Randomisierte und approximative Algorithmen für harte Berechnungsprobleme

Marek Karpinski

ausgearbeitet von

Carsten Dorgerloh

Christoph Günzel

Mathias Hauptmann

Peter Wegner

Jürgen Wirtgen

- 2007 -

Fünfte, neu bearbeitete Auflage 2007.

Universität Bonn

Römerstraße 164

53117 Bonn

Vorwort

Dieses Skript ist aus meinen Vorlesungen *Randomisierte und approximative Algorithmen für harte Berechnungsprobleme* im Wintersemester 96/97 und *Approximationsalgorithmen für NP-harte Optimierungsprobleme* im Wintersemester 98/99 an der Universität Bonn hervorgegangen. Ich möchte mich an dieser Stelle bei Carsten Dorgerloh, Christoph Günzel, Mathias Hauptmann, Peter Wegner und Jürgen Wirtgen für deren Hilfe bei der Erstellung des Skriptes bedanken.

Im Sommersemester 2000 wurde das Skript neu bearbeitet und auch noch um einige neue hier relevante Resultate erweitert.

Im Sommersemester 2007 wurde das Skript erneut erweitert und aktualisiert.

Bonn, im Sommersemester 2007 Marek Karpinski

Inhaltsverzeichnis

Kapitel 1

Grundlagen

In den folgenden Abschnitten werden wir grundlegende Begriffe einführen, mit denen sich dieses Skript beschäftigt. Zum einen definieren wir die für Approximationen unabdingbaren Begriffe wie Gütegarantie, Approximationsschema, lineare- und Gap-Reduktionen. Zum anderen werden die klassischen diskreten Strukturen definiert, auf denen die Mehrzahl der Approximationsprobleme definiert sind.

1.1 Einführung

In der klassischen Algorithmentheorie werden oft die folgenden Entscheidungsprobleme behandelt:

Gegeben sei eine Menge L und ein Element (Instanz) x aus dem Universum. Ist x ein Element von L?

Wir wollen uns hier zusätzlich mit folgenden allgemeinen Optimierungsproblemen beschäftigen:

Gegeben sei eine Lösungsmenge L und eine Gewichtsfunktion w über $\mathbb{R}$ (oder $\mathbb{Q}$). Bestimme *das x, für das gilt, daß $w(x) = \min_{y \in L} w(y)$ (oder $w(x) = \max_{y \in L} w(y)$)*

Man kann die meisten Optimierungsprobleme mit Hilfe von Entscheidungsproblemen lösen, so daß die Komplexität für das Bestimmen der Optimallösung *polynomiell* mit der *Entscheidungskomplexität* verknüpft ist. Wir werden auch im Laufe der Vorlesung sehen, daß man im allgemeinen wahrscheinlich nicht immer damit rechnen kann, daß diese Probleme effizient exakt gelöst werden können.

In der Praxis genügt es jedoch häufig, nur eine *gute angenäherte* Lösung zu berechnen. Die Berechnungsverfahren die dieses tun, werden auch Approximationsalgorithmen genannt.

Für eine Vielzahl von Optimierungsproblemen werden wir solche Approximationsalgorithmen kennenlernen und analysieren. Häufig scheint es aber genauso schwer zu sein, eine gute Approximation an die Optimallösung zu finden, wie das Finden dieser. Aus diesem Grund werden Approximationskomplexitätsklassen eingeführt, für die wir Härte-Resultate beweisen können.

Warum beschäftigt man sich mit Approximationsalgorithmen? Einer der Motivationspunkte war das Lösen von kombinatorischen Optimierungsproblemen. Es gab dabei zahlreiche Erfolge hinsichtlich des Bestimmens der Optimallösung einiger Probleme. Als Beispiele sind dort kürzeste Wege, spannende Bäume, Matchings, Flüsse etc. zu nennen. Auf der anderen Seite gibt es Probleme, für die bis heute keine *polynomzeit-beschränkte* (”*polynomielle*”) Algorithmen bekannt sind, die sie optimal lösen. Ursprünglich wurde der Begriff der NP-Vollständigkeit [C71] eingeführt, um den Schwierigkeitsgrad mancher kombinatorischer Optimierungsprobleme [K72] exakt auszudrücken. Es stellte sich heraus, daß wenn eines dieser Probleme in polynomieller Zeit gelöst werden kann, dieses auch für alle Probleme dieser Klasse möglich ist. Man versuchte schließlich auch Algorithmen zu entwickeln, die approximative Lösungen dieser Probleme in polynomieller Zeit berechnen. Nach anfänglichen Erfolgen stellte sich heraus, daß es für eine große Teilmenge der Optimierungsprobleme scheinbar unmöglich ist, gute Approximationsalgorithmen zu finden. Man kann die Optimierungsprobleme (nach Gesichtspunkten der Approximierbarkeit) in etwa drei Gruppen aufteilen:

1. Probleme, für die es polynomielle Approximationsalgorithmen beliebiger Genauigkeit (s.g. *PTAS* vgl. Def. 1.6) gibt.

2. Probleme, für die polynomielle Approximationsalgorithmen mit konstanter Gütegarantie existieren.

3. Probleme, für die noch nicht einmal eine Approximation mit konstanter Gütegarantie gefunden worden sind.

Papadimitriou und Yannakakis definierten in [PY91] die Klasse MAX SNP (vgl. Def. 2.1), die einen großen Anteil der Probleme der Gruppe 2 beinhaltet. Die in ihr enthaltenen Probleme widersetzten sich allen Versuchen, für sie ein PTAS zu konstruieren. Heute wissen wir, daß es tiefgreifende Gründe für das Scheitern mancher solcher Versuche gibt (vgl. Kapitel 2). Ferner entwickelten sie einen für diese Klasse sinnvollen Reduktionsbegriff (vgl. Def. 1.9).

1.2 Notationen

Zum besseren Umgang mit dem asymptotischen Verhalten von Funktionen führen wir eine vereinfachte Notation ein. Solche Funktionen sind für unsere Zwecke meistens Funktionen, die die *Rechenzeit*, den *Speicherbedarf*, den *Prozessorenverbrauch* und den Verbrauch von *Zufallsbits* widerspiegeln, also den Verbrauch von Resourcen, die zur Lösung eines Problems notwendig sind.

Definition 1.1 *Sei f eine Funktion $f : I\!N \to I\!N$. Dann definieren wir die folgenden Klassen von Funktionen.*

$$O(f) = \{g : I\!N \to I\!N | \exists c > 0 \quad \exists n_0 \in I\!N \quad \forall n > n_0 \quad [g(n) \le cf(n)]\}.$$

Für Funktionen $g \in O(f)$ gibt f also bis auf einen konstanten Faktor eine obere Schranke für g an, so z.B. für die Laufzeit und den Speicherverbrauch eines Algorithmus.

$$\Omega(f) = \{g : I\!N \to I\!N | \exists c > 0 \quad [g(n) \ge cf(n) \text{ für unendlich viele } n]\}.$$

Für Funktionen $g \in \Omega(f)$ gibt f eine untere Schranke für g an. Die Struktur der Definition unterscheidet sich jedoch von der von $O(f)$. Dadurch ist es möglich, enge untere Schranken anzugeben. Außerdem sei

$$\Theta(f) = \Omega(f) \cap O(f).$$

Mit Hilfe dieser Definitionen haben wir Klassen von Funktionen charakterisiert, deren Wachstum durch f majorisiert bzw. minorisiert wird, bzw. deren Wachstum gleich dem Wachstum von f ist. Wir führen folgende Konvention ein. Wir schreiben

$$g = O(f) \text{ statt } g \in O(f) \text{ und}$$

$$g = \Omega(f) \text{ statt } g \in \Omega(f).$$

Definition 1.2 *Sei f wie oben. Dann definieren wir auch*

$$o(f) = \{g | \lim_{n \to \infty} \frac{g(n)}{f(n)} = 0\}$$

$$\omega(f) = \{g | \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0\}.$$

Damit gilt offensichtlich: $f \in o(g) \Leftrightarrow g \in \omega(f)$.

Definition 1.3 *Sei f wie oben. Sei* $\log : I\!N_0 \to I\!N$ *definiert durch*

$$\log(n) = \begin{cases} 1 & \textit{falls } n \leq 1 \\ \lceil \log_2(n) \rceil & \textit{falls } n \geq 2 \end{cases}$$

und $\log^k$ *durch* $\log^k(n) = (\log(n))^k$. *Dann definieren wir die Klasse $O^\sim(f)$ durch*

$$O^\sim(f) = \{g : I\!N \to I\!N \mid \exists\, k \ : g \in O(f \log^k(f))\}.$$

Diese Klasse besteht aus allen Funktionen, deren Wachstum unter Vernachlässigung logarithmischer Terme von f majorisiert wird.

Die in diesem Skript benutzten Maschinenmodelle und Komplexitätsklassen entsprechen denen, die auch im Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther)) benutzt worden sind.

1.3 Grundlagen der Approximationstheorie

Wir wollen in diesem Abschnitt die Begriffe *Optimierungsproblem*, *Approximationsgüte* und *Approximationsalgorithmen* einführen und formalisieren (Siehe auch [H97b].

Definition 1.4 (Max(Min)-Problem) *Ein Maximierungsproblem ist eine Menge P von Instanzen $I = (F, c)$. F ist dabei die Menge der zulässigen Lösungen und $c : F \to \mathbb{R}$ ist eine Kostenfunktion, die uns für jede zulässige Lösung $s \in F$ die Kosten $c(s)$ angibt.*

Gegeben sei eine Instanz $I = (F, c) \in P$. Wir wollen nun ein $\bar{s} \in F$ berechnen, so daß

$$c(\bar{s}) = \max_{s \in F} c(s) = OPT(I).$$

Minimierungsprobleme werden analog definiert.

Nun ist es häufig NP-schwer, diese *Optimallösung* ($OPT(I)$) zu finden und man ist in der Praxis auch schon an einer *Näherung* an diese Optimallösung zufrieden, die man unter Umständen *schneller* berechnen kann. Es gibt verschiedene Ansätze eine solche Näherung zu beschreiben. Die erste Möglichkeit ist eine *relative Näherung*, deren Qualität im folgenden Approximationsgüte genannt wird.

Definition 1.5 (Approximationsgüte (Approximation Ratio, *A.R.*)) *Gegeben sei ein Optimierungsproblem P. Ein Approximationsalgorithmus $\mathcal{A}$ besitzt eine Approximationsgüte (Approximation Ratio, kurz A.R.) von α, wenn für jede Instanz $I \in P$*

$$\max\left\{\frac{c(\mathcal{A}(I))}{OPT(I)}, \frac{OPT(I)}{c(\mathcal{A}(I))}\right\} \leq \alpha$$

Der Idealfall für die Approximierbarkeit eines Optimierungsproblemes wäre eine Folge von Algorithmen, die das Problem beliebig genau approximieren. Die Laufzeit von Approximationsalgorithmen ist dabei eine Funktion in der Größe $|I|$ der Darstellung der Instanz I (die Anzahl der Bits).

Definition 1.6 (PTAS) *Ein polynomzeitbeschränktes Approximationsschema (kurz PTAS) $\mathcal{A}$ ist eine Familie $(\mathcal{A}_\epsilon, \epsilon > 0)$ von polynomiellen Algorithmen $\mathcal{A}_\epsilon$, mit Approximationsgüte* $1 + \epsilon$.

Die zweite Möglichkeit, eine Näherung zu beschreiben, ist die *additive Näherung* (auch *absolute Approximation* genannt).

Definition 1.7 (Absolute Approximation) *Gegeben sei ein Optimierungsproblem P. Ein Approximationsalgorithmus $\mathcal{A}$ ist ein absoluter ϵ-Approximationsalgorithmus, wenn für jede Instanz $I \in P$*

$$|c(\mathcal{A}(I)) - OPT(I)| \leq \epsilon$$

ist.

Wir werden nun unseren ersten Approximationsalgorithmus kennenlernen. Er findet für eine Instanz des Vertex-Cover Problems eine Lösung, die *höchstens doppelt so teuer* wie die Optimallösung ist. Die Eingabe des Vertex-Cover Problems ist ein ungerichteter Graph $G = (V, E)$. Ziel ist es eine Knotenteilmenge W zu finden, so daß jede Kante mindestens einen Endpunkt in W hat. $|W|$ soll dabei minimal sein. Der folgende Algorithmus approximiert ein optimales Vertex-Cover des Graphen G:

Eingabe: Graph $G = (V, E)$.

Ausgabe: Ein Vertex-Cover $VC(G)$.

Schritt 1: $C_0 = \emptyset$; $E_0 = E$; $i = 0$;

Schritt 2: While $E_i \neq \emptyset$ do

1. Wähle $e = \{v, w\} \in E_i$;
2. $C_{i+1} = C_i \cup \{v, w\}$;
3. $E_{i+1} = E_i \setminus \{\{v', w'\} : \{v', w'\} \cap \{v, w\} \neq \emptyset\}$;
4. $i = i + 1$;

Schritt 3: $VC(G) = C_i$;

Dieser Algorithmus wird auch Matchingalgorithmus für Vertex-Cover genannt. Bezeichne $VC^*(G)$ ein optimales Vertex-Cover von G. Es gilt nun, daß

$$|VC(G)| \leq 2|VC^*(G)|. \tag{1.1}$$

$VC(G)$ enthält $\frac{1}{2}|VC(G)|$ Kanten, die ein Matching bilden. Jedes Vertex-Cover enthält mindestens einen Knoten pro ausgewählte Kante. Insbesondere gilt das für $VC^*(G)$. Also gilt (1.1).

Es gibt aber auch Probleme, die keinen *guten* Approximationsalgorithmus besitzen. Als Beispiel sei da das *Traveling Salesman Problem* (TSP) genannt: Eingabe ist eine n-elementige Clique $G = K_n$ und eine Gewichtsfunktion $w : E(G) \to I\!R$. Ziel ist es, eine Rundreise in G zu finden, die jeden Knoten genau einmal durchläuft und minimales Gewicht hat.

Satz 1.1 *Unter der Bedingung, daß $NP \neq P$ ist, existiert für alle $k \geq 1$ kein polynomieller Approximationsalgorithmus für das TSP mit einem A.R. von k.*

Beweis: Angenommen es existiert ein polynomieller Approximationsalgorithmus $\mathcal{A}$ mit

$$\mathcal{A} \leq k\ OPT.$$

Wir zeigen, daß wir dann auch das NP-vollständige Hamiltonische Kreisproblem (HC) in polynomieller Zeit lösen können: Sei $G = (V, E)$ eine Instanz von HC. Wir konstruieren eine Instanz des TSP wie folgt: Wir betrachten die Clique K_n auf n Knoten ($n = |V|$) und definieren die Kostenfunktion als $w(i,j) = 1$, wenn $\{i,j\} \in E$, und $w(i,j) = 2 + (k-1)n$ sonst. Wende nun $\mathcal{A}$ auf diese Instanz an. Wenn G einen hamiltonischen Kreis enthält, dann ist OPT gleich n, sonst mindestens $n + 1 + (k-1)n = kn + 1$. Da $\mathcal{A}$ einen $A.R.$ von k hat, können wir somit entscheiden, ob G einen solchen Kreis hat oder nicht. ■

Es gibt aber Spezialfälle des TSP, für die wir polynomielle Approximationsalgorithmen angeben können (siehe auch Kapitel 13.1). Als einfaches Beispiel betrachten wir das metrische TSP. Als Einschränkung haben wir hier, daß die Kostenfunktion die Dreiecksungleichung erfüllen muß, $w : E(G) \to \mathbb{R}_0^+$. Es ist bekannt, daß man minimal spannende Bäume in polynomieller Zeit konstruieren kann. Wenn man eine Rundreise an einer beliebigen Stelle auftrennt, erhält man einen Baum. Somit gilt für die Kosten t eines minimal spannenden Baumes, daß sie höchstens so hoch wie die Kosten der minimalen Rundreise r sind. Haben wir nun einen solchen Baum, durchmustern wir ihn in einer Tiefensuche. Jeweils das erste Auftreten eines Knotens wird dabei notiert. Die Kostenfunktion erfüllt die Dreiecksungleichung. Jede Kante wird höchstens zweimal benutzt: Das erste Mal beim Erreichen eines neuen Knotens, das zweite Mal beim Berechnen der Kosten zum nächsten Knoten der angegebenen Rundreise. Somit sind die Kosten dieser Rundreise höchstens doppelt so hoch wie die Kosten der optimalen Rundreise.

1.3.1 GP-Reduktionen

Wir wollen in diesem Skript für Optimierungsprobleme beweisen, daß es NP-hart ist, eine „*gute*" Approximation an deren Optimallösung zu finden. Der Reduktionsbegriff der polynomiellen Reduktion ist für *Approximationshärtebeweise* offensichtlich zu schwach. Wir benötigen daher hier einen speziellen Reduktionsbegriff.

Definition 1.8 *Seien P_1 und P_2 zwei Maximierungsprobleme. Eine* **GP**-*Reduktion (***GP***steht dabei für Gap-Preserving) von P_1 auf P_2 mit den Parametern (l_1, ρ_1) und (l_2, ρ_2), ist eine in polynomieller Zeit berechenbare Funktion $\phi : \Sigma^* \to \Sigma^*$ mit folgenden Eigenschaften: Für jede*

Instanz x_1 von P_1 ist $x_2 = \phi(x_1)$ eine Instanz von P_2, so daß

$$\begin{aligned} OPT(x_1) \geq l_1 \quad &\Rightarrow \quad OPT(x_2) \geq l_2 && (1.2) \\ OPT(x_1) < \frac{l_1}{\rho_1} \quad &\Rightarrow \quad OPT(x_2) < \frac{l_2}{\rho_2} && (1.3) \end{aligned}$$

Hierbei sind l_1, ρ_1 abhängig von $|x_1|$ und l_2, ρ_2 abhängig von $|x_2|$. Wir nennen ρ_1 die GAP von P_1 und ρ_2 die GAP von P_2. l_1 und l_2 können als relative Güteangaben interpretiert werden, während ρ_1 und ρ_2 normalerweise > 1 sind.

1.3.2 Die L-Reduktion

Die im letzten Abschnitt definierte GP-Reduktion ist offenbar geeignet, Nichtapproximierbarkeitsresultate zu übertragen. Wenn jedoch $P_1 \leq_{GP} P_2$ gilt und A ein Approximationsalgorithmus für das Optimierungsproblem P_2 ist, so kann man im allgemeinen hieraus keinen Approximationsalgorithmus für das Problem P_1 erhalten.

Wir werden nun einen weiteren Reduktionsbegriff einführen, der genau dieses ermöglicht.

Definition 1.9 (Lineare Reduktion (L-Reduktion)) *Gegeben seien zwei Optimierungsprobleme P_1, P_2. P_1 heisst L-reduzierbar auf P_2 genau dann, wenn es in polynomieller Zeit berechenbare Funktionen $f, g\colon \Sigma^* \to \Sigma^*$ und Konstanten $\alpha, \beta > 0$ gibt, so dass gilt:*

1. *Zu jeder Instanz x von P_1 ist $f(x)$ eine Instanz von P_2.*
2. *Zu jeder Lösung y der Instanz $f(x)$ von P_2 ist $g(x, y)$ eine Lösung der Instanz x von P_1.*
3. *$OPT(f(x)) \leq \alpha OPT(x)$*
4. *Wenn $c(y)$ die Kosten einer Lösung y zur Instanz $f(x)$ von P_2 sind und $c(g(x, y))$ die Kosten der zugehörigen Lösung $g(x, y)$, so gilt*

$$|OPT(x) - c(g(x, y))| \leq \beta \cdot |OPT(f(x)) - c(y)|$$

Unmittelbar aus der Definition folgt, daß die L-Reduktion transitiv ist. Falls das Problem P_1 ein Minimierungsproblem ist, ist sie sogar approximationsbewahrend.

Lemma 1.1 *Wenn P_1 ein Minimierungsproblem ist, P_2 ein Optimierungsproblem (Max. oder Min.), P_1 auf P_2 L-reduzierbar ist mit Konstanten α, β und ein polynomiell zeitbeschränkter Algorithmus mit Approximationsgüte $r = 1 + \epsilon$ für P_2 existiert, dann existiert ein polynomiell zeitbeschränkter Algorithmus für P_1 mit Approximationsgüte $r' = 1 + \alpha\beta\epsilon$.*

Beweis:

Es sei x eine Instanz von P_1, und es gelte $P_1 \leq_L P_2$ vermöge f, g mit Konstanten α, β. Es sei A ein polynomiell zeitbeschränkter Approximationsalgorithmus mit Approximationsgüte $r = 1 + \epsilon$ für P_2. Wir betrachten den Algorithmus A', der als Ausgabe $g(A'(f(x))) =: y'$ mit Kosten $c_1(y')$ liefert. Es sei y die Ausgabe von Algorithmus A bei Eingabe $f(x)$. Wir betrachten zuerst den Fall, dass P_2 ein Minimierungsproblem ist. Dann gilt

$$\begin{aligned}
|c_1(y) - OPT(x)| &= c_1(y) - OPT(x) \ \leq \ \beta \cdot |c_2(y) - OPT(f(x))| \\
&= \beta \cdot (c_2(y) - OPT(f(x))) \\
&\leq \beta \cdot ((1+\epsilon) - 1) \cdot OPT(f(x))) \\
&\leq \beta \cdot ((1+\epsilon) - 1) \cdot \alpha \cdot OPT(x)
\end{aligned}$$

und somit $c_1(x) \leq (1 + \alpha \cdot \beta \cdot \epsilon) \cdot OPT(x)$.

