)$ tiefen Sortierschaltkreis konstruiert, sei dem Leser als Übung überlassen.

Kapitel 6

Zwei $\mathcal{NP}$-vollständige Probleme schnell gelöst

6.1 Färbung in konstanter Zeit

Das Problem der k-Färbung ist eines der ältesten graphthereotischen Probleme. Diese Aufgabe finge wohl mit der Frage an, ob eine Landkarte mit Staatsgebieten so eingefärbt werden kann, dass keine zwei benachbarte Länder die selbe Farbe besitzen. Wegen der eingeschränkten Drucktechnologie war die Reduzierung der Anzahl der verschiedenen Farben elementar. Waren die Staatsgebilde zusammenhängend (was in der Praxis sehr oft verletzt wird), so war den Kartographen kein Beispiel bekannt, in dem man mehr als vier Farben benötigt. Erst viel später wurde bewiesen in [RSST97], dass dies im Allgemeinen der Fall ist.

Formal ist das Färbungsproblem beschrieben durch eine Instanz eines ungerichteten Graphen, und einer Schranke für die Anzahl der verfügbaren Farben. Das Färbungsproblem war unter den ersten Problemen, die als $\mathcal{NP}$-vollständig erkannt worden sind. Andererseits ist es aber sehr einfach dieses Problem für einen zufälligen Graphen zu lösen, wie von Wilf in seinem Buch [Wil87] im letzten Abschnitt beschrieben wird (entsprechender Artikel [Wil84]).

6.1.1 Das chromatische Polynom

Zum Aufwärmen werden wir uns mit der Anzahl der Möglichkeiten einen gegebenen Graphen mit k Farben korrekt einzufärben befassen. Wir definieren hierfür die Funktion $\#\mathrm{C} : \mathcal{G} \times \mathbb{N} \to \mathbb{N}$ als

$$\#\mathrm{C}_G(k) := \text{Anzahl gültiger Färbungen von } G \text{ mit } k \text{ Farben} .$$

Das einzige was die Anzahl der Färbungsmöglichkeiten behindert sind Kanten. Daher gilt für die kantenentleerten Graphen $P_n := ([n], \emptyset)$:

$$\#\mathrm{C}_{P_n}(k) := k^n .$$

Sei nun $e = \{v, w\}$ eine Kante des Graphen $G = (V, E)$. Dann sei

$$G - \{e\} := (V, E \setminus \{e\}) .$$

Es gilt das folgende Lemma.

Lemma 21 *Die Anzahl der Färbungen von $G - \{v, w\}$ mit unterscheidlichen Färbungen von v und w ist gleich der Anzahl der Färbungen von G*

Dieses Lemma folgt sofort aus der Definition einer gültigen Färbung, die verlangt, dass zwei Knoten einer Kante unterschiedlich gefärbt sind.

Lemma 22 *Seien v und w Knoten, so dass $e = \{v, w\} \in E(G)$. Die Anzahl der Färbungen in $G - \{e\}$ in denen v und w die gleichen Färbungen haben ist gleich der Anzahl aller Färbungen von $G \setminus \{e\}$, wobei*

$$G \setminus \{v, w\} := (V \setminus \{w\}, (E \setminus \{\{x, w\} \mid x \in V\}) \cup \{\{x, v\} \mid v \in V \setminus \{w\} \wedge \{x, w\} \in E\} .$$

Beweis: Die Transformationen $G \setminus \{e\}$ kann man beschreiben, als Zusammenziehen der Kante e zu einem neuen Knoten, der alle Kanten der zusammengefassten Knoten erbt (und natürlich auf die Kante e verzichtet).

Durch diese Umformung bleibt die Färbungsbedingung jeweils erhalten und für jede Kante kann man sich leicht verdeutlichen, dass die Anzahl der Färbungsmöglichkeiten durch diese Transformation nicht eingeschränkt wird. □

Fasst man diese beiden Lemmata zusammen, erhält man die folgende Beschreibung der Anzahl der Möglichkeiten einen Graphen $G - \{e\}$ gültig zu färben.

Lemma 23 *Für $e \in E(G)$ gilt:*

$$\#\mathrm{C}(G - \{e\}, k) = \#\mathrm{C}(G \setminus \{e\}, k) + \#\mathrm{C}(G, k) \; .$$

Beweis: Wir haben in den beiden vorgegangen Lemmas jeweils die Anzahl der Färbungsmöglichkeiten von $G - \{e\}$ mit gleicher und unterschiedlicher Färbung der Endknoten von e beschrieben durch $\#\mathrm{C}(G, k)$ und $\#\mathrm{C}(G \setminus \{e\}, k)$. □

Dieses Lemma beschreibt schließlich auch eine konstruktive Methode zur Bestimmung der Anzahl der Färbungen von G. Hierzu formen wir die Gleichung des Lemmas um zu

$$\#\mathrm{C}(G, k) = \#\mathrm{C}(G - \{e\}, k) - \#\mathrm{C}(G \setminus \{e\}, k) \; .$$

Diese Regel kann man nun rekursiv anwenden ohne befürchten zu müssen, in eine Endlosregel zu kommen, da sich in jedem Schritt die Anzahl der Kanten um mindestens eine (nämlich e) reduziert. Die Knotenanzahl bleibt hierbei in $G - \{e\}$ gleich und wird in $G \setminus \{e\}$ um eins reduziert. Wir erinnern uns, dass kantenleere Graphen mit n Knoten als Färbungszahl ein Polynom vom Grad n haben und erhalten daher die Gewissheit, dass das folgende Theorem gilt:

Theorem 31 *Die Anzahl der Möglichkeiten einen Graphen mit n Knoten mit k Farben korrekt einzufärben wird durch ein Polynom über k mit Grad von höchstens n beschrieben.*

Beweis: Als Beweis kann man eine vollständige Induktion über die Anzahl der Kanten angeben. □

6.1.2 Drei nützliche Lemmas

Bevor wir nun zeigen, dass wir 3-Färbungen in konstanter erwarteter Zeit berechnen können, müssen wir den folgenden ganz nützlichen Satz (elegant) beweisen.

Lemma 24 *Für alle $s_1, \ldots, s_k \geq 0$ mit $\sum_{i \in [k]} s_i = L$ gilt:*

$$\sum_{i \in [k]} (s_i)^2 \geq \frac{L^2}{k} \; .$$

Beweis: Es gilt

$$\begin{aligned} 0 \quad \leq \quad \sum_{i=1}^{k} \left(s_i - \frac{L}{k} \right)^2 \quad &= \quad \sum_{i=1}^{k} (s_i)^2 - 2L\frac{s_i}{K} + \frac{L^2}{k^2} \\ &= \quad \sum_{i=1}^{k} (s_i)^2 - 2L\frac{L}{K} + \frac{L^2}{k^2} \\ &= \quad \left(\sum_{i=1}^{k} (s_i)^2 \right) - \frac{L^2}{k} \end{aligned}$$

Damit ist das Lemma bewiesen. □

Wir stellen nun die umgedrehte Frage zur Anzahl Färbungen eines Graphens. Angenommen wir hätten eine Färbung vorgegeben, wie viele Möglichkeiten gibt es für Graphen mit eben dieser Färbung.

Lemma 25 *Sei $C : [n] \to [k]$ eine vorgegebene Färbung von k Farben auf n Knoten. Dann ist die Anzahl der Graphen, für die eine solche Kantenfärbung gültig ist, höchstens $2^{\frac{n^2(1-\frac{1}{k})}{2}}$*

Beweis: Bezeichne für die Färbung C die Zahlen s_i die Anzahl der Knoten, welche die Farbe i erhalten. Dann gilt natürlich

$$\sum_{i=1}^{k} s_i = n \ .$$

Mögliche Kanten können natürlich nur zwischen verschiedenfarbigen Knoten entstehen. Damit ist die Anzahl möglicher Kanten beschrieben durch

$$\begin{aligned}
\sum_{1 \le i < j \le k} s_i s_j &= \sum_{i \neq j} s_i s_j \\
&= \frac{1}{2}\left(\frac{i,j \in [k]}{s \quad _i} s_j - \sum_{i \in [k]} (s_i)^2 \right) \\
&= \frac{1}{2}\left(\sum_{i \in [k]} (s_i)^2 \right)^2 - \frac{1}{2} \sum_{i \in [k]} (s_i)^2 \\
&\le \frac{n^2}{2} - \frac{1}{2}\left(\frac{n^2}{k}\right) = \frac{1}{2} n^2 \left(1 - \frac{1}{k}\right) \ .
\end{aligned}$$

Da jede dieser Kanten vorhanden oder nicht vorhanden ist, ergibt sich also obere Schranke für die Gesamtanzahl aller Möglichkeiten $2^{\frac{1}{2}n^2\left(1-\frac{1}{k}\right)}$. □

Hieraus folgt sofort die folgende obere Schranke für die Anzahl von Färbungen eines Graphen:

Lemma 26 *Die Anzahl gültiger Färbungen eines Graphen mit n Knoten durch k Farben ist höchstens: $n^k 2^{n^2(1-1/k)/2}$*

Beweis: Wir zählen zuerst alle mögliche Färbungen $C_1, \ldots, C_m$ von n Knoten mit k Farben auf. Dieses sind n^k viele. Dann wenden wir das eben bewiesene Lemma an und erhalten das gewünschte Resultat. □

6.1.3 Der Backtracking-Algorithmus

In Abbildung 6.1 ist ein einfacher Backtracking-Algorithmus dargestellt, der das Graphfärbungsproblem löst. Wenn wir den Berechnungsbaum dieses Algorithmus betrachten, erkennen wir, dass in Tiefe i immer der Knoten i gefärbt wird. Als innere Knoten des Baumes ergeben sich damit in jeder Tiefe höchstens so viele Möglichkeiten, wie der Teilgraph bestehend aus den ersten i Knoten (und induzierten Kanten) gefärbt werden kann.

Da der Baum höchstens k-mal mehr Knoten als innere Blätter haben kann, gilt also:

Lemma 27 *Der Algorithmus in Abbildung 6.1 erzeugt einen Berechnungsbaum der Größe $\mathcal{O}(\sum_{i=1}^{n} \#\mathrm{C}(G_i, k)$, wobei $G_m := ([m], E \cap \{\{i,j\} \mid i,j \in [m]\}$.*

Damit ergibt sich die durchschnittliche Anzahl $A(n,k)$ von Knoten im Backtracking-Algorithmus

```
Proc Coloring (G, k)
    begin
        i ← 1
        j ← 1
        while i ∈ [n] do
            C_i ← j (Färbe Knoten i mit Farbe j)
            if Färbung C_1, ..., C_i gültig then
                i ← i + 1
                j ← 1
            else
                if j < k then
                    j ← j + 1
                else
                    i ← i − 1
                    if i ≥ 1 then
                        j ← C_i + 1
                    fi
                fi
            fi
        od
        if i = n + 1 then
            return Graph G mit k Farben färbbar
        else
            return Graph G mit k Farben nicht färbbar
        fi
    end
```

Abbildung 6.1: Ein einfacher Backtracking-Algorithmus für das Färbungsproblem

durch:

$$\begin{aligned}
A(n,k) &= \frac{1}{\text{\# Graphen mit } n \text{ Knoten}} \cdot \sum_{G \in G_n} \text{\# Knoten im Suchbaum für } G \\
&= 2^{-\binom{n}{2}} \sum_{G \in \mathcal{G}_n} \sum_{d=0}^{n} \text{\# Knoten in Tiefe } d \\
&= 2^{-\binom{n}{2}} \sum_{G \in \mathcal{G}_n} \sum_{d=0}^{n} \#\mathrm{C}(G_d, k) \\
&= 2^{-\binom{n}{2}} \sum_{d=0}^{n} \sum_{G \in \mathcal{G}_n} \#\mathrm{C}(G_d, k) \\
&= 2^{-\binom{n}{2}} \sum_{d=0}^{n} 2^{\binom{n}{2}-\binom{d}{2}} \sum_{G \in \mathcal{G}_d} \#\mathrm{C}(G_d, k) \\
&= \sum_{d=0}^{n} 2^{-\binom{d}{2}} \sum_{G \in \mathcal{G}_d} \#\mathrm{C}(G, k)
\end{aligned}$$

Hierbei liegt die Beobachtung zugrunde, dass es $2^{\binom{n}{2}-\binom{d}{2}}$ Möglichkeiten gibt, einen bestimmten Graphen G mit d Knoten $1, \ldots, d$ in einen Graphen G mit Knoten $[n]$ wiederzufinden. Das kommt daher, dass $\binom{n}{2} - \binom{d}{2}$ Kanten noch frei wählbar sind in $G \setminus G_d$.