Für den Fall, dass P_2 ein Maximierungsproblem ist, ergibt sich

$$\begin{aligned}
|c_1(y) - OPT(x)| &= c_1(y) - OPT(x) \ \leq \ \beta \cdot |c_2(y) - OPT(f(x))| \\
&= \beta \cdot (OPT(f(x)) - c_2(y)) \\
&\leq \beta \cdot \left(1 - \frac{1}{1+\epsilon}\right) \cdot OPT(f(x)) \\
&\leq \alpha \cdot \beta \cdot \frac{\epsilon}{1+\epsilon} \cdot OPT(x)
\end{aligned}$$

und somit $c_1(x) \leq (1 + \alpha \cdot \beta \cdot \epsilon) \cdot OPT(x)$. ■

Bemerkung 1.1 *Es ist leicht zu sehen, daß eine* **L**-*Reduktion auch eine GP-Reduktionen ist. Die Umkehrung gilt im allgemeinen nicht.*

Wir haben gesehen, dass L-Reduktionen von Minimierungsproblemen auf Optimierungsprobleme Approximierbarkeit übertragen. Der Ansatz wie im Beweis von Lemma 1.1 funktioniert nicht, wenn P_1 ein Maximierungsproblem ist. Der Grund ist die sehr strenge Bedingung der linearen Abhängigkeit der Zielfunktionswerte und Optima. In der Tat legen Resultate von Crescenzi et al. [CKST99] es nahe, daß für Maximierungsprobleme die L-Reduktion wohl nicht approximationsbewahrend ist.

1.3.3 Die E-Reduktion

In diesem Abschnitt führen wir einen weiteren Reduktionsbegriff ein, der die Nachteile der L-Reduktion, nicht auf Maximierungsprobleme anwendbar zu sein, dadurch behebt, dass anstatt einer linearen Abhängigkeit der Zielfunktionen und Optima eine lineare Abhängigkeit der Approximationsgüten gefordert wird.

Definition 1.10 (E-Reduktion)
Es seien P und Q Optimierungsprobleme. P heisst E-reduzierbar auf Q (Notation: $P \leq_E Q$) genau dann, wenn es polynomzeit-berechenbare Funktionen $f, g\colon \Sigma^* \to \Sigma^*$ und eine Konstante $\alpha > 0$ gibt, so dass gilt:

1. *Für jede Instanz x von P ist $f(x)$ eine Instanz von Q, und für jede Lösung y zu $f(x)$ ist $y' := g(x, y)$ eine Lösung zur Instanz x von P.*
2. *Es seien $R_P(x, g(x, y))$ und $R_Q(f(x), y)$ die Approximationsgüten von $g(x, y)$ bzw. y. Dann gilt:*
$$R_P(x, g(x, y)) \;\leq\; 1 + \alpha \cdot (R_Q(f(x), y) - 1)$$

Im folgenden Lemma wird die approximationsbewahrende Eigenschaft der E-Reduktion behandelt.

Lemma 1.2 *Es seien P und Q Optimierungsprobleme, es gelte $P \leq_E Q$ mit Konstante α, und A sei ein polynomiell zeitbeschränkter Approximationsalgorithmus für Q mit Approximationsgüte $1+\epsilon$. Dann gibt es einen polynomiell zeitbeschränkten Approximationsalgorithmus mit Approximationsgüte $1+\alpha\cdot\epsilon$ für das Optimierungsproblem P.*

BEWEIS: Dies folgt direkt aus Bedingung 2 in Definition 1.10. ■

Unter gewissen Umständen impliziert die Existenz einer L-Reduktion auch die Existenz einer E-Reduktion:

Lemma 1.3 *Es seien P und Q Optimierungsprobleme, und es gelte $P \leq_L Q$. Weiterhin existiere für ein $\epsilon > 0$ ein polynomiell zeitbeschränkter Approximationsalgorithmus mit Approximationsgüte $1+\epsilon$ für P. Dann gilt auch $P \leq_E Q$.*

BEWEIS: Es gelte $P \leq_L Q$ mit polynomzeit-berechenbaren Funktionen f, g und Konstanten $\alpha, \beta > 0$. Ferner sei A ein polynomiell zeitbeschränkter Approximationsalgorithmus mit Approximationsgüte $1+\epsilon$ für P. Wir werden eine E-Reduktion mit Funktionen g_E, f_E von P auf Q konstruieren. Wir betrachten den Fall, daß P ein Maximierungsproblem ist. Wir unterscheiden zwei Unterfälle. Falls y eine Lösung zur Instanz $f(x)$ von Q ist mit Kosten $c_Q(y)$ so, dass $|OPT(f(x)) - c_Q(y)| \leq \frac{OPT(f(x)}{2\cdot\alpha\cdot\beta}$, so folgt, dass $g(x,y)$ eine Lösung zur Instanz x von P mit Approximationsgüte $r \leq 1+2\cdot\alpha\cdot\beta\cdot\epsilon$ ist. Andernfalls können wir die Approximationsgüte durch $1+\frac{1}{2\alpha\beta}$ nach unten abschätzen, so dass wir für die Ausgabe y' des Algorithmus A bei Eingabe x die Approximationsgüte durch $2\cdot\alpha\cdot\beta\cdot\epsilon\cdot r$ beschränken können. Wir setzen also $f_E(x) = f(x)$ für alle Instanzen x von P und $g_E(x,y) = g(x,y)$, falls $c_A(g(x,y)) \geq c_A(y')$ und $g_E(x,y) = y'$ sonst. ■

Kapitel 2

Die Komplexitätsklasse MAX-SNP

Wir wollen in diesem Kapitel eine Komplexitätsklasse einführen, die besser mit Approximierbarkeit der in ihr enthaltenen Probleme umgehen kann, als die klassischen Klassen P oder NP.

Diese Klassen sind durch den Begriff der Berechenbarkeit definiert, der uns zum folgenden Problem führt:

Es gibt scheinbar keine einfache Definition von einer approximativ korrekten Berechnung.

Wenn wir zum Beispiel die Anzahl der korrekten Bits in der Eingabe als Optimierungsmaß nehmen, dann gibt es scheinbar keinen Reduktionsbegriff, der die Approximierbarkeit erhält.

2.1 MAX-SNP

Fagin [F74] zeigte, daß die NP-Klasse durch die Menge aller Prädikate auf Strukturen G definierbar ist, die in der Form

$$\exists S \quad : \quad \psi(G, S) \tag{2.1}$$

dargestellt sind. Dabei sei S eine Struktur und ψ eine Formel 1. Ordnung. Wir können nun (2.1) in die folgende äquivalente Form überführen:

$$\exists S[\forall \bar{x} \exists \bar{y} \quad : \quad \phi(G, S, \bar{x}, \bar{y})] \tag{2.2}$$

Wenn wir den zweiten Existenzquantor entfernen, erhalten wir eine Unterklasse von NP, SNP (auch *strict-NP* genannt). Zum Beispiel ist k-SAT ist in SNP enthalten: Seien $C_0, C_1, ..., C_k$ die Relationen, für die gilt, daß C_j alle Klauseln mit j negierten Literalen enthält. Wir können dann k-SAT als

$$\begin{aligned} \exists S : \forall (x_1, ..., x_k)[(\quad & (x_1, ..., x_k) \in C_0 \Rightarrow x_1 \in S \vee ... \vee x_k \in S) \\ \wedge \quad & ... \\ \wedge \quad & (x_1, ..., x_k) \in C_k \Rightarrow x_1 \notin S \vee ... \vee x_k \notin S)] \end{aligned}$$

ausdrücken.
Anstatt darauf zu bestehen, daß ϕ für alle x erfüllt ist, können wir verlangen, daß ϕ für möglichst viele der x erfüllt werden kann und erhalten somit

Definition 2.1 (MAX-SNP **[PY91]**) *Ein Maximierungsproblem P ist in MAX-SNP enthalten, wenn eine Folge von Relationsymbolen $G_1, ..., G_m, S$ und eine quantorenfreie Formel*

$$\Phi(G_1, ..., G_m, S, x_1, ..., x_k)$$

existieren, so daß

- *ein polynomieller Algorithmus existiert, der für alle Instanzen $x \in P$ ein Universum U und eine Folge von Relationen $G_1^U, ..., G_m^U$ konstruiert. Die Stelligkeit von G_i^U entspricht dabei der Stelligkeit von G_i für alle i.*
- *Wenn $OPT(x)$ die Optimallösung von $x \in P$ ist, dann gilt*

$$OPT(x) = \max_{S^U} |\{(x_1, ..., x_k) \in U^k : \Phi(G_1^U, ..., G_m^U, S^U, x_1, ..., x_k)\}|.$$

S^U hat dabei dieselbe Stelligkeit wie S.

Als Beispiel für ein Problem, das in MAX-SNP enthalten ist, nennen wir das MAX-CUT Problem: Gegeben sei ein ungerichteter Graph $G = (V, E)$. Wir wollen eine Knotenteilmenge $V_1 \subseteq V$ finden, so daß der Schnitt $E(V_1, V \setminus V_1)$ maximal viele Kanten enthält. Wir definieren zunächst das zweistellige Relationensymbol G_1 und das einstellige Relationssymbol S mit

$$\begin{aligned} G_1(u,v) = \textsc{True} \quad &\Leftrightarrow \quad \{u,v\} \in E \\ S(v) = \textsc{True} \quad &\Leftrightarrow \quad v \in V_1. \end{aligned}$$

Die Formel Φ sei definiert durch

$$\Phi(G_1, S, u, v) = (u \neq v) \wedge G_1(u,v) \wedge (S(u) \neq S(v)).$$

Zu einer gegeben Knotenteilmenge $V_1 \subseteq V$ und einem Paar $(u,v) \in V^2$ ist $\Phi(G_1, S, u, v)$ genau dann wahr, wenn genau ein Knoten aus $\{u,v\}$ in V_1 ist, und eine Kante $\{u,v\}$ in E existiert. Somit können wir das MAX-CUT Problem für G ausdrücken als

$$\max_{S^V} |\{(u,v) \in V^2 : \Phi(G_1^V, S^V, u, v)\}|.$$

Definition 2.2 *Ein Problem P ist MAX-SNP-hart, wenn für es für jedes Problem $P' \in MAX$-SNP eine* **L**-*Reduktion von P' auf P gibt. Wenn $P \in MAX$-SNP und MAX-SNP-hart ist, dann nennen wir P MAX-SNP-vollständig.*

Bemerkung 2.1 ([H97b]) *Man beachte, daß diese Definition äquivalent zu der folgenden ist: Ein Problem P ist MAX-SNP-hart, wenn für jedes Problem $P' \in MAX$-SNP und für alle Zahlen $l_1, \rho_1 > 1$, Zahlen $l_2, \rho_2 > 1$ existieren, so daß es eine* **GP**-*Reduktion mit Parametern $(l_1, \rho_1), (l_2, \rho_2)$ von P' auf P gibt.*

Man kann zeigen, daß jedes Problem in MAX-SNP einen Approximationsalgorithmus mit einem konstanten *A.R.* besitzt [PY91].

Satz 2.1 ([PY91]) *MAX-$3SAT$ (Definition siehe Appendix) ist MAX-SNP-vollständig.*

Beweis: Wir müssen zeigen, daß

1. MAX-$3SAT$ in MAX-SNP enthalten ist und, daß
2. MAX-$3SAT$ MAX-SNP-hart ist.

Um zu zeigen, daß MAX-$3SAT$ in MAX-SNP enthalten ist, nehmen wir eine beliebige Formel $f = c_1 \wedge ... \wedge c_m$ in $3KNF$. Wir definieren nun die Relationssymbole G_0, G_1, G_2, G_3 durch

$$\begin{aligned}
G_0(x,y,z) &\iff \exists i[c_i = (x \vee y \vee z)],\\
G_1(x,y,z) &\iff \exists i[c_i = (\bar{x} \vee y \vee z)],\\
G_2(x,y,z) &\iff \exists i[c_i = (\bar{x} \vee \bar{y} \vee z)],\\
G_3(x,y,z) &\iff \exists i[c_i = (\bar{x} \vee \bar{y} \vee \bar{z})].
\end{aligned}$$

Ferner definieren wir die Formel Φ_f als

$$\begin{aligned}
&\Phi_f(G_0, G_1, G_2, G_3, S, x, y, z) =\\
&\qquad (G_0(x,y,z) \wedge (S(x) \vee S(y) \vee S(z))\\
&\quad \vee \quad (G_1(x,y,z) \wedge (\neg S(x) \vee S(y) \vee S(z))\\
&\quad \vee \quad (G_2(x,y,z) \wedge (\neg S(x) \vee \neg S(y) \vee S(z))\\
&\quad \vee \quad (G_3(x,y,z) \wedge (\neg S(x) \vee \neg S(y) \vee \neg S(z))).
\end{aligned}$$

S ist dabei die unäre Relation, die die Menge der erfüllten Variablen repräsentiert.

Aus der Konstruktion folgt, daß

$$\max_{S \in \{0,1\}^n} |\{i | c_i(x)\}| = \max_S |\{(x,y,z) : \Phi_f(G_0, G_1, G_2, G_3, S, x, y, z)\}|.$$

Also ist MAX-$3SAT$ in MAX-SNP enthalten.

Wir zeigen nun, daß MAX-$3SAT$ MAX-SNP-hart ist. Sei dazu

$$\max_{S^U} |\{(x_1, ..., x_k) \in U^k : \Phi(G_1^U, ..., G_m^U, S^U, x_1, ..., x_k)\}$$

die Formulierung eins beliebigen Problems P aus MAX-SNP. Wir zeigen nicht direkt, daß es eine **L**-Reduktion von P auf MAX-$3SAT$ gibt, sondern konstruieren eine **L**-Reduktion von P auf MAX-$FUNCTIONAL$-k_Φ-SAT. Danach konstruieren wir eine **L**-Reduktion von MAX-$FUNCTIONAL$-k-SAT auf MAX-$3SAT$. Wir haben dabei zusätzlich gezeigt, daß MAX-$FUNCTIONAL$-k-SAT MAX-SNP-hart ist.

Für $u = (u_1, ..., u_k) \in U^k$ definieren wir

$$\phi_u(S^U) = \Phi(G_1^U, ..., G_m^U, S^U, u_1, ..., u_k)$$

Es gibt nur 3 atomare Ausdrücke, die in den Formeln ϕ_u auftreten:

1. Die Relationen G_i^U, die Eingabe.
2. $=$ zwischen den u_i.
3. Die Relation S_U.

Die Ausdrücke in 1. und 2. können direkt zu TRUE und FALSE evaluiert werden. Also können wir ohne Beschränkung annehmen, daß ϕ_k eine boolesche Kombination von verschiedenen $S^U(u_{i_1}, ..., u_{i_k})$ ist. Da k eine Konstante ist, ist die Größe von ϕ_k auch konstant. Also können wir ϕ_k als boolsche Formel betrachten mit einer konstanten Anzahl von Variablen. Wir schätzen im folgenden diese Anzahl mit $k_\Phi \in O(1)$ ab. Wir erhalten mit

$$\bigwedge_{u=(u_1,...,u_k)\in U^k} \phi_u$$

eine Instanz von MAX-$FUNCTIONAL$-k_Φ-SAT. Die Transformation ist trivialerweise eine **L**-Reduktion . Es bleibt nun eine **L**-Reduktion von MAX-$FUNCTIONAL$-k-SAT auf MAX-$3SAT$ zu konstruieren (Siehe dazu [PY91]). ■

2.2 PCP-Beweissysteme

Wir werden jetzt eine neue und wichtige Charakterisierung der Klasse NP einführen (s. [ALM+92]).

Erinnern wir uns an die Definition der Klasse NP: Sie beinhaltet alle Sprachen, die von nichtdeterministischen Turingmaschinen in polynomieller Zeit erkannt werden können. Man sieht leicht ein, daß folgende Definition äquivalent ist: Eine Sprache L ist in NP, wenn für ein $x \in L$ ein Beweis π (z.B. eine Kodierung der Berechnung) existiert, so daß eine deterministische Turingmaschine diesen Beweis in polynomieller Zeit auf Korrektheit überprüfen kann. Ein einfaches Beispiel ist 3-COLOR: Ein Beweis wäre eine Kodierung der Färbung. Bei gegebenen Graphen und dieser Färbung können wir nun in polynomieller Zeit verifizieren, ob diese Färbung eine korrekte 3-Färbung der Eingabe ist. Hier sieht man jedoch schon die Beschränkung dieser Charakterisierung: wir müssen den gesamten Beweis (die Färbung) kennen, damit wir entscheiden können, ob diese auch korrekt ist. Kommen wir auch mit weniger Information aus? Hier setzt die neue Charakterisierung von Arora et al. an.

Wir werden sehen, daß für Sprachen aus NP polynomiell lange Beweise in Form von Orakeln existieren, die *effizient* (randomisiert) auf Richtigkeit überprüft werden können. *Effizient* heißt hier, daß nur konstant viele Bits des Beweises untersucht werden müssen. Ferner werden bei Eingabelänge n nur $O(\log(n))$ Zufallsbits benötigt. Der Begriff der *randomisiert überprüfbaren Beweise* (vgl. Def 2.3-2.4) wurde 1992 von Arora und Safra [AS92b] als eine Variante der *randomisierten Orakel-Maschinen* von Fortnow et al. [FRM88] bzw. der *transparenten Beweisen* von Babai et al. [BFLS91] eingeführt. Alle diese Modelle sind wiederum Varianten der *interaktiven Beweissysteme* und der *Multi Prover Protokolle* [BGKW88].

Definition 2.3 (Beschränkte Tester)

- *Ein Tester ist eine randomisierte polynomzeitbeschränkte Turingmschine M (siehe dazu Kapitel 4), mit Zugriff (via einem Orakel) auf eine binäre Zeichenkette π, die einen Beweis für die Zugehörigkeit zu einer Sprache repräsentiert. M kann dabei auf* **jedes**

Bit von π zugreifen (via einem "random access" Orakel). Ein Tester arbeitet in zwei Phasen. Erst berechnet er randomisiert eine Folge von Adressen für das Orakel und liest die Bits unter den berechneten Adressen auf π. In der zweiten Phase entscheidet der Tester deterministisch "Ja (1)" oder "Nein (0)".

- *Ein Tester M heißt $(r(n), q(n))$-beschränkt, wenn M bei einer Eingabe x der Länge n, nur O(r(n)) Zufallsbits benutzen und O(q(n)) Fragen an das Orakel stellen darf. Sei dabei*

$$M(x,\tau,\pi) := \begin{cases} 1 & , \quad M \text{ akzeptiert } x, \text{ mit Zugriff auf } \pi \text{ und dem Zufallsstring } \tau \\ 0 & , \quad \text{sonst} \end{cases}$$

Definition 2.4 (Randomisiert überprüfbare Beweise)
Eine Sprache $L \subseteq \Sigma^*$ ist in $PCP(r(n), q(n))$, wenn ein $(r(n), q(n))$-beschränkter Tester M existiert, so daß für alle $x \in \Sigma^*$ gilt:

1. *Wenn $x \in L$, dann existiert ein Beweis π_x , so daß*

$$\Pr_{\tau \in \{0,1\}^{r(n)}}[M(x,\tau,\pi_x) = 1] = 1$$

2. *Wenn $x \notin L$, dann gilt für alle Beweise π,*

$$\Pr_{\tau \in \{0,1\}^{r(n)}}[M(x,\tau,\pi) = 1] < \frac{1}{4}$$

(Siehe auch Abbildung 2.1)

Wir können in Definition 2.4 die Konstante 1/4 durch jede beliebige Konstante $0 < \epsilon < 1$ ersetzen und erhalten analog $PCP_\epsilon(r(n), q(n))$, denn

Lemma 2.1 *Für alle Konstanten $0 < \epsilon < 1$ gilt*

$$PCP_\epsilon(r(n), q(n)) = PCP(r(n), q(n)).$$

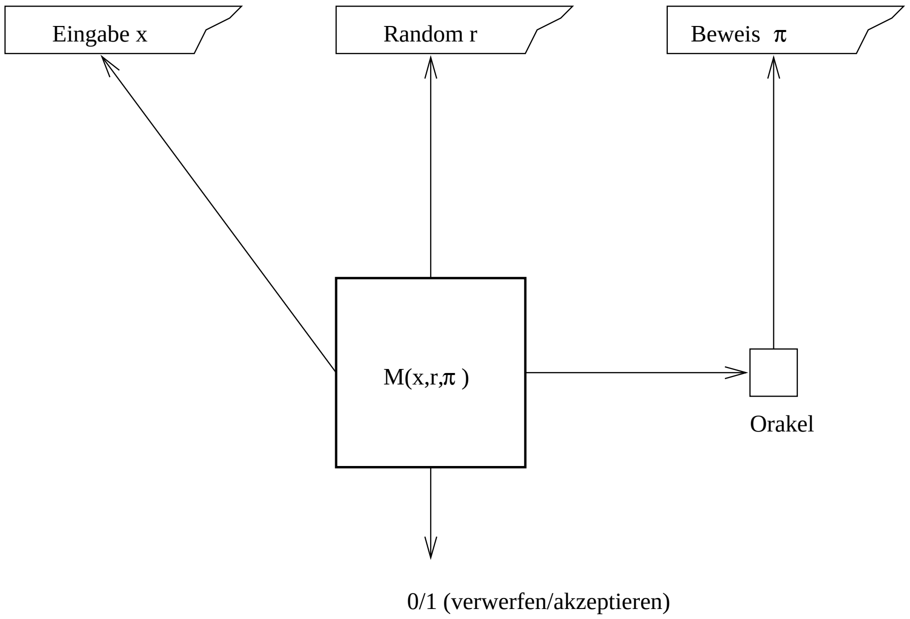

Abbildung 2.1: Ein PCP-Beweissystem.

Beweis: Durch hinreichend (aber konstant) viele unabhängige Läufe unseres Testers und Akzeptanz, wenn alle diese Läufe erfolgreich waren, wird die Fehlerwahrscheinlichkeit kleiner als ϵ^k, wobei k die Anzahl der Läufe ist. Die Resourcen bleiben dabei asymptotisch gleich.

■

Satz 2.2 $NP = PCP(\log n, 1)$

Beweis:

- $\supseteq$: Sei $L \in PCP(\log n, 1)$. Ein Tester für $PCP(\log n, 1)$ kann nur auf $2^{O(\log n)} = n^{O(1)}$ verschiedene Zufallsstrings, die die Beweise repräsentieren zugreifen. Konstruiere eine NTM N, die für ein $x \in \Sigma^*$ den polynomiell langen Beweis π (für $x \in L$) errät, und danach für alle möglichen Zufallsstrings die Verifikationsprozedur M simuliert. N akzeptiert genau dann, wenn alle $n^{O(1)}$ Simulationen akzeptieren ($\Leftrightarrow x \in L$). Aus der Konstruktion folgt $L \in NP$, da N polynomiell zeitbeschränkt ist.

- $\subseteq$: Siehe [ALM$^+$92] und Kapitel 11 .

$\blacksquare$

Obwohl das PCP-Resultat an sich sehr interessant ist und einiges über die Struktur von NP aussagt, hat es auch Auswirkungen auf ein scheinbar verschiedenes Gebiet - auf das Gebiet der Approximationsalgorithmen.

Wir haben die Klasse MAX-SNP eingeführt und gesehen, daß für alle in ihr enthaltenen Probleme Approximationsalgorithmen mit konstanter Gütegarantie existieren. Es stellt sich nun die Frage, ob für jede Konstante $\epsilon > 0$ ein Approximationsalgorithmus mit Gütegarantie $(1+\epsilon)$ existiert. Diese Frage wird als Konsequenz von Theorem 2.2 verneint. Wir beweisen nun Lemma 10.1.

Beweis: von Lemma 10.1. Sei L eine beliebige Sprache aus NP und sei M ein $(\log(n), 1)$-beschränkter Tester für L (Existenz folgt aus Theorem 2.2). Bei gegebener Eingabe x konstruieren wir nun wie folgt eine $3SAT$-Formel Φ_x mit $2^{O(\log(n))}$ Klauseln:
Die einzelnen Bits im Beweis π von L werden als die Variablen von Φ_x betrachtet *(beachte dabei, daß $|\pi| = n^{O(1)}$)*. Für jeden (der $2^{O(\log(n))}$ möglichen) Zufallstrings r von M wird eine Formel ψ_x^r konstruiert, so daß gilt

$$\psi_x^r = 1 \Leftrightarrow M \text{ akzeptiert } x \text{ bei der Wahl von } r \in_R \{0,1\}^{O(\log n)}.$$

Diese Formel hat polynomielle Größe. Daraus folgt, daß die folgende Formel polynomielle Größe hat:

$$\Psi_x = \bigwedge_{r \in \{0,1\}^{O(\log(n))}} \psi_x^r$$

Wir können nun Ψ_x in eine 3SAT-Formel Φ_x polynomieller Größe umwandeln, so daß $\Phi_x \equiv \Psi_x$. Wir erhalten nun:

1. Wenn $x \in L$, dann ist Φ_x erfüllbar.
2. Wenn $x \notin L$, dann verwirft M mit Wahrscheinlichkeit größer als $\frac{3}{4}$. Daraus folgt, daß höchstens ein konstanter Anteil $(1-g)$ der Klauseln gleichzeitig erfüllt werden kann.

Angenommen, es gäbe eine PTAS ($P_\epsilon, \epsilon > 0$) für $MAX - 3SAT$. Wende $P_{\frac{g}{2}}$ auf Φ_x an. Es gilt nun, daß Φ_x genau dann erfüllbar ist, wenn $P_{\frac{g}{2}}$ in polynomieller Zeit eine Belegung von Φ_x berechnet, die mehr als einen Anteil $\geq (1 - \frac{g}{2})$ der Klauseln in Φ_x erfüllt. Somit können wir in polynomieller Zeit entscheiden, ob $x \in L$ ist. ■

Korollar 2.1 *Für alle MAX-SNP-harten Probleme existiert ein $\epsilon > 0$, s.d. eine Approximation mit A.R. $(1 + \epsilon)$ NP-hart ist.*

Kapitel 3

Probabilistische Methoden

Wir wollen in diesem Kapitel einige probabilistische Methoden einführen, die uns später bei komplexeren Beweisen hilfreich sein werden. Zur weiteren Vertiefung der probabilistische Methoden seien [AS92a], [MR95] empfohlen.

3.1 Das Isolierungslemma

Eine Variante des Isolierungslemmas von [MVV87] wurde bereits im Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther) [K91] eingeführt. In Kapitel 10.1 werden wir eine weitere Anwendung des Isolierungslemmas kennenlernen. Wir definieren dort das Unique Clique Problem, eine Variante des Cliquenproblems, und reduzieren das Unique Clique Problem randomisiert auf das Cliquenproblem. Das Unique Clique Problem gehört zu den sogenannten Promiseproblemen.

Definition 3.1 (Gewichtssystem) *Ein* **Gewichtssystem** GS *ist ein Tripel* (Q, F, w)*, wobei*

- *Q eine endliche Menge,*
- *F eine Familie von Teilmenge aus Q, d.h $F \subseteq \mathcal{P}(Q)$ und*
- *w eine Gewichtsfunktion $w : Q \to \mathbb{N}$*

sind.

Definition 3.2 *Die Funktion $\bar{w} : \mathcal{P}(Q) \to \mathbb{N}$ ist definiert durch*

$$\bar{w}(S) = \sum_{x \in S} w(x).$$

Außerdem sei

$$\max(Q, F, w) = \{S \in F | \bar{w}(S) = \max\{\bar{w}(S') | S' \in F\}\}.$$

Wir nennen ein Gewichtssystem eindeutig, falls es unter den Mengen aus F eine eindeutige Menge S mit größtem Gewicht $\bar{w}(S)$ gibt, d.h.

$$\max(Q, F, w) = \{S\}.$$

Wir schreiben dann (falls S eindeutig ist)

$$S = \max(Q, F, w).$$

Nun kommen wir zum

Lemma 3.1 (Gewichtslemma, Isolationslemma (Maximierungsvariante)) *Sei (Q, F, w) ein Gewichtssystem mit $|Q| = n$ und $w : Q \to [1 \ldots 2n]$ eine randomisierte Funktion, die eine unabhängige Gleichverteilung induziert. Dann ist*

$$\Pr[(Q, F, w) \textit{ ist eindeutig}] \geq 1/2$$

Beweis: Sei $Q = \{x_1 \ldots x_n\}$ und seien $w(x_1), \ldots, w(x_n)$ unabhängige und gleichverteilte Zufallsvariablen mit Wertebereich $[1 \ldots 2n]$.

Seien für $1 \leq i \leq n$

$$W_i = \max\{\bar{w}(S) | S \in F, x_i \notin S\} \quad \text{und}$$

$$W_i{}' = \max\{\bar{w}(S \setminus \{x_i\}) | S \in F, x_i \in S\}$$

$$W_i{}'' = \max\{\bar{w}(S) | S \in F, x_i \in S\} = W_i{}' + w(x_i).$$

Damit sei

$$\Delta_i = W_i - W_i{}'.$$

Man beachte, daß die Zufallsvariable Δ_i stochastisch unabhängig von der Zufallsvariable $w(x_i)$ ist, $\forall 1 \leq i \leq n$. Es können nun die folgenden Fälle auftreten:

- falls $\forall S \in \max(Q, F, w): \ x_i \notin S \longrightarrow W_i > W_i{}'' \longrightarrow \Delta_i > w(x_i)$
- falls $\forall S \in \max(Q, F, w): \ x_i \in S \longrightarrow W_i > W_i{}'' \longrightarrow \Delta_i < w(x_i)$
- falls $\exists S, S' \in \max(Q, F, w)$ mit $x_i \in S, x_i \notin S'$, dann ist $\Delta_i = w(x_i)$.

Im letzten Fall, nämlich falls $\Delta_i = w(x_i)$ für ein $1 \leq i \leq n$ gilt, ist das Gewichtssystem nicht eindeutig, da zwei verschiedene gewichtsmaximale Mengen sich um das Element x_i unterscheiden. Die Wahrscheinlichkeit hierfür kann folgendermaßen abgeschätzt werden:

$$\Pr[\exists i \text{ mit } \Delta_i = w(x_i)] \leq \sum_{i=1}^{n} \Pr[\Delta_i = w(x_i)].$$

Da die Zufallsvariablen $w(x_i)$ unabhängig von den Δ_i sind, gilt nun

$$\Pr[w(x_i) = \Delta_i] = 1/2n.$$

Also ist die Wahrscheinlichkeit, daß das Gewichtssystem eindeutig ist, größer gleich $1/2$. ∎

Analog können wir folgende Minimierungsvaraiante beweisen.

Lemma 3.2 (Gewichtslemma, Isolationslemma (Minimierungsvariante)) *Sei* (Q, F, w) *ein Gewichtssystem mit* $|Q| = n$ *und* $w : Q \to [1 \ldots 2n]$ *eine randomisierte Funktion, die eine unabhängige Gleichverteilung induziert. Dann ist*

$$\Pr[\min(Q, F, w) \ \textit{ist eindeutig}] \geq 1/2$$

3.2 Das Bernstein-Lemma

In Kapitel 12 betrachten wir Zählalgorithmen. Ein im Bereich der Zählalgorithmen oft benötigtes Hilfsmittel ist das Bernstein-Lemma (siehe auch [KLM89]). Angenommen wir haben unabhängige Wahrscheinlichkeitsvariablen $Y_1, Y_2, ..., Y_N$, die den Wert $|U|$ mit Wahrscheinlichkeit $|G|/|U|$ annehmen und sonst 0 sind. Es gilt für $Y = \sum_{i=1}^{N} Z_i$, $Z_i := \frac{1}{N} Y_i$,

$$E[Y] = E[1/N \sum_{i}^{N} Y_i] = 1/N \sum_{i}^{N} E[Y_i] = |G|.$$

Wie groß müssen wir nun N wählen, damit Y nicht sehr stark von $E[Y]$ abweicht?

Satz 3.1 (Satz von Bernstein) *Sei $\mu = \frac{|G|}{|U|}$. Dann gilt*

$$\Pr[(1-\epsilon) \cdot |G| \leq Y \leq (1+\epsilon) \cdot |G|] \geq 1 - \delta,$$

wenn die Anzahl der Versuche N größer gewählt wird als

$$\frac{1}{\mu} \cdot \frac{4}{\epsilon^2} \ln(\frac{2}{\delta}).$$

Der Beweis dieses Satzes benötigt einige wahrscheinlichkeitstheoretische Lemmata.

Lemma 3.3 *Seien $Z, Z_1, Z_2, \cdots Z_k$ unabhängige Zufallsvariablen mit der gleichen Verteilung. Seien c und d Konstanten. Dann gilt*

$$E[e^{d(Z_1 + \cdots + Z_k) + c}] = E[e^{dZ}]^k \cdot e^c.$$

Lemma 3.4 *Für jede Zufallsvariable Z und jede Konstante $d \geq 0$ gilt*

$$\Pr[Z \geq 0] \leq E[e^{dZ}].$$

Beweis: Sei die Zufallsvariable

$$Z' = \begin{cases} 1 & \text{falls } Z \geq 0 \\ 0 & \end{cases}$$

Dann ist $Z' \leq e^{dZ}$ und daher $E[Z'] \leq E[e^{dZ}]$. Außerdem gilt $E[Z'] = \Pr[Z \geq 0]$. ■

Im folgenden seien die Zufallsvariablen $Y, Y_1, \cdots Y_N$ $\{0,1\}$-wertige unabhängige Zufallsvariablen mit der gleichen Verteilung und Erwartungswert $E[Y] = \mu$.

Lemma 3.5 *Sei $d \leq 1$ eine Konstante. Dann gilt*

$$E[e^{dY}] \leq e^{\mu d(1+d)}.$$

Beweis: Für den Erwartungswert von e^{dY} gilt

$$E[e^{dY}] = e^d \cdot Pr[y=1] + e^0 \cdot Pr[y=0] = \mu \cdot e^d + (1-\mu).$$

Die Funktion e^x können wir für $x \leq 1$ durch $1 + x + x^2$ nach oben und für $x \geq 0$ durch $1 + x$ nach unten abschätzen. Dadurch ergibt sich

$$E[e^{dY}] \leq \mu \cdot (1 + d + d^2) + (1 - \mu) = 1 + \mu d(1 + d).$$

Benutzen wir die zweite Abschätzung, so erhalten wir

$$E[e^{dY}] \leq e^{\mu d(1+d)}.$$

■

Mit diesem Lemma können wir die Wahrscheinlichkeit für große Abweichungen der Ausgabe des Algorithmus von dem zu berechnenden Wert beschränken. Es gilt das folgende

Korollar 3.1 *Für $\epsilon \leq 2$ gilt*

$$\Pr[\sum_{i=1}^{N} Y_i > (1+\epsilon)\mu N] \leq e^{-\mu\epsilon^2 N/4}.$$

Beweis: Bezeichne

$$(*) = \Pr[\sum_{i=1}^{N} Y_i > (1+\epsilon)\mu N].$$

Aus Lemma 3.4 folgt

$$(*) \leq E[e^{d\cdot(\sum_{i=1}^{N} Y_i-(1+\epsilon)\mu N)}].$$

Nun wenden wir Lemma 3.3 an und es ergibt sich

$$(*) \leq E[e^{dY}]^N \cdot e^{-d(1+\epsilon)\mu N}.$$

Nun schätzen wir $E[e^{dY}]$ mittels Lemma 3.5 ab und setzen $d = \epsilon/2$:

$$(*) \leq e^{-\mu\epsilon^2 N/4}.$$

■

Damit haben wir die Wahrscheinlichkeit für eine Abweichung nach oben abgeschätzt. Nun müssen wir auch die Wahrscheinlichkeit für Abweichungen nach unten begrenzen. Dies geschieht analog.

Lemma 3.6 *Sei $d \leq 1$ eine Konstante. Dann gilt*

$$E[e^{-dY}] \leq e^{-\mu d(1-\frac{d}{2})}.$$

Beweis: Für den Beweis benötigen wir jetzt Abschätzungen der Funktion e^{-x}. Es gilt $1-x \leq e^{-x} \leq 1-x-x^2/2$. Aus diesen Ungleichungen ergibt sich

$$E[e^{-dY}] = \mu e^{-d} + (1-\mu) \leq \mu(1-d-d^2/2) + 1 - \mu = 1 - \mu d(1-d/2) \leq e^{-\mu d(1-d/2)}.$$

■

Nun können wir auch die Wahrscheinlichkeit für ein Abweichen nach unten beschränken.

Korollar 3.2 *Für $\epsilon \leq 2$ gilt*

$$\Pr[\sum_{i=1}^{N} Y_i < (1-\epsilon)\mu N] \leq e^{-\mu\epsilon^2 N/4}.$$

Beweis: Analog zum Beweis zu Korollar 3.1. ■

Mit Hilfe der Korollare 3.1 und 3.2 ergibt sich der Satz von Bernstein.

3.3 Chernoff'sche Schranken

Bevor wir uns mit Sampling und randomisiertem Runden beschäftigen, wollen wir hier Chernoff Schranken besprechen. Chernoff Schranken sind für das Design und die Analyse randomisierter Algorithmen sehr nützlich. Die Analyse des Rundungslemmas 3.10 ist etwas schwieriger, da wir auch negative Koeffizienten erlauben.

Lemma 3.7 (Positive Variablen) *Seien $X_1, \dots, X_n$ unabhängige und gleichverteilte Zufallsvariablen mit*

$$\Pr[X_i = 1] = p = 1 - \Pr[X_i = 0].$$

Weiterhin seien $w_1, \dots, w_n \in [0,1]$ Gewichte. Die Zufallsvariable X ist dann die gewichtete Summe der unabhängigen Versuche, d.h. $X = \sum_{i=1}^n w_i X_i$, und μ der Erwartungswert $\mu = E[X] = \sum_{i=1}^n p_i$. Dann gilt die folgende Ungleichung:

$$\forall t > 0 \ \Pr[|X - \mu| > t] \leq 2e^{-2t^2/n}$$

Beweis: Siehe [MR95]. ■

Korollar 3.3 *Für jedes $f > 0$ gibt es eine Konstante $c > 0$, so daß mit den Voraussetzungen aus Lemma 3.7 gilt:*

$$\Pr[|X - \mu| > c\sqrt{n \log n}] \leq 2n^{-f}.$$

Beweis: Wir wenden Lemma 3.7 mit $t := c\sqrt{n \log n}$ und $c = \sqrt{\frac{f}{2}}$ an. Dann gilt:

$$\Pr[|X - \mu| > c\sqrt{n \log n}] \leq 2e^{-2c^2 \log n} \leq 2n^{-2c^2} \leq 2n^{-f}$$

■

Nun wollen wir wie bereits erwähnt auch negative Koeffizienten zulassen:

Lemma 3.8 (Positive und negative Variablen) *Seien $X_1, \ldots, X_n$ unabhängige und gleichverteilte Zufallsvariablen. Die Menge $\{1, \ldots, n\}$ sei die Vereinigung der beiden disjunkten Mengen I und J mit $0 \leq w_i \leq 1$ für $i \in I$ und $-1 \leq w_j \leq 0$ für $j \in J$. Dann gibt es für jedes $f > 0$ Konstanten $c_1 > 0$ und $c_2 > 0$, so daß für $\mu = E[X]$ und $X = \sum_{i=1}^{n} w_i X_i$*

$$\Pr[|X - \mu| > c_1\sqrt{n \log n}] \leq c_2 n^{-f}$$

gilt.

Beweis: Wir definieren die Zufallsvariablen und ihre Erwartungswerte wie folgt:

$$\begin{aligned} X' &= \sum_{i \in I} w_i X_i \\ \mu' &= E[X'] \\ X'' &= \sum_{j \in J} w_j X_j \\ \mu'' &= E[X''] \end{aligned}$$

Beachte, daß $|X - \mu| \leq |X' - \mu'| + |-X'' - (-\mu'')|$. Nun können wir Korollar 3.3 auf X' und $-X''$ anwenden, die erhaltenen Ungleichungen angemessen kombinieren und erhalten das gewünschte Ergebnis. ■

3.4 Sampling und Randomisiertes Runden

Folgendes Lemma ist eine der Grundlagen des *exhaustiv sampling* in Kapitel 6.2

Lemma 3.9 (Sampling Lemma) *Sei A eine Menge von Konstanten $a_1, ..., a_n$. Definiere $q = \sum_{i=1}^{n} a_i$ und $s = c \log n/\epsilon^2$. Wir wählen nun, unabhängig voneinander und gleichverteilt, s Werte aus A aus. Diese definieren Zufallsvariablen $X_1, ..., X_s$. Sei $X = \sum_{i=1}^{s} X_i$. Dann gilt*

$$\Pr\left[q - \epsilon n \leq \frac{n \sum_{a_i \in S} a_i}{s} \leq q + \epsilon n\right] > 1 - \frac{1}{2n}.$$

Beweis: Es gilt

$$\begin{aligned} \mathrm{E}[X] &= \sum_{i=1}^{s} \mathrm{E}[X_i] \\ &= \frac{s}{n} \sum_{j=1}^{n} a_j \\ &= \frac{s}{n} q \end{aligned}$$

Wir zeigen die 2. Ungleichung, d.h.

$$\Pr\left[\frac{n \sum_{a_i \in S} a_i}{s} > q + \epsilon n\right] < \frac{1}{n}$$

Die 1. Ungleichung wird analog bewiesen. Addition ergibt, daß die Fehlerwahrscheinlichkeit durch $\frac{1}{2n}$ beschränkt ist.

$$\begin{aligned} \Pr\left[\frac{n}{s} X > q + \epsilon n\right] &= \Pr\left[X > \frac{s}{n} q + s\epsilon\right] \\ &= \Pr[X > E[X] + \frac{c \log n}{\epsilon}] \\ \text{(Lemma 3.4)} \quad &\leq \mathrm{E}[\exp(d(\sum_{i=1}^{s} X_i - (\mathrm{E}[X] + c \log n/\epsilon)))] \\ \text{(Lemma 3.3)} \quad &= \frac{\mathrm{E}[\exp(dX)]^s}{\exp(\mathrm{E}[n/q])^s \exp(c \log n/\epsilon)} \\ &\leq \frac{1}{2} \frac{1}{n} \text{ für geeignete Wahl der Konstanten } c \text{ und } d. \end{aligned}$$

■

Während der Ausführung des Algorithmus in Kapitel 6.2 werden wir ein 0/1-Integer Programm lösen müssen. Da dieses Problem NP-vollständig ist, können wir nur die Relaxierung lösen, deren Lösung wir auf 0 oder 1 *runden* müssen. Folgendes Lemma beschreibt das Runden und sagt aus, daß der dabei gemachte Fehler *klein* ist:

Lemma 3.10 (Rundungslemma [R88]) *Sei $x = (x_1, \ldots, x_n)^T$ eine Lösung des linearen Programms*

$$Ax = b,$$

$$0 \leq x_i \leq 1 \quad (i = 1, \ldots, n)$$

wobei die Koeffizienten der Matrix A Konstanten sind. Wir definieren einen zufälligen $0-1$ Vektor $y = (y_1, \ldots, y_n)^T$ mit

$$y_i = \begin{cases} 1 & \textit{mit Wahrscheinlichkeit } x_i \\ 0 & \textit{sonst} \end{cases}$$

Dann gilt mit großer Wahrscheinlichkeit, daß

$$Ay = (b_1 + O(\sqrt{n \log n}), \ldots, b_n + O(\sqrt{n \log n}))^T.$$

Beweis: Die Koeffizienten von A sind Konstanten, sagen wir $a_{ij} \in [-c, c]$ für $1 \leq i, j \leq n$. Definieren wir $\bar{a}_{ij} = a_{ij}/c$ und $\bar{b}_i = b_i/c$, dann können wir zusammen mit

$$\mu_i = E[\sum_{j=1}^{n} \bar{a}_{ij} y_j] = \sum_{j=1}^{n} \bar{a}_{ij} E[y_j] = \sum_{j=1}^{n} \bar{a}_{ij} x_j = \bar{b}_i$$

Lemma 3.8 auf jede Reihe i des Systems anwenden. Damit erhalten wir

$$|\sum_{j=1}^{n} \bar{a}_{ij} y_j - \bar{b}_i| \leq c_1 \sqrt{n \log n}$$

mit einer Wahrscheinlichkeit von mindestens $1 - c_2 n^{-f}$ und entsprechend

$$|\sum_{j=1}^{n} a_{ij} y_j - b_i| \leq c c_1 \sqrt{n \log n} = O(\sqrt{n \log n})$$

mit einer Wahrscheinlichkeit von mindestens $1 - c_2 n^{-f}$. Wählen wir nun $f \geq 2$, dann gilt für das ganze System die Behauptung mit Wahrscheinlichkeit $1 - c_2 n^{-f+1}$. ∎

Kapitel 4

Randomisierte Maschinenmodelle und Algorithmen

In diesem Kapitel beschäftigen wir uns mit den Grundlagen der randomisierten Berechnungen. Dabei hängt die Berechnung und damit unter Umständen auch das Ergebnis der Berechnung von zufälligen Ereignissen ab. Solche Ereignisse werden durch Zufallszahlengeneratoren erzeugt.