Durch Anwendung des Lemmas folgt nun:

$$\begin{aligned}
A(n,k) &\leq \sum_{d=0}^{n} 2^{-\binom{d}{2}} \cdot k^d \cdot 2^{\frac{1}{2}d^2(1-1/k)} \\
&\leq \sum_{d=0}^{n} 2^{-\frac{d(d-1)}{2} + \frac{d^2}{2} - \frac{d^2}{2k} + d\log k} \\
&\leq \sum_{d=0}^{n} 2^{-\frac{d^2}{2k} + \frac{d}{2} d \log k} \quad \leq \quad 2^{\mathcal{O}(k \log^2 k)}
\end{aligned}$$

Damit konviergiert diese Folge unabhängig von n. Wir können also erwarten, dass der Suchbaum höchstens k innere Knoten hat. Für diese Größe kann man aber den Färbungstest innerhalb eines Teilgraphen mit k Knoten in Zeit k^2 lösen. Der gesamte Algorithmus ließe sich also in konstanter Zeit lösen. Daneben muss aber auch bemerkt werden, dass die Laufzeit in der Anzahl der verfügbaren Farben exponentiell zunimmt.

Der Wermutstropfen an diesem Ergebnis ist, dass der Algorithmus nur dann effizient ist, wenn keine Färbung gefunden wird. Er ist also nur deswegen schnell, weil ziemlich schnell verifiziert werden kann, dass der Graph nicht färbbar ist. Damit ist der Algorithmus für die wirklich interessanten Fälle nicht zu gebrauchen.

6.2 Das Rucksackproblem in erwartet polynomieller Zeit

Das Rucksackproblem ist eines der wichtigsten Optimierungsprobleme. Formal ist es gegeben durch:

- Gegeben n Elemente $(w_1, p_1), \ldots, (w_n, p_n) \in \mathbb{R}^2$ und eine Schranke $c \in \mathbb{R}$.
- Gesucht ist eine Teilmenge $S \subseteq [n]$, wobei
  1. $\sum_{i \in S} w_u \leq c$,
  2. $\sum_{i \in S} p_i$ ist maximal.

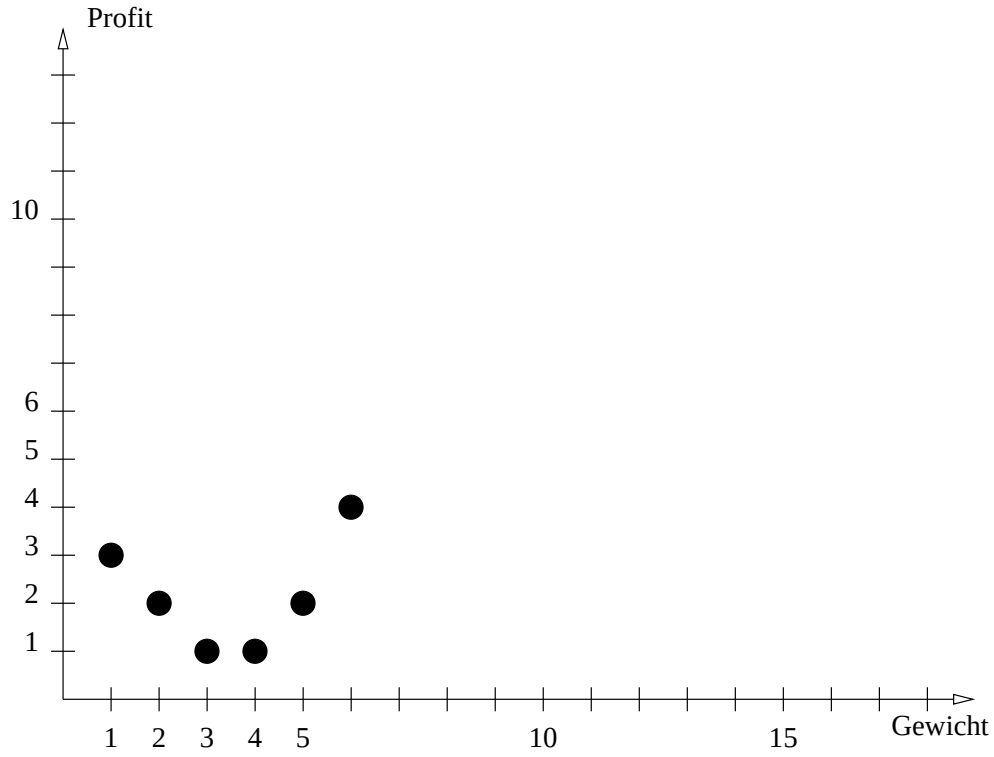


Abbildung 6.2: Gewicht/Profit-Diagramm — Ausgangslage

Die Parameter w_i werden Gewichte genannt und p_i Profite. Wir stellen uns also einen Schatzsucher vor, der n verschiedene Stücke findet und nur ein Gesamtgewicht von c nach Hause tragen kann. Der Wert eines Stücks ist nun p_i und dieser Profit soll maximiert werden.

Wir diskutieren kurz eine einfache Strategie für dieses Problem.

Gierige Profitmaximierung Wenn der Schatzsucher nun das teuerste Stück nimmt, das er gerade noch tragen kann und dann das nächst teuerste und so weiter. Dann verschenkt er viel, wie man an den folgenden Beispiel sieht:

i	1	2	3	4	5	6
w_i	1	2	3	4	5	6
p_i	3	2	1	1	2	4

Wenn nun $c = 6$, würde der gierige Schatzsuche das schwere Stück mit Gewicht 6 und Wert 4 nehmen. Er würde aber mit den Elementen $\{1, 2, 3\}$ das selbe Gewicht, aber Wert 6 erreichen.

Bis jetzt ist noch kein polynomial-zeitbeschränkter Algorithmus bekannt, der dieses Problem effizient löst. Was nicht verwundert, wenn man weiss, dass das Rucksackproblem $\mathcal{NP}$-vollständig ist.

Wir tragen nun die Elemente unseres Beispiels in Abbildung6.2 in ein Gewichts/Profit-Diagramm. Der senkrechte Strich deutet an, wie groß der Rucksack ist. Kombiniert man zwei Elemente, wie z.B. 1 und 2 so erhält man einen neuen Punkt. In Abbildung6.3 sind alle Kombinationen dargestellt. Die interessanten Kombinationen sind die, deren Gewicht kleiner ist als alle anderen, wobei der Profit größer ist als andere. Diese Menge entsprechen der dargestellte Treppenkurve. Formal führen wir hierfür den Begriff der Dominanz ein.

Definition 20 (Weingartner,Neuss) *Für eine Instanz I dominiert die Menge $S_1 \subseteq [n]$ die Menge $S_2 \subseteq [n]$ genau dann wenn*

$$\sum_{i \in S_1} w_i \leq \sum_{i \in S_2} w_i \quad \wedge \quad \sum_{i \in S_1} p_i \geq \sum_{i \in S_2} p_i \, .$$

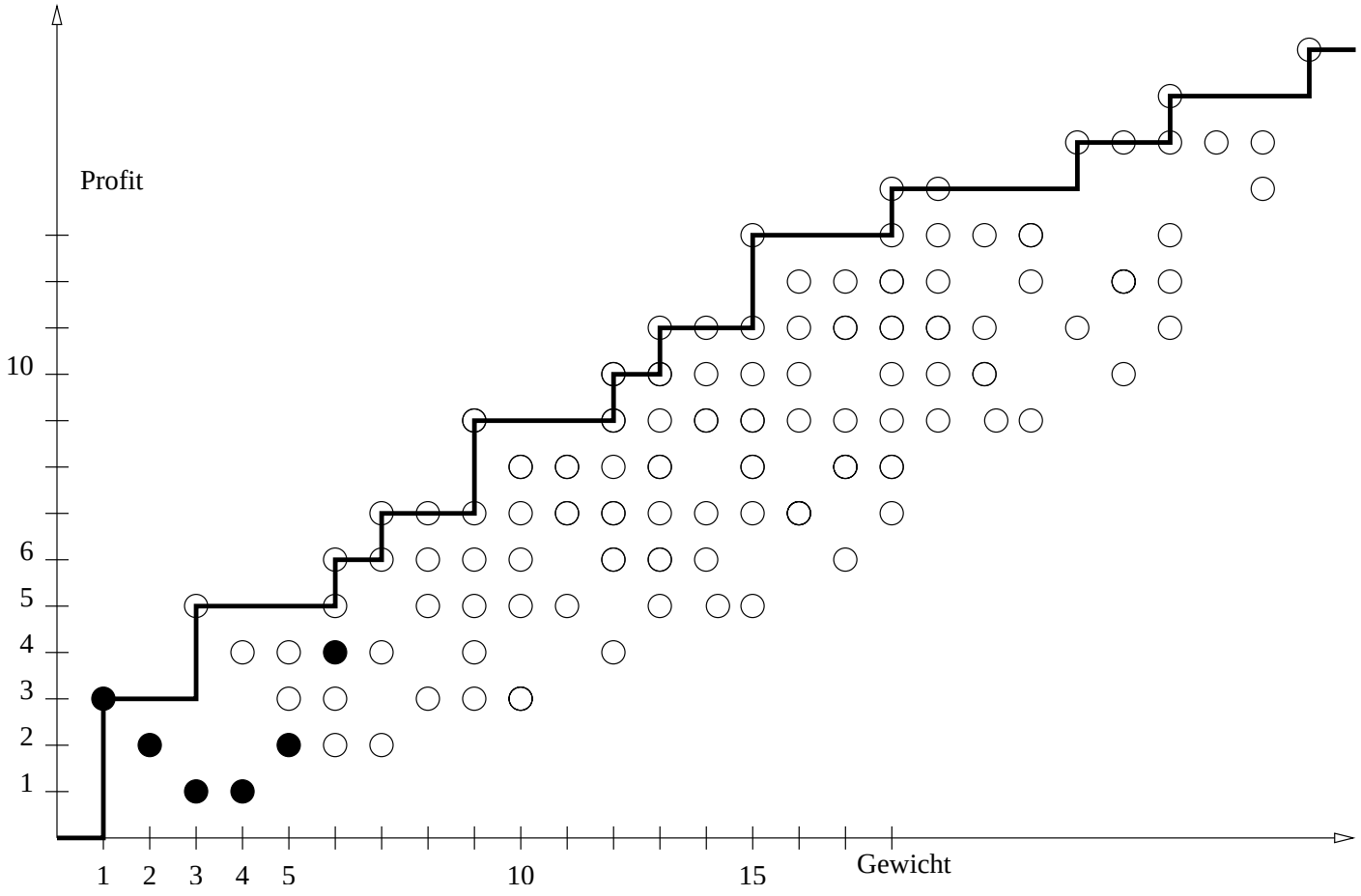


Abbildung 6.3: Gewicht/Profit-Diagramm mit allen Kombinationsmöglichkeit und Pareto-optimaler Linie

Für das Gesamtgewicht der Elemente aus $S \subseteq [n]$ schreiben wir auch $w(S) := \sum_{i \in S} w_i$, sowie $p(S) := \sum_{i \in S} p_i$ für den Gesamtprofit.