Definition 4.1 (Idealer Münzwurf) *Unter einem idealen Münzwurf verstehen wir eine gleichverteilte Zufallsvariable X mit Wertebereich $\{0, 1\}$. Die Werte einer Folge von Auswertungen der Zufallsvariable X seien* **gleichverteilt** *und voneinander* **unabhängig**.

Ein für uns gültiger Zufallszahlengenerator erzeugt einen idealen Münzwurf. Nun können wir unsere bisherigen Modelle geeignet modifizieren, indem wir die Verwendung von Zufallsinformation erlauben, die durch einen gültigen Zufallsgenerator erzeugt wird.
Im folgenden werden die elementaren Definitionen und Rechenregeln mit Zufallsvariablen vorausgesetzt (s. [C74]).

4.1 Probabilistische Turing Maschinen

Wir beginnen mit einer Erweiterung der herkömmlichen deterministischen Modelle.

Operationell besteht eine TM aus einer Zentraleinheit und einer endlichen Anzahl von Bändern. Auf diese Bänder kann die TM mittels Schreib/-Leseköpfe zugreifen. In der Zentraleinheit befindet sich ein endliches Programm, das die Bewegung der Köpfe und die Schreiboperationen in Abhängigkeit von den Bandinhalten an den aktuellen Bandpositionen und dem inneren Zustand der endlichen Kontrolle bewirkt. Je nach Anzahl und Art der Bänder, der erlaubten Bewegungsrichtungen der Köpfe und der Fähigkeit zu Schreib- bzw. Leseoperationen ergeben sich eine Vielzahl verschiedener TM Modelle. Wir betrachten hier nur einfache Modelle, die wir wie folgt formal definieren wollen.

Definition 4.2 *Eine (k-Band) TM M ist ein 5-Tupel $M =< Q, \delta, q_0, F, \Gamma >$ wobei*

- *Q die endliche Menge der Zustände der endlichen Kontrolle,*
- *$q_0 \in Q$ der Startzustand,*
- *$F \subset Q$ die Menge der akzeptierenden Zustände,*
- *Γ das Bandalphabet, d.h. jede Zelle der Bänder beinhaltet ein Element aus Γ ($|\Gamma| \geq 2$), und*
- *δ die Zustandsüberführungsfunktion (das Programm)*

$$\delta : (Q, \Gamma^k) \to (Q, \Gamma^k, \{L, N, R\}^k)$$

sind.

M akzeptiert $x \in \Gamma^*$, falls M hält, d.h. nach einer endlichen Anzahl von Zustandsübergängen in einem Endzustand $q' \in F$ übergeht. Die Menge aller akzeptierten Wörter bildet die von M erkannte Sprache $L(M)$.

$$L(M) := \{x \in \Gamma^* \mid M \text{ akzeptiert } x\}.$$

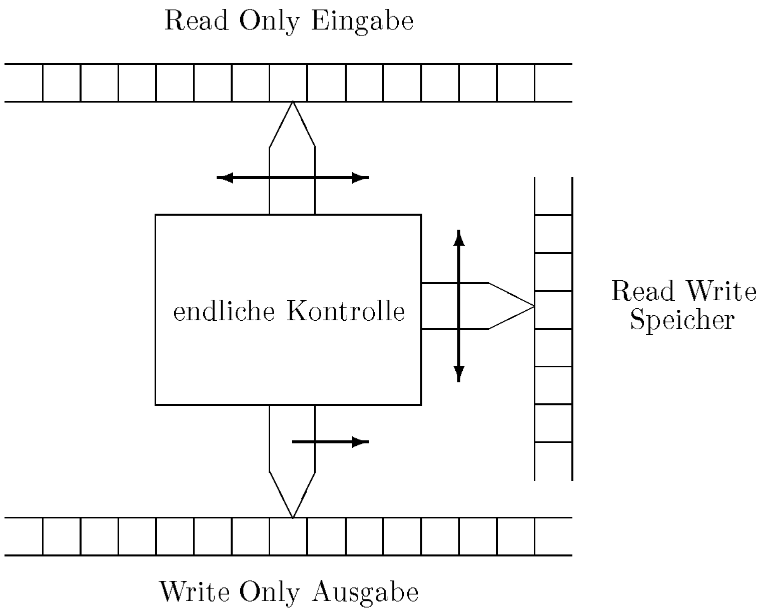

Abbildung 4.1: Offline Turing Maschine

Definition 4.3 (Offline TM)
Eine Offline TM ist eine Turing Maschine mit einem Eingabe-, einem Ausgabe- und einem Arbeitsband. Von dem Eingabeband darf nur gelesen werden und der Lesekopf darf in beide Richtungen bewegt werden. Das Ausgabeband ist eine reines Schreibband, hier darf der Schreibkopf jedoch nur nach rechts bewegt werden, d.h. eine einmal gemachte Ausgabe kann nicht überschrieben und damit korrigiert werden. Das Arbeitsband ist ein normales Band, auf ihm sind sowohl Schreib- als auch Lesezugriffe erlaubt, der Schreib/Lesekopf kann in beide Richtungen bewegt werden.

Abbildung 4.1 zeigt ein Diagramm einer Offline TM. Für weitere Informationen siehe auch [K91]

Definition 4.4 (Probabilistische Turing Maschine (PTM)) *Eine probabilistische TM ist eine Offline TM im deterministischen Sinne mit den folgenden Modifikationen:*

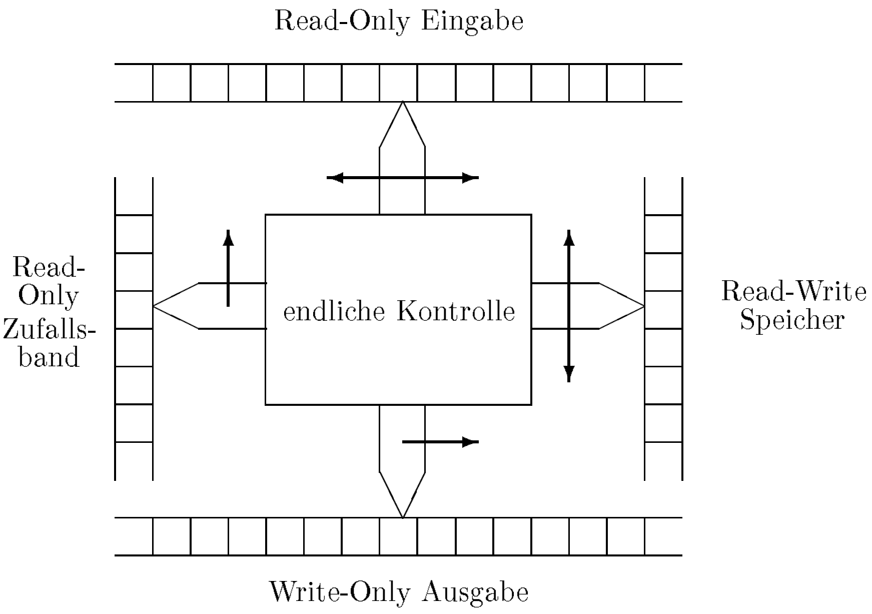

Abbildung 4.2: Probabilistische Turing Maschine

- *Es gibt ein zusätzliches Band, welches eine Folge von* **Zufallsbits** $\in \{0,1\}^*$ *enthält, die von einem gültigen Zufallszahlengenerator erzeugt worden sind. Der Kopf des Zufallsbandes darf nur* **in eine Richtung** *bewegt werden. Somit kann eine Zufallsinformation* **nur einmal** *verwendet werden.*

- *Die Zustandsüberführungsfunktion hängt auch von dem Inhalt der Zelle unter dem Lesekopf des Zufallsbandes ab. Sie bleibt aber deterministisch.*

- *Eine Berechnung ist nur* **gültig**, *falls die Länge der Zufallsfolge mindestens so groß ist wie die Länge der Berechnung.*

Abbildung 4.2 zeigt eine probabilistische Turing Maschine.

Bemerkung 4.1 *Man kann eine probabilistische TM als eine deterministische TM ansehen, falls man das Zufallsband als ein zusätzliches Eingabeband interpretiert.*

Definition 4.5 *Sei M eine PTM über dem Alphabet Σ. Dann induziert M eine partielle Funktion*

$$\Phi_M : \Sigma^* \times \underbrace{\{0,1\}^*}_{\textit{Zufallsbits}} \to \Sigma^*.$$

Diese Funktion ist nicht total, da die Maschine nicht bei jeder Eingabe anhalten muß. So ist $\Phi_M(x, \rho)$ undefiniert, falls die Berechnung länger ist als $|\rho|$.

Wir betrachten eine PTM nicht als deterministische TM, sondern gehen davon aus, daß der Inhalt des Zufallsbandes nicht bekannt ist. Durch die Hinzunahme einer weiteren Resource — dem **Zufall** — ergibt sich die Notwendigkeit den Berechenbarkeitsbegriff für dieses neue Modell zu definieren.

Definition 4.6 (Berechenbarkeitsbegriff für PTM) *Sei M eine PTM. M berechnet eine partielle Funktion*

$$\phi_M : \Sigma^* \to \Sigma^*$$

mit

$$\phi_M(x) = \begin{cases} y & \textit{wenn } \Pr[M(x) = y] > 1/2 \\ \textit{undefiniert} & \textit{wenn es kein solches } y \textit{ gibt.} \end{cases}$$

Dabei ist

$$Pr[M(x) = y] = \lim_{n \to \infty} \frac{|\{\rho \in \{0,1\}^n | \Phi_M(x, \rho) = y\}|}{2^n}$$

Der Definitionsbereich von M sei mit $\mathcal{D}(\phi_M)$ bezeichnet.

Definition 4.7 (Probabilistisch berechenbar) *Eine Funktion*

$$f : \Sigma^* \to \Sigma^*$$

ist probabilistisch berechenbar ($f \in$ PBF), wenn es eine PTM M mit $f \equiv \phi_M$ gibt.

Man kann zeigen, daß jede probabilistisch berechenbare Funktion f berechenbar ist, d.h. f ist auch mit einer deterministischen Turing Maschine berechenbar. Anders ausgedrückt: Die

probabilistisch berechenbaren Funktionen sind genau die **partiell rekursiven** Funktionen.

Definition 4.8 (Fehlerwahrscheinlichkeit) *Sei M eine PTM. Die* **Fehlerwahrscheinlichkeit** *(engl. error probability) von M ist eine Funktion*

$$e_M(x) = \begin{cases} \Pr[M(x) \neq \phi_M(x)] & \textit{wenn } \phi_M(x) \textit{ definiert} \\ \textit{undefiniert} & \textit{sonst.} \end{cases}$$

M berechnet ϕ_M mit beschränkter Fehlerwahrscheinlichkeit (engl. bounded error probability), wenn es ein $\epsilon > 0$ gibt, so daß

$$\forall x \in \mathcal{D}(\phi_M): \; e_M(x) \leq \frac{1}{2} - \epsilon.$$

Definition 4.9 (Monte-Carlo PTM) *Eine PTM M, die ϕ_M mit beschränkter Fehlerwahrscheinlichkeit berechnet, nennt man* **randomisierte Turing Maschine** *(RTM) oder* **Monte-Carlo TM**.

Lemma 4.1 (δ-Lemma) *Zu jeder Sprache A, die von einer RTM M erkannt werden kann, gibt es für jedes $\delta > 0$ eine RTM M_δ, die A mit Fehlerwahrscheinlichkeit $e_{M_\delta} \leq \delta$ erkennt. Dabei erhöht sich die Anzahl der Schritte nur um einen Faktor, der logarithmisch von δ abhängt.*

Beweis: Die RTM M_δ wiederholt den Lauf der RTM M mehrmals ($2k$-mal). Dabei gibt sie das Ergebnis aus, das am häufigsten erzielt wurde. Bei Parität wählen wir irgendein Ergebnis aus. Wir wollen nun die Fehlerwahrscheinlichkeit hierfür abschätzen.

Also gilt $\Pr[M$ macht einen Fehler im i-ten Lauf$] = p \leq 1/4$ und somit

$$\Pr[\text{ Fehler tritt öfters als } k\text{-mal auf}] = \sum_{i=k}^{2k} \binom{2k}{i} p^i (1-p)^{2k-i} \tag{4.1}$$

Wir wollen nun zeigen, daß $p^i(1-p)^{2k-i} \leq p^k(1-p)^k$ im Intervall $[k:2k]$ ist. Dafür differenzieren wir $p^i(1-p)^{2k-i}$ nach i und erhalten

$$p^i(1-p)^{2k-i}(\ln p - \ln(1-p))$$

Dieser Wert ist kleiner als Null, für $p \leq 1/4$ im Intervall $[k:2k]$. Also ist $p^i(1-p)^{2k-i} \leq p^k(1-p)^k$. Dann ergibt sich mit Gleichung 4.1, daß

$$\Pr[\text{ Fehler tritt öfters als } k\text{-mal auf}] \leq 2^{-2k}2^{2k-1}(3/4)^k = 1/2^{O(k)}$$

Daher kann die Wahrscheinlichkeit durch Iterieren beliebig verbessert werden. ■

Mit Hilfe dieses Lemmas können wir die **Monte-Carlo TM** auch dadurch charakterisieren, daß sie eine **Fehlerwahrscheinlichkeit** $e_M(x) < 1/4$ besitzen.

Auf der anderen Seite reicht es auch aus, fehlerbeschränkte Turing-Maschinen dadurch zu kennzeichnen, daß die Fehlerwahrscheinlichkeit kleiner ist als $1/2 - 1/p(n)$ für ein festes Polynom $p(n)$. Durch eine polynomielle Anzahl von Iterationen kann auch hier die Fehlerwahrscheinlichkeit beliebig klein gehalten werden.

Definition 4.10

Eine Monte-Carlo PTM M heißt starke (strenge) Monte-Carlo PTM (R_STM), wenn folgendes gilt:

1. $e_M(x) = 0$ *für* $x \notin L(M)$,
2. $e_M(x) < 1/4$ *für* $x \in L(M)$.

Bemerkung 4.2 *Für starke Monte-Carlo PTM's reicht es aus, die Fehlerwahrscheinlichkeit für $x \in L(M)$ durch ein beliebiges $\epsilon < 1$ zu beschränken. Durch mehrfache Iteration wird dann eine beliebig geringe Fehlerwahrscheinlichkeit erzielt.*

Damit haben wir geklärt, was eine Berechnung einer probabilistischen Turing Maschine ist. Nun wollen wir wieder Komplexitätsklassen einführen. Hierzu muß jedoch erst die Laufzeit und der Speicherverbrauch einer probabilistischen Turing Maschine definiert werden. Für die Laufzeitdefinition haben wir zwei Möglichkeiten.

Definition 4.11 (Laufzeit einer PTM ([G77])) *Die* **probabilistische Laufzeit** T_M *einer PTM M ist definiert durch:*

$$T_M(x) = \begin{cases} \min\{n \mid \Pr[M(x) = \phi_M(x) \textit{ in } n \textit{ Schritten}] > 1/2\} & \textit{falls } \phi_M(x) \textit{ definiert} \\ \infty & \textit{falls } \phi_M(x) \textit{ undefiniert.} \end{cases}$$

Definition 4.12 (Mittellaufzeit einer PTM)
Die **probabilistische Mittellaufzeit** *(engl. average run time)* $\bar{T}_M$ *einer PTM M ist der* **Erwartungswert** *der Schrittanzahl:*

$$\bar{T}_M(x) = \sum_{n=1}^{\infty} n \cdot \Pr[\textit{Berechnung über } x \textit{ benötigt genau } n \textit{ Schritte}].$$

Die Mittellaufzeit ist eine ungeeignete Definition, wenn wir die Klasse der PTM's betrachten. Das anormale Verhalten wird im Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther) [K91] sichtbar.

Definition 4.13 (Blum'sches Komplexitätsmaß ([B67]))
Eine Funktion T_M ist ein Komplexitätsmaß, falls T_M die folgenden beiden Axiome erfüllt sind.

- *$T_M(x)$ ist genau dann definiert, wenn $M(x)$ definiert ist,*
- *Das Prädikat $T_M(x) = n$ ist entscheidbar.*

Die probabilistische Laufzeit T_M ist ein Komplexitätsmaß für die ganze Klasse der PTM's. Die Mittellaufzeit ist jedoch <u>kein</u> Komplexitätsmaß für diese Klasse (Siehe Vorlesungsskript

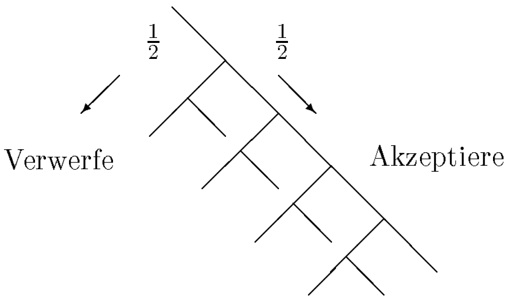

Abbildung 4.3: Berechnungsbaum

Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen) von M. Karpinski (ausgearbeitet von K. Werther) [K91]).

Der nachfolgende Satz zeigt, daß für die Klasse der randomisierten Turing Maschinen, d.h. derjenigen probabilistischen Turing Maschinen, die eine beschränkte Fehlerwahrscheinlichkeit besitzen, die beiden Laufzeitdefinitionen jedoch äquivalent sind. Daher ist die Mittellaufzeit in diesem Fall auch ein Komplexitätsmaß.

Satz 4.1 *Sei M eine Monte-Carlo TM (RTM). Dann gibt es ein c, so daß*

$$T_M(x) \leq c \cdot \bar{T}_M(x) \quad \textit{falls } \phi_M \textit{ definiert ist.}$$

Beweis: Da M eine Monte-Carlo TM ist, hat M eine mit $\epsilon < 1/2$ beschränkte Fehlerwahrscheinlichkeit. Sei $c = 1/(1/2 - \epsilon)$. Wenn $\bar{T}_M(x) = \infty$ ist, gibt es nichts zu beweisen. Sei also $\bar{T}_M(x) < \infty$. Dann gilt nach der Tschebyscheff'schen Ungleichung 1.Art

$$\Pr[\text{Zeit von } M(x) > c\bar{T}_M(x)] < \bar{T}_M(x)/c\bar{T}_M(x) = 1/c = 1/2 - \epsilon.$$

Diese Gleichung ist äquivalent zu

$$\Pr[\text{Zeit von } M(x) \leq c\bar{T}_M(x)] > 1/2 + \epsilon.$$

Da die Fehlerwahrscheinlichkeit von M beschränkt ist, gilt

$$\Pr[M(x) \neq \phi_M(x) \text{ in Zeit } c\bar{T}_M(x)] \leq \Pr[M(x) \neq \phi_M(x)] < \epsilon.$$

Somit folgt

$$\Pr[M(x) = \phi_M(x) \text{ in Zeit } c\bar{T}_M(x)] > 1/2.$$

Also ist

$$T_M(x) \leq c\bar{T}_M(x).$$

■

Nun wollen wir mit Hilfe der neuen Maschinenmodelle auch neue Komplexitätsklassen einführen.

Definition 4.14 (Probabilistische Komplexitätsklassen)

$$\mathit{PrTIME}(T(n)) = \{A | \exists \mathit{PTM}\ M\ \mathit{mit}\ L(M) = A\ \mathit{und}\ T_M(n) = O(T(n))\}$$

$$\mathit{RTIME}(T(n)) = \{A | \exists \mathit{RTM}\ M\ \mathit{mit}\ L(M) = A\ \mathit{und}\ T_M(n) = O(T(n))\}$$

$$R_S\mathit{TIME}(T(n)) = \{A | \exists R_S\mathit{TM}\ M\ \mathit{mit}\ L(M) = A\ \mathit{und}\ T_M(n) = O(T(n))\}$$

$$\mathit{PP} = \{A | \exists \mathit{PTM}\ M\ \mathit{mit}\ L(M) = A\ \mathit{und}\ T_M(n) = n^{O(1)})\}$$

$$\mathit{RP} = \{A | \exists \mathit{RTM}\ M\ \mathit{mit}\ L(M) = A\ \mathit{und}\ T_M(n) = n^{O(1)})\}$$

$$R_S P = \{A | \exists R_S\mathit{TM}\ M\ \mathit{mit}\ L(M) = A\ \mathit{und}\ T_M(n) = n^{O(1)})\}$$

Eine weitere Klasse ist Δ^P. Man nennt sie Las-Vegas Klasse. Eine PTM, die einen Las-Vegas Algorithmus darstellt, hat als Ausgabe 0, 1 und ?. Das Fragezeichen bedeutet, daß die PTM zu keinem korrekten Ergebnis gekommen ist. Die Wahrscheinlichkeit für ? ist beschränkt.

Für diese Komplexitätsklassen gelten nun die folgenden Inklusionen.

Satz 4.2 *Es gilt*

$$P \subseteq \Delta^P \begin{array}{c} \subseteq RP \subseteq \\ \subseteq NP \subseteq \end{array} PP.$$

Beweis: Siehe Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther) [K91] .

■

4.2 Randomisierte Schaltkreise

Wir betrachten jetzt *randomisierte* Schaltkreise. Diese Schaltkreise sind Schaltkreise mit den weiteren folgenden Eigenschaften.