Die Klasse aller Mengen, die alle anderen Mengen dominieren, ist besonders interessant, da diese unsere Lösungsmenge beinhaltet. Diese Mengen nennt man Pareto-optimal. Wir notieren sie mit $D_I(n)$ für eine Instanz I. Man kann sie schrittweise konstruieren, indem man sukzessive die Elementmenge vergrößert.

Definition 21 *Für eine Instanz I seien $D_I(i) := \{S_1, \ldots, S_m\}$ mit $S_j \subseteq [i]$ die Menge aller Mengen, die alle anderen Mengen $S_{\subseteq}[i]$ dominieren. Also*

$$D_I(i) := \{S \subseteq [i] \mid \nexists S' \subseteq [n] \ : \ S' \textit{ dominiert } S\} \ .$$

Kennt man $D_I(i)$, so kann man die Menge $D_I(i+1)$ bestimmen, indem man das folgende Lemma anwendet.

Lemma 28

$$D_I(i+1) \ \subseteq \ D_I(i) \cup \{S \cup \{i+1\} \mid S \in D_I(i)\} \ .$$

Beweis: Wir beweisen diese Lemma durch einen Widerspruchsbeweis in dem wir eine Menge S voraussetzen, die in $D_I(i+1)$ vorhanden ist aber nicht in der Menge auf der rechten Seite der Inklusion. Hierfür gibt es zwei Fälle

1. Fall $\{i+1\} \notin S$.

   Dann gibt es keine Teilmenge von $[i+1]$, die S dominiert. Somit dominiert auch keine Teilmeng von $[i]$ die Menge S. Also ist $S \in D_I(i)$.

2. Fall: $\{i+1\} \in S$.

   Keine Teilmenge in $[i+1]$ dominert S. Damit dominiert auch keine Teilmenge von $[i] \cup \{i+1\}$ die Menge S. Sei $S' = S \setminus \{i+1\}$. Dann dominiert keine Teilmenge von $[i]$ die Menge S', wie man leicht aus der Definition der Dominanz folgern kann. Also ist $S' \in D_(i)$.

□

Die Mengen $D_I(i)$ kann man in geeigneter Darstellung effizient, d.h. in Linearzeit berechnen. Hierfür werden die Elemente S gemäß ihres Gewichts $w(S)$ sortiert in eine Liste gespeichert. Damit folgt sofort das folgende Lemma.

Lemma 29 *Aus $D_I(i)$ kann man $D_I(i+1)$ in Linearzeit berechnen.*

Beweis: Aus der Liste L zu $D_I(i)$ berechnen wir zuerst die Liste L', die der Menge $\{S \cup \{i+1\} \mid S \in D_I(i)\}$ entspricht. Nach dem vorangegangen Lemma wissen wir, dass $D_I(i+1)$ eine Teilmenge der Vereinigung dieser beiden Mengen ist. Nun führen wir eine Merge-Operation der Listen L und L' aus, wobei wir eine gemäß des Gewichts sortierte neue Liste $\hat{L}$ erhalten. In dieser Liste werden nun durch eine Durchlauf die nicht-pareto-optimalen Elemente entfernt, indem man die Elemente entfernt, die nicht der Treppenkurve entsprechen. □

Jetzt sind wir in der Lage, den Nemhauser-Ullman-Algorithmus zu beschreiben, siehe Abbildung6.4.

Bezüglich der Laufzeit des Nemhauser-Ullmann-Algorithmus können wir die folgenden Beobachtungen machen.

- Der Nemhauser-Ullmann-Algorithms hat im Worst-Case exponentielle Laufzeit.

  Zwar benötigt der einzige nicht-triviale Berechnungsschritt der Berechnung von $D_I(i+1)$ aus $D_I(i)$ nur lineare Laufzeit, jedoch bezieht sich dies Laufzeit auf die Größe von $D_I(i)$ und diese Menge kann sich im schlimmsten Fall in jeden Schritt verdoppeln, was zwangsläufig zu exponentieller Laufzeit führt.

- Als zweite Beobachtung erhalten wir für die Laufzeit $\text{time}_{NU}(I)$ des Nemhauser-Ullman-Algorithmus.

$$\text{time}_{NU}(I) \ = \ \mathcal{O}\left(\sum_{i=1}^{n} |D_I(i)|\right) \ = \ \mathcal{O}(n \cdot \max_{i \in [n]} |D_I(n)|) \ .$$

Proc Nemhauser-Ullman-Algorithmus
Geg.: Instanz $I = (p_i, w_i)_{i\in[n]}$ mit $p_i, w_i \in \mathbb{R}^+$.
begin
$D_I(1) \leftarrow \{(p_1, w_1)\}$
for $i \leftarrow 2$ **to** n **do**
Berechne $D_I(i+1)$ aus $D_I(i)$
od
return $D_I(n)$
end

Abbildung 6.4: Der Nemhauser-Ullman-Algorithmus zur Berechnung aller Pareto-opimalen Lösungen einer Instanz des Rucksackproblems

6.2.1 Average-Analyse

Wir bezeichnen mit $q(n) := \max_{i\in[n]} |D_I(n)|$. Wenn die Größe von $q(n)$ im Erwartungswert polynomiell beschränkt ist, dann ist der Nemhauser-Ullmann-Algorithmus ebenfalls im Erwartungswert polynomiell beschränkt. Damit genügt es folgendes Theorem zu beweisen:

Theorem 32 *Werden p_i, w_i für alle $i \in [n]$ gleichwahrscheinlich und unabhängig aus dem Intervall $[0, 1]$ gezogen, dann gilt*

$$\mathbf{E}[q(n)] \;=\; \mathbf{E}\left[\max_{i\in[n]} |D_I(n)|\right] \;\leq\; 16n^3 + 1\,.$$

Wir werden den Rest diesen Abschnitts den Beweis dieses Theorems widmen, da dies die folgende Laufzeitaussage zulässt.

Korollar 2 *Das Rucksackprobem läßt sich in erwartet polynomieller Zeit von $\mathcal{O}(n^4)$ berechnen.*

Sei nun $q := |D_I(n)|$ die Anzahl der dominierenden Mengen von I. Sei $m := 2^n$ und seien $S_1, \ldots, S_m$ die Menge aller Teilmengen von $[n]$ sortiert nach ihrem Gewicht.
Bezeichne nun $P_I := p(S_u) := \sum_{i\in S_u} p_i$ und $\Delta_u := \max_{v\in[u]} p_v - \max_{v\in[u-1]} p_v$.
Es gilt folgendes Lemma.

Lemma 30 *Für $u \geq 2$ gilt:*

$$\mathbf{E}[\Delta_u \mid \Delta_u > 0] \;\geq\; \frac{1}{32n^2}.$$

Beweis: Es gilt nach der Markov-Ungleichung:

$$\mathbf{E}[\Delta_u \mid \Delta_u > 0] \;\geq\; \mathrm{P}\left[\Delta_u \geq \frac{1}{16n^2}\right] \cdot \frac{1}{16n^2}\,.$$

Wir werden im folgenden Lemma zeigen, dass $\mathrm{P}\left[\Delta_u \geq \frac{1}{16n^2} \mid \Delta > 0\right] \geq \frac{1}{16n^2}$. Hieraus folgt dann der Beweis dieses Lemmas.
Nach Definition gilt

$$\mathrm{P}\left[\Delta_u \geq \frac{1}{16n^2} \mid \Delta > 0\right] \;=\; \mathrm{P}\left[\forall v \in [u-1] \;:\; p(S_u) \geq p(S_v) + \frac{1}{16n^2} \mid \forall v \in [u-1] \;:\; p(S_u) > p(S_v)\right]\,.$$

Wir halten u fest und bezeichnen nun mit $X_v := S_u \setminus S_v$ und $Y_v := S_v \setminus S_u$. Dann gilt also:

$$\mathrm{P}\left[\Delta_u \geq \frac{1}{16n^2} \mid \Delta > 0\right] \;=\; \mathrm{P}\left[\forall v \in [u-1] \;:\; p(X_u) \geq p(Y_v) + \frac{1}{16n^2} \mid \forall v \in [u-1] \;:\; p(X_v) > p(Y_v)\right]\,.$$

Wir sortieren nun die Elemente aus $\{1, \ldots, n\}$ so um, dass $S_u = \{1, \ldots, k\}$ und $[n]\setminus S_u = \{k+1, \ldots, n\}$. Damit gilt $X_v \subseteq [k]$ und $Y_v \subseteq \{k+1, \ldots, n\}$.
Sei nun $L_v = p(X_v)$. Wir zeigen zuerst das folgende Lemma.

Lemma 31

$$\mathrm{P}[L_v \geq \frac{1}{4n} \mid \forall v \in [u-1] \ : \ p(X_v) \geq p(Y_v)] \ \geq \ \frac{3}{4} \, .$$

Beweis: Es gilt:

$$\begin{aligned}
\mathrm{P}[L_v \leq \frac{1}{4n} \mid \forall v \in [u-1] \ : \ p(X_v) \geq p(Y_v)] \ &\leq \ \mathrm{P}[\exists j \in [k] : p_j \geq \frac{1}{4n} \mid \forall v \in [u-1] \ : \ p(X_v) \geq p(Y_v)] \\
&\leq \ \sum_{j \in [k]} \mathrm{P}[p_j \geq \frac{1}{4n} \mid \forall v \in [u-1] \ : \ p(X_v) \geq p(Y_v)] \\
&\leq \ \sum_{j \in [k]} \mathrm{P}[p_j \geq \frac{1}{4n}] \\
&= \ \frac{k}{4n} \ \leq \ \frac{1}{4} \, .
\end{aligned}$$

Die Abschätzung die von bedingten Wahrscheinlichkeiten zu unbeedingten überführt„ gilt, da die Bedingung $\forall v \in [u-1] : p(X_v) \geq p(Y_v)$ wegfällt, dann p_j sogar wahrscheinlicher wird, da $j \in [k]$ wie eben auch $X_v \subseteq [k]$. □

Wir kehren zum Beweis des vorherigen Lemmas zurück und beobachten, dass

$$\mathrm{P}[\forall v \in [u-1] \ : \ p(X_u) \geq p(Y_v) + \frac{1}{16n^2} \ \mid \forall v \in [u-1] \ : \ p(X_v) > p(Y_v)]$$

$$= \ \mathrm{P}[\forall v \in [u-1] \ : \ L_v \geq p(Y_v) + \frac{1}{16n^2} \ \mid \forall v \in [u-1] \ : \ p(Y_v) > L_v] \ .$$

Sei nun

$$A := \left\{ (p_{k+1} \times \ldots \times p_n) \in [0,1]^{n-k} \mid \forall v \in [u-1] \ : \ p(Y_v) > L_v - \frac{1}{16n^2} \right\}$$

eine Teilmenge des $n-k$-dimensionalen Raums. Ferner sei

$$B := \left\{ (p_{k+1} \times \ldots \times p_n) \in [0,1]^{n-k} \mid \forall v \in [u-1] \ : \ p(Y_v) > L_v \right\} \ .$$

Sei $\mathrm{Vol}(A)$ der Rauminhalt einer Menge A. Dann gilt gemäß dieser Notation

$$\mathrm{P}[\Delta_u \geq \frac{1}{16n^2} \mid \Delta_u > 0] \ = \ \frac{\mathrm{Vol}(A \cap B)}{\mathrm{Vol}(B)} = \frac{\mathrm{Vol}(A)}{\mathrm{Vol}(B)} \, ,$$

da $A \subseteq B$.

Sei nun

$$B_\epsilon := \{ \left\{ (p_{k+1} \times \ldots \times p_n) \in [0,1-\epsilon]^{n-k} \mid \forall v \in [u-1] \ : \ p(Y_v) > (1-\epsilon) L_v \right\} \ .$$

Dann gilt $\mathrm{Vol}(B_\epsilon) = (1-\epsilon)^{n-k} \cdot \mathrm{Vol}(B)$. Wie muss nun ϵ gewählt werden, so dass $B_\epsilon \subseteq A$ mit Wahrscheinlichkeit $\frac{3}{4}$? Da $L_v \geq \frac{1}{4n}$ mit Fehlerwahrscheinlichkeit $\frac{1}{4}$ erhalten wir für $\epsilon = \frac{1}{4n}$, dass $L_v(1-\epsilon) = L_v(1-\frac{1}{4n}) \leq L_v - \frac{1}{16n^2}$ und damit ist $B_\epsilon \subseteq A$.