Definition 4.15 (Probabilistische Schaltkreise) *Ein probabilistischer Schaltkreis $\mathcal{C}$ ist die Realisierung einer booleschen Funktion*

$$f' : \mathbb{B}^n \times \mathbb{B}^{\rho(n)} \to \mathbb{B}$$

$$f' : (x, \rho) \mapsto f'(x, \rho).$$

Diese Funktion induziert eine Funktion $f : \mathbb{B}^n \to \mathbb{B}$ mit

$$f(x) = \begin{cases} 1 & \Pr[f'(x,\rho) = 1] > 1/2 \\ 0 & \Pr[f'(x,\rho) = 0] > 1/2 \\ \textit{undefiniert} & \textit{sonst.} \end{cases}$$

Ein solcher Schaltkreis ist in Abbildung 4.4 dargestellt.

Definition 4.16 (Randomisierter Schaltkreis) *Ein probabilistischer Schaltkreis $\mathcal{C}$ ist ein* **randomisierter Schaltkreis**, *falls eine Funktion g existiert mit*

$$\Pr[f'(x, \rho) = g(x)] > 3/4.$$

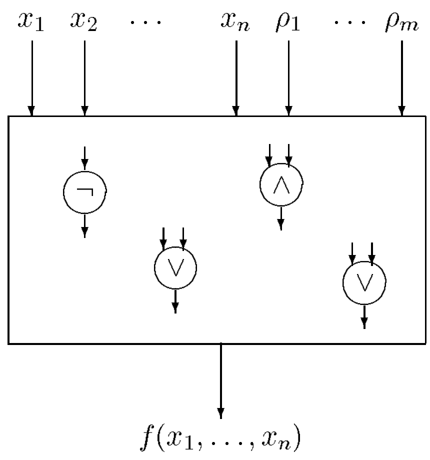

Abbildung 4.4: Probabilistischer Schaltkreis

Mit Hilfe von Lemma 4.1 reicht es auch aus

$$\Pr[f'(x, \rho) = g(x)] > 1/2 + 1/p(n) \qquad (*)$$

zu fordern. Ein Schaltkreis mit Fehlerwahrscheinlichkeit $< 1/4$ *besteht dann aus einer polynomiellen Anzahl von Schaltkreisen, die* $(*)$ *erfüllen und parallel arbeiten. Die Ausgabe ist dann ein Mehrheitsentscheid wie nach Lemma 4.1.*

Nun interessieren uns jedoch nur uniforme Schaltkreise, d.h. Schaltkreise, die log-space Turing berechenbar sind. Dies definiert die RUC Schaltkreisfamilien.

Definition 4.17 (Randomisierte uniforme Schaltkreise)
Ein randomisierte Schaltkreisfamilie $\{\overline{\mathcal{C}}_n\}$ *ist uniform, falls die Compilerabbildung* $1^n \to \mathcal{C}_n$ *log-space Turing berechenbar ist, wobei* n *die Anzahl der Eingänge ohne die Eingänge der Zufallsbits ist. Eine solche Schaltkreisfamilie nennen wir dann RUC-Algorithmus.*

Dadurch können wir analog zu den NC-Klassen auch hier Komplexitätsklassen einführen.

Definition 4.18 *Sei $f : I\!N \rightarrow I\!N$. Damit sei*

$$\begin{aligned} \textit{U-RDEPTH}(f(n)) = \quad & \{A \mid \exists RUC\text{-}\textit{Algorithmus}\ \{\mathcal{C}_n\}\ \textit{mit} \\ & L(\{\mathcal{C}_n\}) = A\ \textit{und}\ \textit{Depth}(\mathcal{C}_n) = O(f(n))\} \end{aligned}$$

$$\begin{aligned} \textit{U-RSIZE}(f(n)) = \quad & \{A \mid \exists RUC\text{-}\textit{Algorithmus}\ \{\mathcal{C}_n\}\ \textit{mit} \\ & L(\{\mathcal{C}_n\}) = A\ \textit{und}\ \textit{Size}(\mathcal{C}_n) = O(f(n))\} \end{aligned}$$

$$\begin{aligned} RNC^k = \quad & \{A \mid \exists RUC\text{-}\textit{Algorithmus}\ \{\mathcal{C}_n\}\ \textit{mit}\ L(\{\mathcal{C}_n\}) = A \\ & \textit{und}\ \textit{Size}(\mathcal{C}_n) = n^{O(1)}\ \textit{und}\ \textit{Depth}(\mathcal{C}_n) = O(\log^k n)\} \end{aligned}$$

$$RNC = \bigcup_{k \in I\!N} RNC^k.$$

Man beachte, daß bei randomisierten Schaltkreisen die Zufallsbits während der Berechnung öfters benutzt werden können. Bei randomisierten Turing Maschinen ist dies nur möglich, falls die benutzten Zufallbits auf dem Speicherband zwischengespeichert werden. Das Berechnungsmodell der randomisierten Schaltkreise kann daher eher mit randomisierten Turing Maschinen verglichen werden, die eine Kopfbewegung über das Randomband in beide Richtungen erlauben. Solche Maschinen nennt man Two-Way Random Tape Turing Maschinen. Für diese Maschinen gilt der folgende überraschende Satz.

Satz 4.3 ([KV85]) *Eine Two-Way Random Tape Turing Maschine sei mit $R^{(2)}TM$ bezeichnet. Damit gilt*

$$R^{(2)}\textit{SPACE}(\log n) = \textit{PSPACE}.$$

4.3 Probabilistische Testalgorithmen

In diesem Kapitel werden wir einige Testalgorithmen vorstellen, mit deren Hilfe algebraische Gleichheiten überprüft werden können. Diese Algorithmen sind probabilistisch und haben den gleichen einfachen Aufbau: Falls eine algebraische Gleichung nicht allgemeingültig erfüllt ist, so gibt es nur *relativ wenige* Belegungen der Variablen, die diese Gleichung zufällig erfüllen. Ist diese algebraische Gleichung z.B. eine Polynomgleichung, so sind diese ungünstigen Belegungen der Variablen gerade die Nullstellen des Polynoms.

Unser erster Algorithmus testet, ob ein gegebenes **multivariates Polynom** identisch dem Nullpolynom ist. Dabei ist das Polynom natürlich nicht bekannt, sondern durch eine Black-Box gegeben, d.h. es sind Auswertungen an beliebigen Stellen möglich.

Algorithmus 1 Nulltest für Polynome

Eingabe: Eine Black-Box für ein Polynom $P(x_1, \ldots, x_n)$ über $\mathbb{Z}$ und eine Schranke D für den Grad des Polynoms.

Schritt 1: Konstruiere die Menge $E = \{1, 2, \ldots, 4D\}$

Schritt 2: Wähle zufälliges $\bar{x} = (x_1, \ldots, x_n) \in E^n$

Ausgabe: $\begin{cases} P \equiv 0 & \text{falls } P(\bar{x}) = 0 \\ P \not\equiv 0 & \text{sonst} \end{cases}$

Satz 4.4 (Schwartzsches Lemma [S80a]) *Sei $P = P_1(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ und $P \not\equiv 0$. Sei d_1 der Grad von x_1 in P_1 und P_2 das Koeffizientenpolynom von $x_1^{d_1}$:*

$$P_1(x_1, \ldots, x_n) = x_1^{d_1} \cdot P_2(x_2, \ldots, x_n) + q_1(x_1, \ldots, x_n).$$

Die P_i und d_i für $i > 1$ seien entsprechend induktiv definiert. Dann hat P höchstens

$$|I_1 \times \cdots \times I_n| \left(\frac{d_1}{|I_1|} + \cdots + \frac{d_n}{|I_n|} \right)$$

Nullstellen in $I_1 \times \cdots \times I_n$.

Beweis: Wir führen den Beweis über Induktion nach der Anzahl der Variablen.

- Für den Fall eines univariaten Polynoms P gilt, daß die Anzahl der Nullstellen beschränkt ist durch
$$\deg(P) = d_1 = |I_1| \frac{d_1}{|I_1|}.$$
- Induktionsschritt:
P_1 hat die Darstellung
$$P_1(x_1, \ldots, x_n) = x_1^{d_1} \cdot P_2(x_2, \ldots, x_n) + q_1(x_1, \ldots, x_n).$$
Nun gibt es zwei Möglichkeiten:
    1. $(z_2, \ldots, z_n)$ ist Nullstelle von P_2. Dann kann eventuell $P(x_1, z_2, \ldots, z_n) = 0$ für alle $x_1 \in I_1$ gelten. Da die Anzahl der Nullstellen von P_2 nach Induktionsannahme durch
    $$|I_2 \times \cdots \times I_n| \left(\frac{d_2}{|I_2|} + \ldots + \frac{d_n}{|I_n|} \right)$$
    beschränkt ist, kann die Anzahl der Nullstellen von P für diesen Fall durch
    $$|I_1| \cdot |I_2 \times \cdots \times I_n| \left(\frac{d_2}{|I_2|} + \ldots + \frac{d_n}{|I_n|} \right)$$
    abgeschätzt werden.
    2. $(z_2, \ldots, z_n)$ ist keine Nullstelle von P_2. Dann ist $P(x_1, z_2, \ldots, z_n)$ ein nicht verschwindendes Polynom in x_1 vom Grade d_1. Damit ist eine Schranke für die Anzahl der Nullstellen von P in diesem Fall
    $$d_1 \cdot |I_2 \times \cdots \times I_n|.$$

    Damit kann die Gesamtanzahl von Nullstellen von P durch
$$\begin{aligned} &|I_1| \cdot |I_2 \times \cdots \times I_n| \left(\frac{d_2}{|I_2|} + \ldots + \frac{d_n}{|I_n|} \right) + d_1 |I_2 \times \cdots \times I_n| \\ &= |I_1 \times \cdots \times I_n| \left(\frac{d_1}{|I_1|} + \ldots + \frac{d_n}{|I_n|} \right) \end{aligned}$$
    beschränkt werden.

■

Korollar 4.1 *Sei* $P = P(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$, $P \not\equiv 0$ *und* $I = I_1 = \cdots = I_n$ *mit* $|I| \geq c \cdot deg(P)$. *Dann ist die Anzahl der Nullstellen in* I^n *durch* $|I|^n/c$ *beschränkt.*

Beweis: Wir wenden den Satz 4.4 an. In diesem Fall gilt

$$\begin{aligned} |I|^n \cdot \textstyle\sum_{j=1}^n d_j/|I| &= |I|^{n-1} \cdot (\textstyle\sum_{j=1}^n d_j) \\ &\leq |I|^{n-1} \cdot \deg(P) \leq |I|^n/c \end{aligned}$$

und somit ist die Anzahl der Nullstellen durch $|I|^n/c$ beschränkt. ∎

Satz 4.5 (Korrektheit von Algorithmus 1) *Der Algorithmus 1 ist ein korrekter starker Monte-Carlo Algorithmus.*

Beweis: Mit Hilfe von Lemma 4.1 gilt, daß die Anzahl der Nullstellen des Polynoms P durch $|E|^n/4$ beschränkt ist. Für einen zufällig gewählten Vektor $\bar{x}$ aus E^n und $P \not\equiv 0$ gilt damit

$$\Pr[P(\bar{x}) = 0] \leq \frac{|E|^n/4}{|E^n|} = 1/4.$$

Damit ist die Ausgabe $P \equiv 0$ mit Wahrscheinlichkeit $\geq 3/4$ korrekt. Die Ausgabe $P \not\equiv 0$ ist immer korrekt, da für $P \equiv 0$ kein $\bar{x}$ mit $P(\bar{x}) \neq 0$ existiert. ∎

Als nächstes wenden wir uns dem Problem der **Matrizenmultiplikation** zu. Dabei ist zu entscheiden, ob das Produkt zweier Matrizen eine dritte gegebene Matrix ergibt. Ein möglicher deterministischer Algorithmus zur Lösung dieses Problems besteht darin, die Matrizen A und B miteinander zu multiplizieren und das Ergebnis mit C zu vergleichen. Dies liefert auf naive Weise einen Algorithmus, der in TIME(n^3) und in $NC^1(n^3)$ liegt (vgl. [K91]). Mit Hilfe der neuesten Algorithmen im Bereich der Matrizenmultiplikation — aufbauend auf den Ideen von Strassen — ist dies sogar in TIME($n^{2.376}$) und $NC^1(n^{2.376})$ möglich. Eine untere Schranken für einen solchen Algorithmus wäre in jedem Fall $\Omega(n^2)$.
Der nachfolgende Algorithmus ist ein optimaler R_STM Algorithmus zur Lösung dieses Problems. Er liegt in R_STIME(n^2) bzw. in $RNC^1(n^2)$ (vgl. [K91]).

Algorithmus 2 Test für Matrizenmultiplikation

Eingabe: Reellwertige $n \times n$ Matrizen A,B und C.

Schritt 1: Erzeuge zufällig 2 Vektoren $X_1, X_2 \in \{-1,1\}^n$:

$$X_1 = (x_{11}, \ldots, x_{1n}), \quad X_2 = (x_{21}, \ldots, x_{2n}).$$

Schritt 2: Berechne: $A(BX_1), A(BX_2), CX_1$ und CX_2.

Ausgabe: $\begin{cases} 1 & \text{falls } ABX_1 = CX_1 \text{ und } ABX_2 = CX_2 \\ 0 & \text{sonst} \end{cases}$

Satz 4.6 *Der Algorithmus 2 arbeitet korrekt und liegt in R_STIME(n^2) bzw. in $RNC^1(n^2)$.*

Beweis: Sei $D = (d_{ij}) = A \cdot B$ und $C = (c_{ij})$. Sei nun $\bar{x} = (x_1, \ldots, x_n)$ mit $x_i \in \{-1, 1\}$. Falls nun für $\bar{x}$

$$(A \cdot B - C) \cdot \bar{x} = 0$$

gilt, ist dies äquivalent zu

$$\forall i \quad (d_{i1} - c_{i1}, \ldots, d_{in} - c_{in}) \perp (x_1, \ldots, x_n). \tag{4.2}$$

Falls $A \cdot B \neq C$, könnten wir bei unserer zufälligen Wahl der Vektoren X_1 und X_2 schlechte Vektoren bekommen haben, für die trotzdem die Orthogonalitätsbeziehung (4.2) gilt. Es können nach Lemma 4.2 jedoch höchstens 2^{n-1} der 2^n vielen $\{-1,1\}$ Vektoren orthogonal zu den n Vektoren $(d_{i1} - c_{i1}, \ldots, d_{in} - c_{in})$ sein, da wenigstens einer dieser Vektoren ungleich dem Nullvektor ist. Somit ist die Wahrscheinlichkeit, zweimal einen schlechten Vektor zu wählen, kleiner als $1/4$.
Der Algorithmus ist ein starker Monte-Carlo Algorithmus, da die Ausgabe $\not\equiv$ mit Fehlerwahrscheinlichkeit 0 erfolgt. Der Algorithmus arbeitet auch innerhalb der geforderten Zeit- und Prozessorschranken, da nur Produkte zwischen Matrizen und Vektoren berechnet werden.

■

Lemma 4.2 *Sei $v \neq (0, \ldots 0)$. Dann sind höchstens 2^{n-1} viele $\{-1,1\}$ Vektoren orthogonal zu v.*

Beweis: Da $v = (v_1, \ldots, v_n)$ ungleich dem Nullvektor ist, ist wenigstens ein $v_{i_0} \neq 0$. Sei i_0 o.B.d.A gleich n, und seien $x = (x_1, \ldots, x_{n-1}, -1)$ und $y = (x_1, \ldots, x_{n-1}, 1)$. Somit gilt

$$< x, v >= \sum_{i=1}^{n-1} x_i \cdot v_i - v_n \neq < x, v > +2v_n =< y, v > .$$

Daher können nicht sowohl $< x, v >$ als auch $< y, v >$ gleich 0 sein. Die Zuordnung $x \mapsto y$ ordnet also einem zu v orthogonalen Vektor eineindeutig einen nicht orthogonalen Vektor zu. Somit können höchstens die Hälfte der 2^n Vektoren orthogonal zu v sein. ■

Kapitel 5

Die Anwendung von Testalgorithmen zur Lösung von Optimierungsproblemen

Die in Kapitel 3.1 eingeführten Testalgorithmen für Polynomidentitäten kann man leicht zur Lösung von kombinatorischen Optimierungsproblemen anwenden.

5.1 Matchingprobleme

Wir werde in diesem Abschnitt die Klasse der Matchingalgorithmen näher betrachten.

Definition 5.1 *Matching*
Gegeben sei ein Graph $G = (V, E)$. Eine Teilmenge $M \subseteq E$ heißt

- *Matching, falls keine zwei Kanten in M inzident sind;*
- *nichterweiterbares oder auch maximales Matching, falls es kein Matching gibt, das M echt enthält;*

- *Maximum Matching, falls M ein Matching maximaler Kardinalität ist;*
- *Perfektes Matching, falls jeder Knoten aus V mit genau einer Kante aus M inzident ist.*
- *MIN-W-KPM, falls wir zusätzlich eine Gewichtsmatrix $W = (w_{i,j})$ mit $w_{i,j} \in n^{O(1)}$ haben und ein perfektes Matching in G suchen, so daß die Summe der Gewichte der Kanten minimiert wird. Die Maximierungsversion ist analog definiert und wird MAX-W-KPM genannt.*

Definition 5.2 (Determinante, Permanente) *Gegeben sei eine Matrix $A = (a_{i,j})_{1 \leq i,j \leq n}$. Dann sind die* **Determinante** *und* **Permanente** *von A wie folgt definiert:*

$$det(A) = \sum_{\sigma \in S_n} sign(\sigma) \prod_{i=1}^{n} a_{i,\sigma(i)},$$

$$perm(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} a_{i,\sigma(i)}.$$

Falls $A \in \mathbb{B}^{n \times n}$ kann auch die **logische Permanente** *von A definiert werden:*

$$perm^L(A) = \bigvee_{\sigma \in S_n} \bigwedge_{i=1}^{n} a_{i,\sigma(i)}.$$

Nun können wir die verschiedenen Probleme, die mit dem Perfekt Matching Problem in Zusammenhang stehen, aufzählen. Sei $G = (V, E)$ ein Graph.

1. **Entscheidungsproblem**: Besitzt G ein Perfektes Matching ? (EPM)
2. **Konstruktionsproblem**: Konstruiere ein (beliebiges) Perfektes Matching von $G(KPM)$.

5.1.1 *PERFECT-MATCHING*

Das einfachste dieser Probleme ist das Entscheidungsproblem. Um dieses Problem zu lösen hilft der folgende Satz von Tutte. Zunächst jedoch eine

Definition 5.3 (Tutte'sche Matrix) *Sei $G = (V, E)$ ein Graph mit Adjazenzmatrix A. Zu G sei nun die folgende schiefsymmetrische Matrix (Tutte'sche Matrix) C in den Variablen x_{ij} definiert:*

$$c_{ij} = \begin{cases} x_{ij} & i < j \text{ und } a_{ij} = 1 \\ -x_{ji} & i > j \text{ und } a_{ij} = 1 \\ 0 & i = j \text{ oder } a_{ij} = 0. \end{cases}$$

Man beachte, daß $det(C)$ ein Polynom in den Variablen x_{ij} ist.

Satz 5.1 ((Tutte, 1952)) *Ein ungerichteter Graph $G = (V, E)$ mit Adjazenzmatrix A besitzt genau dann ein Perfektes Matching, wenn $det(C) \not\equiv 0$ ist, d.h. $det(C)$ enthält mindestens ein Monom.*

Beweis: Falls G ein Perfektes Matching besitzt, so induziert die zugehörige Permutation ein Monom in der Determinante der Tutte'schen Matrix, welches sich nicht wegheben kann, da die Permutation aus 2-Zyklen besteht. Somit ist die Determinante nicht identisch 0 und die Tutte'sche Matrix regulär.

Welche weiteren Permutationen können nun Monome in der Tutte'schen Matrix induzieren? Eine Permutation, die einen ungeraden Zyklus enthält, wird durch die Permutation, die diesen Zyklus in entgegengesetzter Richtung durchläuft und sonst gleich ist, wieder gelöscht. Damit können keine Permutationen mit ungeraden Zyklen Monome erzeugen. Falls es Permutationen gibt, die nur gerade Zyklen enthalten, so gibt es auch Permutationen, die nur 2-Zyklen enthalten. Wenn die Tutte'sche Matrix also regulär ist, so gibt es in der Determinante der Tutte'schen Matrix ein Monom, das von einem Perfekten Matching induziert ist, und damit auch ein Perfektes Matching. ∎

Mit Hilfe dieses Satzes können wir das Entscheidungsproblem auf den Nulltest für ein multivariates Polynom zurückführen.

Definition 5.4 *Das Interpolationsproblem, d.h. der Test, ob ein als Black-Box gegebenes n-variates Polynom $P(x_1, \ldots, x_n)$ mit deg$(P) \leq m$ identisch 0 ist, sei mit IP(n, m) bezeichnet.*

Damit gilt mit Algorithmus 1:

$$\mathrm{IP}(n,m) \in \mathrm{R}_S\mathrm{TIME}((n+m)^{O(1)})$$

$$\mathrm{IP}(n,m) \in \mathrm{R}_S\mathrm{SPACE}(\log(n+m))$$

Wir können Determinanten leicht in kubischer Zeit berechnen. Daraus ergibt sich, daß das Entscheidungsproblem in RP liegt.

Wir werden eine Lösung des Perfekt Matching Problems für beliebige Graphen entwickeln, die in RP liegt. Diese Arbeit ist von Mulmeley, U.Vazirani und V.Vazirani (siehe [MVV87]).

Das **Entscheidungsproblem** haben wir bereits gelöst, indem wir den randomisierten Nulltest für multivariate Polynome aus Algorithmus 1 angewandt auf die Determinante der Tutte'schen Matrix verwenden. Mittels Lemma 5.1 gilt ja, daß die Determinante der Tutte'schen Matrix genau dann identisch 0 ist, falls G kein Perfektes Matching besitzt.