Nun gilt für die Volumina dieser Mengen:

$$\mathrm{Vol}(A) \geq \mathrm{Vol}(B_\epsilon) = (1-\epsilon)^{n-k} \cdot \mathrm{Vol}(B) \geq (1-\epsilon(n-k)) \cdot \mathrm{Vol}(B) \geq \frac{3}{4} \mathrm{Vol}(B) \, .$$

Damit gilt aber

$$\mathrm{P}[\Delta_u \geq \frac{1}{16n^3} \mid \Delta_u > 0] \ \geq \ \frac{3}{4} - \frac{1}{4} = \frac{1}{2} \, .$$

Damit folgt das Lemma und auch das zuvor stehende Theorem. □

Literaturverzeichnis

[AKS83] AJTAI, M., J. KOMLÓS und E. SZEMERÉDI: *An $O(n \log n)$ sorting network*. In: ACM (Herausgeber): *Proceedings of the fifteenth annual ACM Symposium on Theory of Computing, Boston, Massachusetts, April 25–27, 1983*, Seiten 1–9, New York, NY, USA, 1983. ACM Press.

[BDCGL92] BEN-DAVID, SHAI, BENNY CHOR, ODED GOLDREICH und MICHAEL LUBY: *On the Theory of Average Case Complexity*. Journal of Computer and System Sciences, 44(2):193–219, April 1992.

[BG95] BLASS, ANDREAS und YURI GUREVICH: *Matrix Transformation Is Complete for the Average Case*. SIAM Journal on Computing, 24(1):3–29, 1995.

[BV03] BEIER, RENE und BERTHOLD VÖCKING: *Random knapsack in expected polynomial time*. In: *Proceedings of the thirty-fifth ACM symposium on Theory of computing*, Seiten 232–241. ACM Press, 2003.

[BW85] BENDER, EDWARD A. und HERBERT S. WILF: *A Theoretical Analysis of Backtracking in the Graph Coloring Problem*. Journal of Algorithms, 6(2):275–282, Juni 1985.

[CM97] COOK, STEPHEN A. und DAVID G MITCHELL: *Finding Hard Instances of the Satisfiability Problem: A Survey*. In: DU, GU und PARDALOS (Herausgeber): *Satisfiability Problem: Theory and Applications*, Band 35, Seiten 1–17. American Mathematical Society, 1997.

[CR92] CHVATAL, V. und B. REED: *Mick gets some (the odds are on his side) (satisfiability)*. In: IEEE (Herausgeber): *33rd Annual Symposium on Foundations of Computer Science: October 24–27, 1992, Pittsburgh, Pennsylvania: proceedings [papers]*, Seiten 620–627, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1992. IEEE Computer Society Press.

[DB97] DUBOIS, O. und Y. BOUFKHAD: *A General Upper Bound for the Satisfiability Threshold of Random r-SAT Formulae*. Journal of Algorithms, 24(2):395–420, August 1997.

[DGY90] DAVID, ILANA, RAN GINOSAR und MICHAEL YOELI: *An Efficient Implementation of Boolean Functions and Finite State Machines as Self-Timed Circuits*. In: *International Conference on Computer Systems and Software Engineering (COMP-EURO)*, Seiten 148–155. IEEE Computer Society Press, Mai 1990.

[DLL62] DAVIS, MARTIN, GEORGE LOGEMANN und DONALD LOVELAND: *A machine program for theorem-proving*. Communications of the ACM, 5(7):394–397, Juli 1962.

[DP60] DAVIS, MARTIN und HILARY PUTNAM: *A Computing Procedure for Quantification Theory*. JACM, 7(3):201–215, 1960.

[For79] FORTUNE, S.: *A note on sparse complete sets*. SIAM Journal on Computing, 8(3):431–433, 1979.

[FS96] FRIEZE und SUEN: *Analysis of Two Simple Heuristics on a Random Instance of k-SAT*. ALGORITHMS: Journal of Algorithms, 20, 1996.

[GJ78] GAREY, MICHAEL R. und DAVID S. JOHNSON: *Computers and Intractability / A Guide to the Theory of NP-Completeness*. W.H. Freeman & Company, San Francisco, 1978.

[Goe92] GOERDT, A.: *A Treshold for Unsatisfiability*. Lecture Notes in Computer Science, 629:264–??, 1992.

[Gol79] GOLDBERG, ALLEN: *Average case complexity of the satisfiability problem*. In: *Fourth Workshop on Automated Deduction*, Austin, Texas, Februar 1-3 1979.

[Gol97] GOLDREICH: *Notes on Levin's Theory of Average-Case Complexity*. In: *ECCCTR: Electronic Colloquium on Computational Complexity, technical reports*, 1997.

[Gur91] GUREVICH, YURI: *Average Case Completeness*. Journal of Computer and System Sciences, 42(3):346–398, Juni 1991.

[JRS94] JAKOBY, ANDREAS, RÜDIGER REISCHUK und CHRISTIAN SCHINDELHAUER: *Circuit complexity: from the worst case to the average case*. In: ACM (Herausgeber): *Proceedings of the twenty-sixth annual ACM Symposium on the Theory of Computing: Montreal, Quebec, Canada, May 23–25, 1994*, Seiten 58–67, New York, NY, USA, 1994. ACM Press.

[JRS95] JAKOBY, ANDREAS, RÜDIGER REISCHUK und CHRISTIAN SCHINDELHAUER: *Malign Distributions for Average Case Circuit Complexity*. In: *12th Annual Symposium on Theoretical Aspects of Computer Science*, Band 900 der Reihe *lncs*, Seiten 628–639, Munich, Germany, 2–4 März 1995. Springer.