Nun wollen wir das **Konstruktionsproblem** betrachten. Wir wollen das Isolierungslemma (Lemma 3.2) nun auf unser Perfekt Matching Problem anwenden. Es ergibt sich also folgendes

Korollar 5.1 *Sei $G = (V, E)$ ein Graph, $|V| = n$. Wir definieren ein Gewichtssystem $GS = (E, PM, w)$ mit $PM = \{M \subseteq E | M$ ist Perfektes Matching in $G\}$ und w als randomisierte Funktion $w : E \to [1, \ldots, 2|E|]$. Dann ergibt sich aus Lemma 3.1*

$$\Pr[\min(E, PM, w) \textit{ ist eindeutig}] \geq 1/2.$$

Mit Hilfe der randomisierten Funktion

$$w : E \to [1, \ldots, 2|E|]$$

aus Korollar 5.1 können wir nun eine gewichtete Adjazenzmatrix $A = (a_{i,j})$ wie folgt konstruieren.

Definition 5.5 *Seien w_{ij} gleichverteilte Zufallsvariablen in $[1, \ldots, 2|E|]$. Dann sei die randomisierte Adjazenzmatrix A durch Einsetzen der $2^{w_{ij}}$ in die Tutte'sche Matrix definiert:*

$$a_{i,j} = \begin{cases} 2^{w_{ij}} & \text{falls } i < j \text{ und } (i,j) \in E \\ -2^{w_{ji}} & \text{falls } i > j \text{ und } (i,j) \in E \\ 0 & \text{falls } i = j \text{ oder } (i,j) \notin E. \end{cases}$$

Mit Hilfe des Korollar 5.1 können wir nun Aussagen über diese Matrix machen.

Lemma 5.1 *Gegeben sei ein Graph $G = (V, E)$ mit einer Gewichtsfunktion für die Kanten. Diese Gewichtsfunktion w sei so gewählt, daß das gewichtsminimale Perfekte Matching* eindeutig *ist. Falls wir eine geeignete randomisierte Gewichtsfunktion verwenden, ist die Wahrscheinlichkeit hierfür nach Korollar 5.1 $\geq 1/2$. Dann ist*

$$det(A) \neq 0$$

und für die größte Zweierpotenz 2^α, die $det(A)$ teilt, gilt

$$\alpha = 2 \cdot \min\{\bar{w}(M) | M \in PM\}.$$

Beweis: Wir erinnern uns an die Definition der Determinante einer Matrix:

$$\det(A) = \sum_{\pi \in S_n} \operatorname{sign}(\pi) \cdot \operatorname{value}(\pi) \quad \text{mit}$$

$$\operatorname{value}(\pi) = \prod_{i=1}^{n} a_{i,\pi(i)}.$$

Dabei tragen nur solche Permutationen π zum Wert der Determinante bei, für die $\operatorname{value}(\pi) \neq 0$, für die $\forall i \; (i, \pi(i)) \in E$ gilt. Wir werden also nur die Permutationen betrachten, die einen Wert zur Determinante beitragen. Ein solcher Wert ist dann immer eine Zweierpotenz, da das Produkt von Zweierpotenzen wieder eine Zweierpotenz ist. Mit $w_{\min}$ bezeichnen wir das minimale Gewicht eines Perfekten Matchings. Für eine Permutation π gibt es die folgenden Möglichkeiten.

- Die Permutation π induziert ein Perfektes Matching.
  In diesem Fall bildet $\pi : i \mapsto j, j \mapsto i$ ab, d.h. es entstehen nur 2-Zyklen. Bei der Berechnung von value(π) trägt dieser 2-Zyklus mit dem Wert $2^{w_{ij}}(-2^{w_{ij}}) = -2^{2w_{ij}}$ zu Buche. Somit ist der Wert dieser Permutation

  $$\text{value}\,(\pi) = (-1)^{n/2} 2^{w_\pi},$$

  wobei w_π die Summe der Gewichte der zu π gehörenden Kanten entspricht. Falls π das eindeutige gewichtsminimale Perfekte Matching darstellt, so ist $w_\pi = 2w_{\mathbf{min}}$, da die Kanten von π doppelt gezählt werden. Für alle anderen Permutationen σ ist $w_\sigma > w_\pi = 2w_{\mathbf{min}}$ und somit 2^{w_σ} immer ein **gerades** Vielfaches von $2^{2w_{\min}}$.

- Die Permutation π enthält keine ungeraden Zyklen.
  Wir können die geraden Zyklen in zwei Perfekte Matchings einteilen, indem wir in den Zyklen alternierende Kanten betrachten. Wir spalten also diese Permutation in zwei Perfekte Matchings M_1 und M_2 auf. Aufgrund der Eindeutigkeit eines minimalgewichteten Matchings, kann höchstens eines dieser beiden Perfekten Matchings minimalgewichtig sein. Damit ist

  $$2^{w_\pi} = 2^{w_{M_1} + w_{M_2}} > 2^{2w_{\min}},$$

  und somit auch ein **gerades** Vielfaches von $2^{2w_{\min}}$.

- Die Permutation enthält auch ungerade Zyklen.
  In diesem Fall betrachten wir neben π auch die Permutation σ, die einen der ungeraden Zyklen in umgekehrter Richtung durchläuft. Wir sehen leicht ein, daß value$(\pi) = -$value(σ) und sign$(\pi) =$ sign(σ) gilt, da σ aus π durch Anwenden einer geraden Anzahl von Transpositionen hervorgeht. Damit heben sich diese beiden Permutationen in der Determinante auf.

Insgesamt erhalten wir als Summanden in der Determinante nur gerade Vielfache von $2^{2w_{\min}}$ und einen Summand von der Form $(-1)^{n/2} 2^{2w_{\min}}$. Daher ist

$$\det\,(A) = (-1)^{n/2} 2^{2w_{\min}} + 2c2^{2w_{\min}}.$$

Also ist $\det(A) \neq 0$ und die höchste $\det(A)$ teilende Zweierpotenz ist 2^α für $\alpha = 2w_{\mathrm{min}}$. ■

Mit Hilfe dieses Lemmas können wir nun einen randomisierten Algorithmus angeben, der zu einem gegebenen Graphen ein Perfektes Matching konstruiert, falls ein solches existiert.

Eingabe: $G = (V, E)$

Schritt 1: Konstruiere die Matrix A wie oben angegeben.

Schritt 2: Berechne $\det(A)$ und $\mathrm{adj}(A) = (-1)^{i+j} \det A(i|j)_{i,j}$.

Schritt 3: Finde die größte Zahl w, so daß 2^{2w} $\det(A)$ teilt. Dies ist das Gewicht des gewichtsminimalen Perfekten Matchings.

Ausgabe: $M = \{(i,j) |\ \det(A(j|i)) \cdot 2^{w_{ij}}/2^{2w}$ ist ungerade$\}$.

Satz 5.2 *Die Menge M aus obigen Algorithmus ist das (minimale gewichtete) gesuchte Perfekte Matching von G.*

Beweis: Wir möchten feststellen, ob die Kante (i,j) zum gewichtsminimalen Perfekten Matching gehört. Dazu betrachten wir alle Permutationen, die diese Kante beinhalten, d.h. alle π mit $\pi(i) = j$. Diese Permutationen tragen

$$D_i^j := \sum_{\pi, \pi(i)=j} \mathrm{sign}(\pi) \mathrm{value}(\pi) = 2^{w_{ij}} \det(A(i|j))$$

zu $\det(A)$ bei. Offensichtlich gilt

$$\det(A) = \sum_i D_i^j = \sum_j D_i^j$$

Für jeden Knoten i ist nur eine Kante (i,j) im Perfekten Matching. Daher müssen alle Permutationen, die diese Kante nicht beinhalten, ein gerades Vielfaches von $2^{2w_{\mathrm{min}}}$ zu $\det(A)$ beitragen. Also ist auch D_i^k für $k \neq j$ ein gerades Vielfaches von $2^{2w_{\mathrm{min}}}$.
Betrachten wir nun D_i^j. Nur ein Summand dieser Summe ist ein ungerades Vielfaches von $2^{2w_{\mathrm{min}}}$, nämlich gerade die Permutation zum gewichtsminimalen Perfekten Matching. Alle

anderen Summanden sind wie oben gerade Vielfache von $2^{w_{\min}}$. Daher ist die Summe ein ungerades Vielfaches. Die Ausdrücke D_i^j sind leicht zu berechnen, da $\operatorname{adj}(A)$ gegeben ist.

■

Man beachte bei unserem Algorithmus jedoch den hohen Verbrauch an Zufallsinformation. Die Anzahl der benutzten Zufallsbits beträgt $n^2 \log n$.

5.1.2 *MAXIMUM-MATCHING*

Als nächstes wollen wir das Problem des Maximum Matchings lösen, indem wir dieses auf das Perfekt Matching Problem zurückführen.

Satz 5.3 *Maximum Matching* $\leq_P$ *KPM*

Beweis: Gegeben sei der Graph $G = (V, E)$. In einem Maximum Matching von G ist die Anzahl der nicht saturierten Knoten minimal. Daher konstruieren wir zu dem Graphen G die Graphenfamilie $G_k = (V_k, E_k)$ mit $G_k \supset G$ und suchen unter diesen Graphen den minimalen Graphen, der ein Perfektes Matching besitzt. Dieses Perfekte Matching in G_k korrespondiert dann zu einem Maximum Matching in G. Die Konstruktion der Graphen sieht nun wie folgt aus:

$$V_k = V \cup \{w_1, \ldots, w_k\}$$

$$E_k = E \cup \{(v, w_i) | v \in V,\ 1 \leq i \leq k\}.$$

Dabei konstruieren wir nur die Graphen G_k mit gerader Knotenmenge, da sonst kein Perfektes Matching möglich ist. Wenn G_k ein Perfektes Matching besitzt, so besitzen auch alle G_i mit $i > k$ ein Perfektes Matching. Unter allen diesen Graphen G_k, die ein Perfektes Matching besitzen (Entscheidungstest), suchen wir mit Hilfe der binären Suche den kleinsten Graphen . Zu diesem kleinsten Graphen G_l konstruieren wir nun ein Perfektes Matching. Dieses Perfekte Matching eingeschränkt auf G ist nun ein Maximum Matching, da die Anzahl der Kanten, die mit Knoten der Form w_i verbunden sind, in G_l kleinstmöglich war. ■

5.1.3 *WEIGHTED-PERFECT-MATCHING*

Wir wollen in diesem Abschnitt bezüglich polynomiellen Kantengewichten ein minimal (maximal) gewichtetes perfektes Matching in einem Graphen finden.

Für einen gegebenen Graphen $G = (V, E)$ haben wir eine Gewichtsfunktion (WF)

$$W : E \to \mathbb{N},$$

so daß die Binärdarstellung von $W_{i,j} = W(\{i, j\})$ polynomiell in $|V|$ ist.

Wir konstruieren ein Gewichtssystem $(E, \mathcal{P}, W)$ mit

$$\mathcal{P} = \{M | M \text{ ist ein perfektes Matching in } G\}.$$

Wir unterscheiden 2 Fälle:

Fall 1: $\min(E, \mathcal{P}, W)$ **ist eindeutig**

In diesem Fall können wir wie zuvor dieses eindeutige gewichtsminimale perfekte Matching konstruieren.

Fall 2: $\min(E, \mathcal{P}, W)$ **ist nicht eindeutig**

Wir pertubieren unsere Gewichtsfunktion wie folgt:

$$W_{i,j}^P \leftarrow aW_{i,j} + b$$

mit $a = 2|E|(n/2)$ $(n = |V|)$ und für alle $\{i, j\}$ unabhängig,

$$b = w_{i,j} \in_R \{1, ..., 2|E|\}.$$

Lemma 5.2 *Sei* $W(M_1) > W(M_2)$. *Dann gilt*

$$W^P(M_1) \geq W^P(M_2) + n/2.$$

Beweis: Sei $W(M_1) > W(M_2)$. Dann haben wir

$$W(M_1) \geq W(M_2) + 1.$$

Also gilt

$$\underbrace{2|E|(n/2)W(M_1)}_{=\alpha_1} \geq \underbrace{2|E|(n/2)W(M_2)}_{=\alpha_2} + 2|E|(n/2).$$

Wir haben

- $W^P(M_1) \geq \alpha_1 + n/2$ und
- $W^P(M_2) \leq \alpha_2 + 2|E|n/2$.

Durch Einsetzen erhalten wir das Lemma. ∎

Wir definieren nun für ein perfektes Matching $M \in \mathcal{P}$

$$\begin{aligned} D(M) &= \sum_{\{i,j\} \in M} 2|E|(n/2)W_{i,j} \\ R(M) &= \sum_{\{i,j\} \in M} w_{i,j} \end{aligned}$$

Aus dem Isolierungslemma (Lemma 3.2) folgt, daß (mit Wahrscheinlichkeit von mindestens 1/2)

$$|\{M | R(M) = \min\{R(N) | N \in \mathcal{P}\}\}| = 1.$$

Wir unterscheiden 2 Fälle:

$W(M_1) \neq W(M_2)$: Dann erhalten wir mit Lemma 5.2, daß $W^P(M_1) \neq W^P(M_2)$.

$W(M_1) = W(M_2)$: Dann ist $D(M_1) = D(M_2)$ und somit

$$W^P(M_1) = D(M_1) + R(M_1) \neq D(M_2) + R(M_2) = W(M_2).$$

Also ist $\min(E, \mathcal{P}, W^P)$ eindeutig, und wir können wie zuvor unseren KPM-Algorithmus anwenden.

Wir haben mit dieser Konstruktion folgendes Theorem bewiesen.

Satz 5.4 MIN-W-$KPM \in RP$.

Wir können analog auch MAX-W-KPM lösen, indem wir statt Lemma 3.2, die Maximierungsvariante des Isolationslemma (Lemma 3.1)einsetzten.

Satz 5.5 MAX-W-$KPM \in RP$.

5.2 $CHINESE$-$POSTMAN$

Im $CHINESE$-$POSTMAN$ ist es das Ziel, in einem Graph $G = (V, E)$ mit Startknoten $v_0 \in V$ ("Postoffice") eine Rundreise zu finden, die in v_0 startet und endet, jede Kante mindestens einmal durchläuft und möglichst wenige Kanten benutzt. Man kann auch eine gewichtete Version betrachten, in der alle Kanten Gewichte haben und das Gewicht der Rundreise minimiert werden soll. Es ist eine einfache Aufgabe den hier vorgestellten Algorithmus in diese Richtung zu erweitern.

Das $CHINESE$-$POSTMAN$ steht im engen Zusammenhang zum Eulerkreis Problem in Multigraphen.

Definition 5.6 *Ein Multigraph ist ein Graph in dem eine Kante mehrfach auftauchen darf (Multikanten).*

Definition 5.7 *Sei $G = (V, E)$ ein Multigraph. Ein Pfad P heißt Eulerpfad, wenn er jede Kante genau einmal durchläuft. P heißt Eulerkreis, wenn der Startknoten gleich dem Endknoten ist.*

Folgende grundlegenden Eigenschaften sind sehr einfach zu beweisen:

Lemma 5.3 *Sei ein ungerichter, zusammenhängender Multigraph $G = (V, E)$ gegeben.*

- *G hat genau dann einen Eulerkreis, wenn der Grad von jedem Knoten gerade ist.*
- *G besitzt genau dann einen Eulerkreis, wenn die Kantenmenge in Kreise zerlegt werden kann.*
- *G habe $2k$ Knoten von ungeradem Grad. Für genau $k \in \{0, 1\}$ hat G einen Eulerpfad.*

Aus diesem Lemma kann man leicht lineare Algorithmen zur Konstruktion von Eulerwegen und Eulerkreisen ableiten.

Das $CHINESE\text{-}POSTMAN$ ist also das Problem, den Graphen so preiswert wie möglich zu erweitern, so daß der resultierende Multigraph einen Eulerkreis enthält.

Eingabe: Ein Graph $G = (V, E)$.

Schritt 1: Es sei U die Menge der Knoten mit ungeraden Grad ($|U| = 2m$). Bestimme für alle $u, v \in U$ die Länge $W_{u,v}$ des kürzesten Pfades zwischen u und v.

Schritt 2: Sei G' der vollständige Graph auf U mit Gewichtung W. Bestimme $MIN\text{-}W\text{-}KPM$ M in G'.

Schritt 3: Addiere kürzesten Wege von u nach v, für alle $\{u, v\} \in M$, zu G.

Satz 5.6 $CHINESE\text{-}POSTMAN \in RP$

BEWEIS: Es genügt die Korrektheit des Algorithmus zu zeigen. M ist das gewichtsminimale perfekte Matching in G'. Also ist die Erweiterung, so daß alle Grade des resultierenden Graphen gerade sind, auch minimal. Mit obigen Lemma folgt die Behauptung. ■

5.3 $CYCLE\text{-}COVER$

Wir wollen in einem gerichteten Graphen $G = (V, E)$ ($E \subseteq V \times V$) eine Familie von gerichteten Kreisen finden, so daß jeder Knoten von genau einem Kreis dieser Familie überdeckt wird und die Summe der Gewichte minimiert wird.

Für $e \in E$ mit $e = (v_1, v_2)$ definieren wir $\pi_1(e) := v_1$ und $\pi_2(e) := v_2$. Ein *Zyklus (oder Kreis)* in G ist eine Folge von Kanten $(e_1, ..., e_k)$, so daß

- $e_i \neq e_j$ für $i \neq j$,
- $\pi_2(e_1) = \pi_1(e_2)$,
- $\pi_2(e_i) = \pi_1(e_{i+1})$ für $1 \leq i < k$,
- $\pi_2(e_k) = \pi_1(e_1)$.

Die Gesamtmenge von Zyklen in G wird mit $C(G)$ bezeichnet.

Sei nun $G = (V, \vec{E})$ $(\vec{E} \subseteq V \times V)$ und <u>WF</u> $W : \vec{E} \to I\!N$ gegeben. Für $C \in C(G)$ definieren wir

$$W(C) = \sum_{e_i \in C} W(e_i)$$

Wir nennen zwei Kreise C_1, C_2 disjunkt $(C_1 \cap C_2 = \emptyset)$, wenn sie keinen gemeinsamen Knoten haben.

Wir können nun $CYCLE$-$COVER$ formal definieren. Eingabe ist ein gerichteter Graph $G = (V, \vec{E})$ $(\vec{E} \subseteq V \times V)$ und eine <u>WF</u> $W : \vec{E} \to I\!N$ mit polynomiellen Gewichten. Ausgabe ist eine Familie $\mathcal{S} \subseteq C(G)$ (falls es eine gibt), so daß $C_i \cap C_j = \emptyset$ für $i \neq j$, $\bigcup_{C \in \mathcal{S}} V(C) = V$ und

$$W(\mathcal{S}) = \sum_{C \in \mathcal{S}} W(C)$$

minimal ist. $\mathcal{S}$ wird $CYCLE$-$COVER$ von G genannt.

Satz 5.7 $CYCLE$-$COVER \in RP$

Beweis: Wir zeigen dies durch Reduktion auf MIN-W-KPM. Wir konstruieren aus G einen bipartiten Graphen $G' = (V_1 \cup V_2, E')$, mit $V_1 \cap V_2 = \emptyset$ und $|V_1| = |V_2|$, so daß

- $V_1 = \{COPY_1(v)|v \in V\}$,
- $V_2 = \{COPY_2(v)|v \in V\}$ und
- $\{COPY_1(v), COPY_2(w)\} \in E' \iff (v, w) \in \vec{E}$.

"COPY" bedeutet: Jeder Knoten $v \in V$ wird in zwei Kopien "$COPY_1(v)$" und "$COPY_2(v)$" aufgespalten. Ein perfektes Matching M in G' induziert uns Teilmengen von $\vec{E}$ mit Ein- und Ausgangsgrad 1 - also Kreise in G. Aufgrund der Minimalität folgt die Behauptung. ■

5.4 *SHORTEST-SUPERSTRING*

In diesem Abschnitt betrachten wir das im Bereich der DNA-Analyse wichtige Shortest-Superstring Problem. Es hat viele Anwendungen in der molekularen Bioinformatik. Eingabe ist eine endliche Menge von Strings $S \subseteq \Sigma^*$ über einem endlichen Alphabet Σ. Ausgabe, soll ein kürzester String $q \in \Sigma^*$ sein, so daß jeder String $s \in S$ ein Unterstring von q (in Zeichen $s \sqsubseteq q$) ist. Das Problem ist NP-hart [GJ79]. Wir werden in diesem Abschnitt einen Approximationsalgorithmus mit einem konstanten $A.R.$ konstruieren. Es gibt wahrscheinlich kein $PTAS$ für dieses Problem, da Blum et al. [BJL$^+$91] gezeigt haben, daß es $MAX-SNP$-hart ist.

Satz 5.8 ([BJL$^+$91]) *SHORTEST-SUPERSTRING ist $MAX - SNP$-hart.*

5.4.1 Elementare Superstring Theorie

Sei Σ ein Alphabet. Für eine Zeichenkette $v \in \Sigma^*$ definieren wir mit $|v|$ die Länge von v. $\Lambda \in \Sigma^*$ bezeichnet das leere Wort und hat die Länge Null. Ferner sei $\Sigma^+ := \Sigma^* \setminus \{\Lambda\}$.

Sei nun $x = uvw \in \Sigma^*$. u, v, w werden als *Faktoren* von x bezeichnet.

- u ist das *Präfix* von x,

- w ist das *Suffix* von x.

Definition 5.8 ($OVERLAP$) *Sei* $s, t \in \Sigma^*$.

$$OVERLAP(s,t) = \text{ längstes Präfix von } t, \text{ das auch Suffix von } s \text{ ist.}$$

Sei dazu $s = uv$ *und* $t = vw$ *und* $v = OVERLAP(s,t)$. *Wie definieren*

$$d(s,t) = DIST(s,t) = |u|.$$

Wir sagen weiterhin

$$PREF(s,t) = u.$$

Also gilt

$$d(s,t) = |PREF(s,t)|.$$

Betrachte nun den String $uvw = PREF(s,t)t$. Die Länge ist dann

$$|uvw| = d(s,t) + |t| = |s| + |t| - |OVERLAP(s,t)|.$$

$PREF(s,t)$ ist somit der kürzeste Superstring von s und t, in dem s vor t auftaucht.

Zur Einfachheit verwenden wir im Anschluß die folgende Notation. Sei $S = \{s_1, ..., s_m\}$. Für $s_i, s_j \in S$ ist $PREF(i,j) = PREF(s_i, s_j)$, $OVERLAP(i,j) = OVERLAP(s_i, s_j)$ und $d(i,j) = |PREF(i,j)|$.