[JRSW94] JAKOBY, A., R. REISCHUK, C. SCHINDELHAUER und S. WEIS: *The Average Case Complexity of the Parallel Prefix Problem*. Lecture Notes in Computer Science, 820:593–605, 1994.

[JS96] JAKOBY, ANDREAS und CHRISTIAN SCHINDELHAUER: *On the Complexity of Worst Case and Expected Time in a Circuit*. In: *13th Annual Symposium on Theoretical Aspects of Computer Science*, Band 1046 der Reihe *lncs*, Seiten 295–306, Grenoble, France, 22–24 Februar 1996. Springer.

[Kap90] KAPIDAKIS, SARANTOS: *Average-case analysis of graph-searching algorithms*. Doktorarbeit, Princeton University, 1990.

[Kar72] KARP, R. M.: *Reducibility Among Combinatorial Problems*, Seiten 85–103. Plenum Press, NY, 1972.

[KKK96] KIROUSIS, LEFTERIS M., EVANGELOS KRANAKIS und DANNY KRIZANC: *Approximating the Unsatisfiability Threshold of Random Formulas (Extended Abstract)*. In: DÍAZ, JOSEP und MARIA SERNA (Herausgeber): *Algorithms—ESA '96, Fourth Annual European Symposium*, Band 1136 der Reihe *Lecture Notes in Computer Science*, Seiten 27–38, Barcelona, Spain, 25–27 September 1996. Springer.

[KMPS94] KAMATH, A., R. MOTWANI, K. PALEM und P. SPIRAKIS: *Tail bounds for occupancy and the satisfiability threshold conjecture*. In: GOLDWASSER, SHAFI (Herausgeber): *Proceedings: 35th Annual Symposium on Foundations of Computer Science, November 20–22, 1994, Santa Fe, New Mexico*, Seiten 592–603, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1994. IEEE Computer Society Press.

[Kob92] KOBAYASHI, K.: *On Malign Input Distributions for Algorithms*. Lecture Notes in Computer Science, 650:239–??, 1992.

[Kra78] KRAPCHENKO: *Depth and Delay in a Network*. Soviet Math. Dokl., 19(4):1006–1009, 1978.

[LBSV93] LAM, W. K. C., R. K. BRAYTON und A. L. SANGIOVANNI-VINCENTELLI: *Circuit Delay Models and Their Exact Computation Using Timed Boolean Functions*. In: IEEE, ACM-SIGDA; (Herausgeber): *Proceedings of the 30th ACM/IEEE Design Automation Conference*, Seiten 128–134, Dallas, TX, Juni 1993. ACM Press.

[Lev84] LEVIN, LEONID A.: *Problems, complete in "average" instance*. In: ACM (Herausgeber): *Proceedings of the sixteenth annual ACM Symposium on Theory of Computing, Washington, DC, April 30–May 2, 1984*, Seiten 465–465, New York, NY, USA, 1984. ACM Press.

[LF80] LADNER, RICHARD E. und MICHAEL J. FISCHER: *Parallel Prefix Computation*. Journal of the ACM, 27(4):831–838, Oktober 1980.

[LP90] LEIGHTON, T. und C. G. PLAXTON: *A (fairly) simple circuit that (usually) sorts*. In: IEEE (Herausgeber): *Proceedings: 31st Annual Symposium on Foundations of Computer Science: October 22–24, 1990, St. Louis, Missouri*, Band 1, Seiten 264–274, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1990. IEEE Computer Society Press.

[LV92] LI, MING und P. M. B. VITANYI: *Average case complexity under the universal distribution equals worst-case complexity*. Information Processing Letters, 42(3):145–149, Mai 1992.

[LV93] LI, MING und PAUL M. B. VITANYI: *An Introduction to Kolmogorov Complexity and Its Applications*. Springer-Verlag, Berlin, 1993.

[Mil93] MILTERSEN, PETER BRO: *The Complexity of Malign Measures*. SIAM Journal on Computing, 22(1):147–156, Februar 1993.

[MSL92] MITCHELL, DAVID, BART SELMAN und HECTOR LEVESQUE: *Hard and easy distribution of SAT problems*. In: *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI'92)*, Seiten 440–446, 1992.

[Pap94] PAPADIMITRIOU, CHRISTO H.: *Computational Complexity*. Addison-Wesley, USA, 1994.

[Pre93] PRETOLANI, DANIELE: *A linear time algorithm for unique Horn satisfiability*. Information Processing Letters, 48(2):61–66, November 1993.

[Rei99] REISCHUK, K. RUEDIGER: *Komplexitaetstheorie, Band 1: Grundlagen*. B.G. Teubner Stuttgart, Leipzig, 1999.

[RS93] REISCHUK, RÜDIGER und CHRISTIAN SCHINDELHAUER: *Precise Average Case Complexity*. In: *10th Annual Symposium on Theoretical Aspects of Computer Science*, Band 665 der Reihe *lncs*, Seiten 650–661, Würzburg, Germany, 25–27 Februar 1993. Springer.

[RSST97] ROBERTSON, SANDERS, SEYMOUR und THOMAS: *The Four-Colour Theorem*. JCTB: Journal of Combinatorial Theory, Series B, 70, 1997.

[SKC94] SELMAN, BART, HENRY A. KAUTZ und BRAM COHEN: *Noise strategies for improving local search*. In: *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI'94)*, Seiten 337–343, Seattle, 1994.

[WB92] WANG, JIE und JAY BELANGER: *Average Case Intractability of Some Classical Problems*. Technischer Bericht, Department of Mathematics and Computer Science, Wilkes University, Wilkes-Barre, PA 18766, März 1992.

[Wil84] WILF, H. S.: *An $O(1)$ Expected Time Algorithm for the Graph Coloring Problem*. Information Processing Letters, 18:119–121, 1984.

[Wil87] WILF, H.: *Algorithms and Complexity*. PHI Series in Computer Science. Prentice Hall, 1987.

[Yam97] YAMAKAMI: *Average Case Computational Complexity Theory*. In: *ECCCTH: Electronic Colloquium on Computational Complexity, theses*, 1997.