Definition 5.9 *Sei* $s_{i_1}, ..., s_{i_k} \in S$. *Wir bezeichnen mit*

$$[s_{i_1}, ..., s_{i_k}]$$

den Superstring

$$PREF(i_1, i_2), PREF(i_2, i_3), ..., PREF(i_{k-1}, i_k)s_{i_k}.$$

Wir werden nun $SHORTEST\text{-}SUPERSTRING$ in Zusammenhang mit $CYCLE\text{-}COVER$ bringen.

Definition 5.10 *$G_S = (S, \vec{E})$ ist der vollständige gerichtete Graph auf der Knotenmenge S, d.h. $\vec{E} = S \times S$.*

Definition 5.11 *Sei $C = i_1 \to ... \to i_r \to i_1$ ein gerichteter Kreis in G_S. Wir bezeichnen mit $PERIODS(C)$ die Menge der Superstrings, die durch kreisweises Konkatenieren der Präfixe entstehen (sogenannte* Rotation*). Der Index $[k]$ gibt an, daß die Rotation mit $PREF(k, k+1)$ beginnt. Mit $CARD(B)$ bezeichnen wir die Größe der Menge B.*

Sei im Folgenden $C = i_1 \to ... \to i_r \to i_1$ ein gerichteter Kreis in G_S.

Behauptung 5.1 *Jeder String s_{i_j} in C ist ein Unterstring von s^k, für alle $s \in PERIODS(C)$ und hinreichend großes k.*

Beweis: Über Induktion zeigt man, daß für jedes $l \geq 0$ s_{i_j} ein Präfix von

$$PREF(i_j, i_{j+1})...PREF(i_{j+l-1}, i_{j+l})s_{i_{j+l}}$$

ist. Jeweils r aufeinanderfolgende Präfixe bilden eine Umrundung in dem Kreis, d.h. einen String in $PERIODS(C)$. Also ist jeder Unterstring von s_{i_j}, der Länge $W(C)$ in $PERIODS(C)$. Somit definieren wir $k \geq |s_{i_j}|/W(C) + 1$. ■

Folgende Behauptung ist durch die Betrachtung der unendlichen Wiederholung von s leicht zu beweisen:

Behauptung 5.2 *Sei paarweise keiner der Strings $\{s_1, ..., s_m\}$ Unterstring eines anderen und sei $s \in PERIODS(C)$. Wenn jeder dieser Strings ein Unterstring von s^k (für hinreichend großes k) ist, dann existiert ein Kreis mit Gewicht s, der alle diese Strings enthält.*

Behauptung 5.3 *Der Superstring $[s_{i_0}, ..., s_{i_{r-1}}]$ ist ein Unterstring von $PERIODS(C)[i_0]s_{i_0}$.*

Beweis: Der String $[s_{i_0}, ..., s_{i_{r-1}}]$ ist sicher ein Unterstring von $[s_{i_0}, ..., s_{i_{r-1}}, s_{i_0}]$. Dieser ist per Definition gleich $PREF(i_0, i_1)...PREF(i_{r-1}, i_0)s_{i_0} = PERIODS(C)[i_0]s_{i_0}$. ■

Behauptung 5.4 *Es existiert ein Kreis $\tilde{C}$ mit Gewicht $CARD(PERIODS(C))$, der alle Knoten von C enthält.*

BEWEIS: Fixiere ein $s \in PERIODS(C)$ und sei u der kürzeste nichtleere Präfix von s, so daß $s = uv = vu$. Dann teilt $k = |U|$, $W(C)$ und $s = u^j$ mit $j = W(C)/k$. Es ist leicht zu sehen, daß $CARD(PERIODS(C)) = k$. Durch Anwendung von Behauptung 5.2 erhalten wir die Behauptung. ■

Lemma 5.4 *Sei $\mathcal{S}$ eine Lösung von CYCLE-COVER in $G_S = (S, \vec{E})$ mit <u>WF</u> $W \equiv d : \vec{E} \to \mathbb{N}$. Seien $C, C' \in \mathcal{S}$ und $s \in C$, $s' \in C'$. Dann gilt*

$$|\underline{OVERLAP}(s, s')| \leq W(C) + W(C')$$

.

BEWEIS: Sei $x = PERIODS(C)$ und $x' = PERIODS(C')$. Es gilt, daß $x \neq x'$, da $\mathcal{S}$ eine Lösung von $CYCLE\text{-}COVER$ in $G_S = (S, \vec{E})$ mit <u>WF</u> $W \equiv d : \vec{E} \to \mathbb{N}$ ist. Durch Behauptung 5.4 wissen wir, daß $CARD(x) = W(C)$ und $CARD(x') = W(C')$.

Nehmen wir nun an, daß der Overlap von s und s' ein String u ist, mit $|u| \geq W(C) + W(C')$. Wir unterscheiden dazu 2 Fälle:

$W(C) = W(C')$: Dann haben s und s' einen Unterstring y mit $|y| = W(C) = W(C')$ gemeinsam. Aus Behauptung 5.1 folgt dann $x = PERIODS(y) = x'$, was ein Widerspruch zur Minimalität von $\mathcal{S}$ ist.

$W(C) \neq W(C')$: O.B.d.A. $W(C) > W(C')$. Sei $u_{i,j}$ der Unterstring von u der mit dem i-ten Symbol beginnt und dem j-ten Symbol endet. Dann haben wir

$$u_{1,W(C)} = u_{1+W(C'),W(C)+W(C')} = u_{1,W(C')}u_{1+W(C'),W(C)}$$

Das zeigt, daß $PERIODS(C)$ weniger als $W(C)$ Rotationen beinhaltet, was durch Behauptung 5.4 zu einem Widerspruch führt.

■

Lemma 5.5 *Sei $C_i \in C(G)$ ein Zyklus von G_S mit <u>WF</u> $W \equiv d$,*

$$C_i = i_1 \to ... \to i_r \to i_1.$$

Dann gilt

$$|[s_{i_1}, ..., s_{i_r}]| \leq W(C_i) + |s_{i_1}|.$$

BEWEIS: Einfache Anwendung von Behauptung 5.3. ■

5.4.2 Ein Approximationsalgorithmus mit $A.R.$ 4

Eingabe: $\{s_1, ..., s_m\} = S \subseteq \Sigma^*$.

Schritt 1: Konstruiere $G_S = (V, \vec{E})$ mit $\underline{W} \equiv d : \vec{E} \to I\!N$.

Schritt 2: Wende Cycle-Cover-Algorithmus auf G_S und W an. Sei dabei $\mathcal{S} = \{C_1, ..., C_k\}$ die Ausgabe.

Schritt 3: Für alle Kreise $i_1 \to ... \to i_r \to i_1 = C_i \in \mathcal{S}$, konstruiere $\bar{s}_i = [s_{i_1}, ..., s_{i_r}]$

Ausgabe: $q = \bar{s}_1 \bar{s}_2 ... \bar{s}_k$.

Satz 5.9 *Der obige Algorithmus ist ein polynomieller Approximationsalgorithmus für $SHORTEST$-$SUPERSTRING$ mit einem A.R. von* 4.

BEWEIS: Sei $OPT(S)$ die Länge des kürzesten Superstrings q und $\mathcal{S} = \{C_1, ..., C_k\}$ wie in Schritt 2 definiert. Wir definieren $W_i = W(C_i)$ und $l_i \max_{s \in V(C_i)} |s|$. Aus Lemma 5.4 folgt, daß der Gesamtoverlap höchstens $2 \sum_{i=1}^{k} W_i$ ist. Also hat der String eine Länge von mindestens $\sum_{i=1}^{k} l_i - 2W_i$. Somit folgt, daß

$$OPT(S) \geq \max\{\sum_{i=1}^{k} W_i, \sum_{i=1}^{k} l_i - 2W_i\}.$$

Aus Lemma 5.5 folgt, daß der Ausgabestring q eine Länge

$$|q| \leq \sum_{i=1}^{k}(l_i + W_i)$$

hat. Also erhalten wir folgende Schranke:

$$\begin{aligned}
|q| &\leq \sum_{i=1}^{k}(l_i + W_i) \\
&= \sum_{i=1}^{k}(l_i - 2W_i) + \sum_{i=1}^{k} 3W_i \\
&\leq OPT(S) + 3OPT(S) \\
&= 4OPT(S)
\end{aligned}$$

■

5.5 *MAX-FLOW*

Als nächstes wollen wir das Max-Flow Problem mit unär dargestellten polynomiell beschränkten Gewichten auf Perfect Matching reduzieren. Dazu zunächst einige Definitionen.

Definition 5.12 *Transport-Netzwerk*
Ein Transport-Netzwerk ist eine endlicher gerichteter Graph $T = (V, E, s, t)$ mit $s \in V$ als Quelle (source) und $t \in V$ als Senke (sink). Der Eingangsgrad von s und der Ausgangsgrad von t seien ohne Einschränkung 0. Ein Fluß von T ist eine Funktion $f : E \to \mathbb{N}$, so daß

$$\forall v \in V \setminus \{s, t\} \quad \sum_{u} f(u, v) = \sum_{u} f(v, u),$$

d.h. für alle Knoten gilt das Kirchhoff'sche Gesetz.

Definition 5.13 *Transport-Netzwerk mit Kapazitäten*
Ein Transport-Netzwerk mit Kapazitäten (TNC) ist ein Tupel

$$C = (V, E, s, t, c),$$

wobei (V, E, s, t) *ein Transport-Netzwerk und* $c : E \rightarrow I\!N$ *eine Kapazitätsfunktion sind. Ein Fluß in einem TNC ist ein Fluß in einem Transport-Netzwerk* (V, E, s, t), *für den zusätzlich*

$$0 \leq f(u, v) \leq c(u, v)$$

gilt. Der Wert eines solchen Flußes ist dann

$$v(c, f) = \sum_v f(s, v).$$

Definition 5.14 *Max-Flow Problem*
Gesucht ist f *mit* $v(c, f) = \max_{f'} v(c, f')$.

Insbesondere werden wir nun das 0-1 Max-Flow Problem betrachten, in der jede Kante die Kapazität 1 oder 0 hat. (Im Falle der Kapazität 0 kann die Kante auch weggelassen werden.)

Satz 5.10 0-1 *Max-Flow* $\leq_P$ *KPM.*

Beweis: Sei C das Transport-Netzwerk. Wir gehen analog zum Beweis von Satz 5.3 vor. Ein System kantendisjunkter Wege von s nach t induziert einen 0-1 Fluß in C, da jeder dieser Wege zum maximalen Fluß beiträgt. Die Anzahl dieser maximalen kantendisjunkten Wege und damit auch der Wert für den Max-Flow kann durch $m = |E|$ abgeschätzt werden. Wir konstruieren jetzt eine Familie ungerichteter Hilfsgraphen $\{H_k\}$ wie folgt:

$$V(H_k) = \{s_1, \ldots, s_k\} \cup \{(e, 1)|e \in E\} \cup \{(e, 2)|e \in E\} \cup \{t_1, \ldots, t_k\}$$

$$\begin{aligned} E(H_k) \quad = \quad & \{(s_i, (e, 1)))|\exists v \text{ mit } e = (s, v) \text{ und alle } 1 \leq i \leq k\} \\ & \cup\{((e, 1), (e, 2))|e \in E\} \cup \{((e, 1), (f, 2))|e \cap f \neq \emptyset\} \\ & \cup\{((e, 2), t_i)|\exists v \text{ mit } e = (v, t) \text{ und alle } 1 \leq i \leq k\}. \end{aligned}$$

Aus einem Perfekten Matching in H_k können wir nun ein maximales System von s-t Wegen wie folgt konstruieren

In dem Perfekten Matching von H_k wird jeder Knoten der Form s_i mit einem Knoten $(e, 1)$ verbunden. Daher kann der zugehörige Zwillingsknoten $(e, 2)$ nicht mit $(e, 1)$, sondern nur mit einem Knoten $(f, 1)$ verbunden sein. Dabei sind die Kanten e und f in C adjazent, d.h. sie haben einen gemeinsamen Knoten. Mit $(f, 1)$ fahren wir jetzt fort, bis der letzte Zwillingsknoten mit t_j verbunden ist. Auf diese Weise ordnen wir jedem s_i einen Pfad $s_i, (e, 1), (f, 1), \ldots, t_j$ in C zu. Aufgrund der Matching-Eigenschaft sind alle diese Pfade disjunkt. Das Perfekte Matching in H_k induziert also ein k-Wegesystem in C. Daher testen wir parallel jeden Graphen H_k auf die Existenz eines Perfekten Matchings.

Nun brauchen wir mittels binärer Suche nur noch den größten Graphen zu suchen, der ein Perfektes Matching besitzt, ein Perfektes Matching zu konstruieren und dann den entsprechenden Max-Flow auszugeben. ■

Als Korollar ergibt sich

Korollar 5.2 *Unary Max-Flow $\leq_P$ KPM.*

Beweis: Man betrachte den Multigraphen G', der dadurch entsteht, daß eine Kante mit Kapazität k in G durch k Kanten mit Kapazität 1 ersetzt wird. ■

5.6 Ein Approximationsalgorithmus mit $A.R.$ 1.5 für das ΔTSP

Wir betrachten in diesem Abschnitt das TSP mit Dreiecksungleichung. In Kapitel 1.3 haben wir schon einen Algorithmus mit $A.R.$ 2 für dieses Problem vorgestellt. Diesen $A.R.$ werden wir nun mit Matching-Techniken auf 1.5 verbessern.

Sei im folgenden für einen Graph $G = (V, E)$ und $X \subseteq V$ und $Y \subseteq P_2(V)$,

- $G[X] = (X, E \cap P_2(X))$ der von X *induzierte Untergraph von G* und
- $G \sqcup Y = (V, E \cup Y)$ der um Y *erweiterte Multigraph von G*.

Nehmen wir an wir haben einen Eulergraph G und τ ist eine Eulerreise in G. Wir können nun aus diesem Eulerkreis einen Hamiltonkreis π konstruieren, indem wir τ folgen und die Orte überspringen, die wir schon besucht haben. Wir bezeichnen mit

$$\pi = SCM(\tau)$$

die so entstandene Hamilton-Tour. Wenn wir Kantengewichte haben, folgt durch die Dreiecksungleichung trivialerweise

$$W(\pi) \leq W(\tau).$$

Eine solche Reise können wir leicht in $O(|E|)$ Zeit konstruieren.

Lemma 5.6 *Sei $G = (V, E)$ ein Graph. Dann ist die Zahl*

$$|\{v \in V | \deg(v) \textit{ in } G \textit{ ist ungerade } \}|$$

gerade.

Beweis: Wir haben

$$\sum_{v \in V} \deg(v) = 2|E|.$$

Also ist

$$\sum_{v \in V : \deg(v) \text{ist ungerade}} \deg(v) + \sum_{v \in V : \deg(v) \text{ist gerade}} \deg(v)$$

eine gerade Zahl. ■

Nun können wir unseren Approximationsalgorithmus präsentieren:

Input: Eine Instanz (G, W) vom metrischen TSP.

Schritt 1: Berechne den MST von (G, W) mit Ergebnis $T = (U, C)$.

Schritt 2: Sei X die Menge der Knoten mit ungraden Grad in T.

Schritt 3: Berechne $MIN\text{-}W\text{-}KPM$ M in $T[X]$.

Schritt 4: Konstruiere $G' = T \sqcup M$. (Beachte, daß G' ein Eulergraph ist (Lemma 5.6).

Schritt 5: Berechne Eulerkreis τ in G'.

Schritt 6: Konstruiere TSP-Tour $\pi = SCM(\tau)$.

Satz 5.11 *Der Algorithmus ist ein polynomieller Approximationsalgorithmus für das metrische TSP mit einem A.R. von* 1.5.

Beweis: Die Laufzeit ist klar. Für die Kosten von π gilt:

$$W(\pi) \leq W(G') = W(T) + W(M)$$

Sei π^* eine optimale Rundreise. Es gilt

$$W(T) \leq W(\pi^*).$$

Seien M_1 und M_2 zwei verschiedene PMs in $T[X]$. Durch die Dreiecksungleichung folgt

$$W(\pi^*) \geq W(M_1) + W(M_2).$$

Also gilt für das minimal gewichtete PM M,

$$W(\pi^*) \geq 2W(M) \Rightarrow W(M) \leq 1/2W(\pi^*).$$

Insgesamt gilt also

$$W(\pi) \leq W(\pi^*) + 1/2W(\pi^*) = 3/2W(\pi^*).$$

■

Kapitel 6

Ein PTAS für dichte NP-harte Probleme

Bisher haben wir immer nur den Worstcase von Berechnungproblemen betrachtet. Das heißt, wir sind davon ausgegegangen, daß man Instanzen erhält, die besonders schwer zu lösen sind. In diesem Kapitel wollen wir nur dichte Instanzen von Berechnungsproblemen betrachten, die häufig einfacher zu lösen sind.

Ein Graph G auf n Knoten ist zum Beispiel dicht, wenn der Knotengrad aller Knoten $\Omega(n)$ ist. Es gibt eine Vielzahl von Problemen, die auf dichten Graphen leichter zu lösen sind, als bei allgemeinen Graphen als Eingabe. Bei einem Knotengrad von mindestens $n/2$ kann man zum Beispiel hamiltonische Kreise finden oder approximativ die Anzahl der maximum Matchings bestimmen. Beide Probleme sind im allgemeinen Fall NP- bzw. $\#P$-hart. E. Dahlhaus, P. Hajnal und M. Karpinski fanden sogar 1993 für $n/2$-dichte Graphen einen parallelen Algorithmus, der mit $O(n)$ Prozessoren in $O(\log^4 n)$ Tiefe auf einer $CREW$-$PRAM$ hamiltonische Kreise findet [DHK93]. Zur Definition von $PRAMS$ siehe auch das Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther) [K91] .

6.1 Ein $PTAS$ für dichte Instanzen von $MAX\text{-}CUT$

In Kapitel 1.3 haben wir den Begriff $PTAS$ eingeführt. Für $MAX\text{-}CUT$ existiert zum Beispiel ein $PTAS$, wenn wir die Menge der Eingaben auf dichte Graphen beschränken, d.h. auf Graphen mit Minimalgrad $\Omega(n)$. Wir nennen dieses Problem $DENSE\text{-}MAX\text{-}CUT$. Später werden wir das Verfahren auf Graphen verallgemeinern, die einen Durchschnittsgrad von $\Omega(n)$ haben.

Satz 6.1 ([AKK95]) *Es existiert ein $PTAS$ für $DENSE\text{-}MAX\text{-}CUT$.*

Beweis: (Hauptidee)

Wenn wir eine Aufteilung von V in V_1 und V_2 haben, so daß die Anzahl der Kanten von V_1 nach V_2 maximal ist, können wir folgende Beobachtung machen: Sei v ein beliebiger Knoten in V_1. Die Majorität der Nachbarn von v liegen in V_2. Wenn dies nicht der Fall wäre, könnten wir v aus V_1 entfernen und in V_2 einfügen. Dieses ergäb einen größeren Schnitt, was ein Widerspruch zur Optimalität wäre.

Der Algorithmus hat im wesentlichen zwei Schritte:

Schritt 1: Wähle zufällig gleichverteilt und unabhängig $O(\log n)$ Knoten aus, die eine Menge S bilden. Man kann zeigen, daß mit hoher Wahrscheinlichkeit jeder Knoten in V einen Nachbarn in S hat.

Schritt 2: Für jede der $2^{O(\log n)}$ Partitionierungen der Menge S konstruiere eine Partitionierung der Gesamtknotenmenge wie folgt und wähle die *beste* aus:

Ordne jeden Knoten der Seite zu, die entgegengesetzt zu der Seite liegt, in der die Majorität der Nachbarn in S ist.

Man kann nun zeigen, daß der beste so erzeugte Schnitt sehr nahe am optimalen Schnitt liegt.

■

Der skizzierte Algorithmus ist die kombinatorische Übersetzung des Approximationsschemas von Arora, Karger und Karpinski [AKK95]. Man kann das MAX-CUT Problem aber auch wie folgt als *quadratisches ganzzahliges Optimierungsproblem* beschreiben:

Wir ordnen jedem Knoten $i \in V$ eine Variable x_i zu, die die Werte 0 und 1 annehmen kann. $x_i = 0$ heißt, daß der Knoten in der Partitionierung *links* liegt, und $x_i = 1$, daß er *rechts* liegt. Eine 0/1 Belegung des entsprechenden Vektors gibt uns also eine Partitionierung. Folgendes Polynom gibt uns die Anzahl der Kanten zwischen den Partitionen an: Für $x \in \{0,1\}^n$ sei

$$P_G(x) = 1/2 \sum_{\{i,j\} \in E} \underbrace{(x_i(1-x_j) + x_j(1-x_i))}_{=p_{i,j}}$$

Es gilt, daß $p_{i,j} = 1$ genau dann, wenn $x_i \neq x_j$ ist, also i und j auf verschiedenen Seiten liegen. Also erhalten wir den Summanden 1 für jede Kante, die zum Schnitt beiträgt.

Nun stellt sich das Problem P_G zu maximieren, was natürlich NP-hart ist. P_G hat jedoch eine besondere Form, es ist ein *SMOOTH INTEGER PROGRAM* [AKK95]. Für solche Programme haben Arora, Karger und Karpinski Approximationsschemata konstruiert. Diese Approximationsschemata werden wir im nächsten Abschnitt betrachten.

Für weite Approximationsalgorithmen auf NP-harten dichten Instanzen siehe auch [K97] und [KR98], [KZ97b], [KZ98], [KWZ97].

6.2 Ein *PTAS* für *SMOOTH INTEGER PROGRAMS*

Kernpunkt des PTAS [AKK95] ist die Idee des *exhaustiv sampling*: Wir wählen eine kleine Teilmenge der Lösungsmenge (randomisiert) aus und probieren dort alle verschiedenen Lösungen. Danach wird die *beste* Lösung zu einer Gesamtlösung erweitert. Es wird gezeigt, daß wir somit eine Gesamtlösung erhalten, die beliebig nahe am Optimum liegt.

Im Detail entwickeln wir ein PTAS für ein allgemeines ganzzahliges numerisches Programm, auf das eine Vielzahl von kombinatorischen Optimierungsproblemen reduziert werden können:

Definition 6.1 (smooth degree-d integer program)

Ein smooth degree-d integer program (kurz SD-d-IP) ist durch folgendes Optimierungsproblem definiert:

$$\begin{aligned} \max\ & p(x_1, ..., x_n) \\ \text{s.t. } & x_i \in \{0,1\} \quad i = 1, ..., n \end{aligned}$$

mit p einem Polynom in $x_1, ..., x_n$, Grad $\deg(p) \leq d$ und der Eigenschaft, daß jedes Monom vom Grade i einen Koeffizienten aus $O(n^{d-i})$ hat. (Der Minimierungsfall ist analog definiert.)

Mit Lemma 3.9 und Lemma 3.10 an der Hand werden wir nun in den folgenden 2 Sätzen zeigen, daß Approximationsschemata für smooth degree-d integer program*s* existieren. Zunächst zeigen wir das für den Fall $d = 2$ und verallgemeinern das Resultat auf beliebiges $d \in O(1)$.

Satz 6.2 ([AKK95]) *Angenommen, es gibt eine $0-1$-Lösung für das ganzzahlige quadratische Programm*

$$x^t A x + b^t x \geq c \quad x \in \{0,1\}^n$$

mit $A \in O(1)^{n \times n}, b \in O(n)^n, c \in O(1)$. Dann können wir für alle $\epsilon > 0$ in Zeit $n^{O(1/\epsilon^2)}$ (mit beschränkter Fehlerwahrscheinlichkeit) eine $0-1$-Lösung x finden, für die

$$x^t A x + b^t x \geq c - \epsilon n^2$$

gilt.

Beweis: Die Idee ist, das quadratische Programm auf ein relaxiertes lineares Programm zu reduzieren und dann mit der Raghavan-Thompson Technik aus der relaxierten Lösung eine *gute* 0-1-Lösung zu konstruieren. Die Reduktion wird eine Laufzeit von $n^{O(1/\epsilon^2)}$ haben, was für hinreichend kleine ϵ die Laufzeit des LP-Solvers (vgl. Abschnitt 8.1) majorisiert. Es ist klar, daß das Ergebnis nur im Fall $c > \epsilon n^2$ Sinn macht. Der andere Fall ist trivial.

Sei nun x^* eine Lösung unseres 0-1-quadratischen Programms. Wir können es wie folgt umformulieren, indem wir $r = (r_i)$ durch

$$r_i = \sum_j x_j a_{j,i} + b_i$$

definieren, $r^* := r(x^*)$ setzen und obiges Programm durch

$$\begin{aligned} &x^t A + b^t = (r^*)^t \\ &(r^*)^t x \geq c \\ &x \in \{0,1\}^n \end{aligned}$$

ausdrücken.

Nehmen wir nun an, daß wir eine Näherung r an r^* konstruieren können, so daß $|r_i^* - r_i| < \epsilon n$ für alle i. Wir definieren nun ein leicht modifiziertes lineares Programm:

$$\begin{aligned} &\max r^t x \\ &x^t A + b \leq r + \epsilon n \\ &x^t A + b \geq r - \epsilon n \\ &0 \leq x \leq 1 \end{aligned}$$

(Die Ungleichungen und arithmetischen Operationen auf den Vektoren verstehen sich komponentenweise.)

Das Programm ist lösbar, denn x^* ist eine zulässige Lösung mit

$$r^t x^* = (r^*)^t x^* - (r^* - r)^t x^* \geq c - \epsilon n^2.$$

Wir lösen obiges Programm mit linearer Programmierung, und erhalten dann eine gebrochen-rationale Lösung x und runden diese mit Hilfe von Lemma 3.10 zu einem $0-1$-Vektor y. Nach Lemma 3.10 erhalten wir y_i, so daß $r^t y \geq r^t x - O(n\sqrt{n \log n})$.

Sei $\delta = x^t A + b^t - r^t$. Nach Definition unseres linearen Programms ist dann $|\delta_i| < \epsilon n$. Weil y ein $0-1$-Vektor ist, ist dann $|\delta y| \leq \epsilon n^2$. Beachte, daß $r^t x \geq r^t x^*$.

$$\begin{aligned} y^t A y + by &= (y^t A + b)y \\ &= \underbrace{(y^t A + b - (x^t A + b))y}_{=O(n^{3/2}\sqrt{\log n})} + \underbrace{\delta y}_{\geq -\epsilon n^2} + \underbrace{r^t y}_{\geq (c - \epsilon n^2 - O(n^{3/2}\sqrt{\log n}))} \\ &\geq c - (2\epsilon + o(1))n^2 \end{aligned}$$

Mit folgender Methode können wir in $n^{O(1/\epsilon^2)}$ eine Menge mit $n^{O(1/\epsilon^2)}$ möglichen r konstruieren, von denen eines mit Wahrscheinlichkeit $1/2$ die gewünschten Eigenschaften hat.

Wähle zufällig und unabhängig voneinander $k = O((\log n)/\epsilon^2)$ Indizes aus, die eine Multimenge S bilden. Die Menge der Samples wird nun gebildet, indem wir den durch S bestimmten Positionen alle verschieden $0-1$-Kombinationen zuweisen und die Näherungen r an r^* durch

$$r_i = b_i + \frac{n}{k} \sum_{j \in S} s_j a_{j,i}$$

bestimmen (s_j ist der der entsprechenden Position zugewiesende Wert). Beachte, daß jeder Koeffizient konstant ist.

Da wir alle möglichen Kombinationen ausprobieren, haben wir sicher auch ein Sample, bei dem $s_j = x_j^*$ für alle $j \in S$ gilt. Nennen wir dieses Sample $\bar{r}$. Anwendung des Sampling-Lemmas 3.9 ergibt, daß für jedes i mit Wahrscheinlichkeit $1 - \frac{1}{2n}$, $\bar{r}_i \in [r_i^* - \epsilon n, r_i^* + \epsilon n]$ gilt. D.h. mit Wahrscheinlichkeit $1/2$ sind alle $\bar{r}_i \in [r_i^* - \epsilon n, r_i^* + \epsilon n]$.

■

Mit ähnlichen Methoden können wir nun induktiv ein smooth degree-d integer program auf ein smooth degree-$(d-1)$ integer program (im approximativen Sinn) reduzieren:

Satz 6.3 ([AKK95]) *Sei OPT der Maximalwert eines smooth degree-d integer program p. Dann existiert für alle $\epsilon > 0$ ein Algorithmus mit Laufzeit $n^{O(1/\epsilon^2)}$, der eine $0-1$-Lösung x*

produziert, so daß

$$p(x_1, ..., x_n) \geq \ OPT \ - \epsilon n^d$$

gilt. (Für Minimierungsprobleme gilt das Analoge.)

Beweis: Via Induktion über den Grad. Der Fall $d = 2$ ist mit Satz 6.3 bewiesen. Sei nun

$$p(x_1, ..., x_n) = \sum_{i=1}^{d} \sum_{m \in M(i)} k(m)m(x_1, ..., x_n)$$

mit $M(i)$ die Menge der Monome mit Grad i von p. Nach der Definition von smooth degree-d integer Programmen ist $k(m) \in O(n^{d-i})$, wenn $m \in M(i)$.

$$q(x_1, ..., x_n) = \sum_{m \in M(d)} k(m)m(x_1, ..., x_n)$$

ist das Teilpolynom von p, das nur aus den Monomen vom Grad d besteht. Sei wieder x^* die Stelle, für die p sein Optimum annimmt und $r^* := q(x^*)$.

Wir wollen nun in $n^{O(1/\epsilon^2)}$ eine Approximation r an r^* ereichen, so daß $|r - r^*| < \epsilon n^d$.

Sei dazu $n' := |M(d)| \leq \binom{n}{d} \leq n^d$. Wähle unabhängig und gleichverteilt voneinander $k = O(\frac{\log n'}{\epsilon^2}) = O(\frac{\log n}{\epsilon^2})$ Monome aus $M(d)$ aus, die eine Menge M bilden. Sei $X(M)$ die Menge der Variablen, die in M vorkommen. Weil der Grad der Monome aus $M(d)$ d ist, ist $|X(M)| \leq d|M| = O(\frac{\log n}{\epsilon^2})$.

Wir berechnen wie beim Beweis von Satz 6.2 $n^{O(1/\epsilon^2)}$ Samples r, indem wir die Elemente von $X(M)$ mit allen möglichen 0-1-Belegungen belegen und

$$r = \frac{n'}{k} \sum_{m \in M} k(m)m(x)$$

setzen. Betrachte nun die Belegung, die durch Einsetzung von x^* entstehen würde, d. h.

$$x_i = x_i^* \quad \forall i \in X(M)$$

Sei dazu $\bar{r}$ das entsprechende Sample. Mit Lemma 3.9 folgt nun, daß

$$\begin{aligned} \bar{r} \ &\in \ [q(x^*) - \epsilon n', q(x^*) + \epsilon n'] \\ &\subseteq \ [r^* - \epsilon n^d, r^* + \epsilon n^d] \end{aligned}$$

mit Wahrscheinlichkeit $\geq 1 - \frac{1}{2n'} \geq 1 - \frac{1}{2n}$.

Sei nun $\tilde{p} = \frac{1}{n}\sum_{i=1}^{d-1}\sum_{m\in M(i)} k(m)m(x|X^*(M))$ das smooth degree-$(d-1)$ Polynom, das durch Entfernen der Grad-d-Monome und Festsetzen der Variablen in $X(M)$ auf die Werte von x^* entsteht.

Bestimme nun nach I.V. in $n^{O(1/\epsilon^2)}$ eine Lösung $\bar{x}$ von

$$\max \tilde{p}(x) \quad x \in \{0,1\}^n,$$

so daß

$$\tilde{p}(\bar{x}) \quad \geq \quad \max_x \tilde{p}(x)) - \epsilon n^{d-1}$$

Es gilt dann

$$\begin{aligned} n\tilde{p}(\bar{x}) + \bar{r} &> n\tilde{p}(\bar{x}) + r^* - \epsilon n^d \\ &\geq \underbrace{n\tilde{p}(x^*) + r^*}_{=p(x^*)} - 2\epsilon n^d. \end{aligned}$$

■

Man kann das Verfahren mit Standardmethoden derandomisieren, indem lediglich seine randomisierten Komponenten deterministisch simuliert werden:

Random Sampling Wir konstruieren einen *Constant-Degree-Expander* mit n Knoten und benutzen die Knoten, die auf einem Random-Walk der Länge $O((\log n)/\epsilon^2)$ besucht worden sind. Die Anzahl der möglichen Wege ist $n^{O(1/\epsilon^2)}$.

Randomized Rounding Mittels der Methode der bedingten Wahrscheinlichkeiten.

Es gilt zu beachten, daß man durch Densifizierung nicht immer bessere Algorithmen erhält (Zum Beispiel beim TSP und auch bei Longest Path Problemen [FK98]).

Kapitel 7

Approximationsalgorithmen für MAX-SAT und MAX-LIN

Wir werden in diesem Kapitel ein häufig benutztes Hilfsmittel in der Konstruktion von Approximationsalgorithmen kennenlernen. Mit einem randomisierten Algorithmus konstruieren wir für eine Instanz von MAX-SAT (und MAX-LIN, s. Anhang A.1) eine Belegung, die mit hoher Wahrscheinlichkeit *viele* der Klauseln (Gleichungen) erfüllt. Danach zeigen wir, wie man diese randomisierte Methode *derandomisieren* kann und somit einen deterministischen Algorithmus mit der gleichen Gütegarantie erhält.

Sei f eine Formel in konjunktiver Normalform

$$f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$$

MAX-$SAT(f)$ ist definiert als

$$MAX\text{-}SAT(f) = \max_{x \in \{0,1\}^n} |\{i | c_i(x) = 1\}|,$$

und MAX-SAT ist das Problem für eine gegebene Formel f, ein $s \in \{0,1\}^n$ zu konstruieren, so daß MAX-$SAT(f) = |\{i | c_i(s) = 1\}|$.

Wir beweisen folgenden Satz.

Satz 7.1 *Für das MAX-SAT Problem existiert ein polynomieller Approximationsalgorithmus mit Approximationsgüte* $1/(1-2^{-k})$ *(k ist dabei die Anzahl der Literale pro Klausel).*

Beweis: Seien $C = \{c_1, ..., c_m\}$ die Klauseln von $MAX\text{-}SAT$. O.B.d.A sind alle Klauseln erfüllbar. Wir werden zunächst ein randomisiertes Verfahren beschreiben , welches wir später derandomisieren.

Nehmen wir an, wir hätten eine zufällige Belegung der Variablen $x_1, ..., x_n$ mit Wahrheitswerten. Für c_i sei $p(c_i)$ die Wahrscheinlichkeit, daß c_i von dieser Belegung erfüllt wird. Da f_{c_i} nur von k Variablen abhängt und es höchstens eine nicht erfüllende Belegung gibt, ist

$$p(c_i) \geq 1 - \frac{2^{n-k}}{2^n} = (1 - 2^{-k}).$$

Die erwartete Anzahl $p(f)$ der erfüllten Klauseln ist

$$p(f) = \sum_{i=1}^{m} p(c_i) \geq m(1 - 2^{-k}).$$

Wir haben somit einen randomisierten Algorithmus für $MAX\text{-}SAT$ mit Gütegarantie $(1 - 2^{-k})$, weil nie mehr als m Klauseln gleichzeitig erfüllt werden können.

Wir derandomisieren ihn mit der folgenden Greedymethode: Weise x_1 den Wert t zu, der die meisten Klauseln erfüllt. Bezeichne die resultierende Funktion als $f[x_1 = t]$. Fahre mit $x_2, x_3, ...$ fort, bis alle Variablen belegt sind. Die Gütegarantie bleibt bei $(1 - 2^{-k})$:

Sei $p(f[x_1 = t_1, ..., x_i = t_i])$ die erwartete Anzahl der erfüllten Funktionen mit festen $x_1, ..., x_i$ und freien $x_{i+1}, ..., x_n$. Aus der Definition des Erwartungswertes folgt,

$$p(f) = \frac{1}{2}(p(f[x_1 = 1]) + p(f[x_1 = 0])).$$

Bei der obigen Wahl von x_1 folgt dann

$$p(f[x_1 = t_1]) \geq p(f).$$

Das gleiche Argument gilt dann für die Wahl von $x_2, ...$ Der Erwartungswert wird also nie kleiner. Schließlich gilt

$$p(f[x_1 = t_1, ..., x_n = t_n]) \geq p(f) \geq m(1 - 2^{-k}).$$

Der Algorithmus ist leicht in polynomieller Zeit implementierbar. ■

Wir haben also folgenden deterministischen Approximationsalgorithmus für das MAX-$3SAT$ Problem (alle Klauseln haben die Länge 3) mit einem $A.R.$ von 8/7.

Eingabe: Eine KNF $f = c_1 \wedge c_2 \wedge ... \wedge c_m$ mit Klauseln der Länge 3.

$\phi := f$; $\beta := \Lambda$ (empty string);

For $i = 1$ to n do

- $\phi := \phi[x_i = t]$ for $p(\phi[x_i = t]) \geq p(\phi)$;
- $\beta := \beta t$;

Ausgabe: β

Wir können MAX-SAT aber leider nicht beliebig genau approximieren, wie wir in Kapitel 10 sehen werden. Håstad [H97a] hat vor kurzem gezeigt ([H97a], Theorem 3.1), daß obiger Approximationsalgorithmus für MAX-$3SAT$ optimal ist. (Für den Begriff der *approximativen* NP-Härte siehe auch Kapitel 10).

Satz 7.2 ([H97a]) *Für jedes $\epsilon > 0$ ist MAX-$3SAT$ NP-hart für den $A.R.$ $8/7 - \epsilon$.*

■

Wir betrachten jetzt das MAX-$3LIN$ Problem: gegeben seien m lineare Gleichungen (modulo 2) über n Variablen mit genau 3 Variablen pro Gleichung. Berechne $s \in \{0,1\}^n$, so daß eine maximale Anzahl der Gleichungen erfüllt wird.

Man macht sofort die Beobachtung, daß ein zufällig gewähltes $s^* \in_R \{0,1\}^n$ die Hälfte der Gleichungen erfüllt. Also ist der erwartete Anteil der erfüllten Gleichungen 1/2. Man benutzt eine ähnliche Greedymethode (wie bei MAX-SAT) um den resultierenden Approximationsalgorithmus zu *derandomisieren*. Wir haben somit

Satz 7.3 *Es existiert ein polynomieller Approximationsalgorithmus für das MAX-3LIN Problem mit A.R. 2.*

■

Håstad ([H97a], Theorem 2.3) hat gezeigt, daß auch diese Approximationsgüte optimal ist(!).

Satz 7.4 ([H97a]) *Für jedes $\epsilon > 0$ ist MAX-3LIN NP-hart für einen A.R. $2 - \epsilon$.*

■

7.1 Ein verbesserter Algorithmus für *MAX-SAT*

Wir überführen unsere Instanz $C = \{c_1, ..., c_m\}$ von *MAX-SAT* in ein ganzzahliges lineares Programm wie folgt:

$$
\begin{array}{lll}
(ILPMS) & & \max \sum_{i=1}^{m} z_i \\
& \text{s.t.} & z_j \leq \sum_{i \in \{1,...,n\}: x_i \in c_j} y_i + \sum_{i \in \{1,...,n\}: \bar{x}_i \in c_j} (1 - y_i) \quad \text{für } j = 1, ..., m \\
& & y_i, z_j \in \{0, 1\} \quad i = 1, ..., n, j = 1, ..., m
\end{array}
$$

$y_i = 1$ bedeutet x_i ist wahr und $z_j = 1$ bedeutet c_j ist erfüllt. Wir lösen nun die *LP*-Relaxierung und erhalten die Lösung (y^*, z^*). Wir setzen die x_i unabhängig voneinander mit Wahrscheinlichkeit y_i^* auf den Wert wahr (Siehe auch Lemma 3.10). Das ergibt einen randomisierten Algorithmus für *MAX-SAT*. Eine elementare Umformung ergibt, daß jede Klausel c_j einen erwarteten Wert von

$$\left(1 - \left(1 - \frac{1}{|c_j|}\right)^{|c_j|}\right) z_j^*$$

zur Gesamtsumme beträgt. Die Optimallösung ist von oben durch $\sum_j^m z_j^*$ beschränkt, so daß man wie zuvor das Verfahren derandomisieren kann. Wir erhalten folgenden Satz.

Satz 7.5 *Die beschriebene Methode ist ein polynomieller Approximationsalgotithmus für MAX-SAT mit einem A.R.*

$$\frac{1}{1-\left(1-\frac{1}{q}\right)^q}.$$

q ist dabei die maximale Länge einer Klausel.

Korollar 7.1 *Es existiert ein polynomieller Approximationsalgorithmus für MAX-SAT mit einem A.R.* $e/(e-1) \approx 1.582$.

Beweis: Es gilt $\left(1-\frac{1}{q}\right)^q < 1/e$. ■

Nun kombinieren wir beide uns bekannten Algorithmen für MAX-SAT und wählen die bessere Lösung aus. Dieser einfache Ansatz gibt uns einen verbesserten Approximationsalgorithmus für MAX-SAT.

Satz 7.6 *Es existiert eine polynomieller Approximationsalgorithmus für MAX-SAT mit einem A.R.* 4/3.

Beweis: Wir wählen also die beste Lösung aus. Der Beitrag jeder Klausel c_j mit $|c_j| = k$ ist mindestens

$$\begin{aligned} & 1/2((1-2^{-k}) + (1-(1-1/k)^k)) \\ \geq\ & 1/2(2-2^{-k}-(1-1/k)^k) \\ \geq\ & 3/4 \end{aligned}$$

■

Kapitel 8

Approximationsalgorithmen für *MAX-CUT*

In diesem Kapitel betrachten wir das *MAX-CUT* Problem. Eingabe ist ein Graph $G = (V, E)$. Ziel ist es eine Partionierung von V in $V = V_1 \cup V_2$, $(V_1 \cap V_2 = \emptyset)$ zu finden, so daß der Schnitt $|E(V_1, V_2)|$ maximal ist. Wir können zusätzlich eine Gewichtsfunktion $w : E \to \mathbb{R}^+$ verlangen. In diesem Fall wollen wir das Gewicht des Schnittes maximieren.

Das Problem ist NP-vollständig, so daß ein Approximationsalgorithmus gerechtfertigt ist. Ein einfacher randomisierter Algorithmus erzeugt einen Schnitt, der mindestens halb so groß wie der optimale ist:

1. Werfe für jeden Knoten v eine Münze. Bei Kopf wird v in die Menge C eingefügt, sonst nicht.
2. $E(C, V \setminus C)$ ist der gefundene Schnitt.

Für eine Kante $\{v, w\}$ ist die Wahrscheinlichkeit, daß sie im Schnitt enthalten ist $1/2$. Für den Erwartungswert der Kosten gilt dann, wegen der Linearität des Erwartungswertes,

$$\mathrm{E}[|E(C, V \setminus C)|] = \sum_{e \in E} 1/2 \geq 1/2 OPT(G)$$

Analog zu *MAX-SAT* können wir das Verfahren leicht derandomisieren.

Im Laufe der Jahre wurden für *MAX-CUT* immer wieder verbesserte Algorithmen entwickelt, die jedoch nicht maßgeblich die Schranke von 2 durchbrachen - bei großen Knotenzahlen näherte sich die Gütegarantie immer 2. Goemans und Williamson [GW94] entwickelten einen Algorithmus der eine Gütegarantie 1.1383 hat. Der Ansatz erreicht mit Hilfe der semidefiniten Programmierung sein Ziel und ist randomisiert. Er kann aber mit entsprechendem Aufwand derandomisiert werden. Wir betrachten ihn in Abschnitt 8.2.

Wir zitieren folgenden Satz.

Satz 8.1 ([PY91]) *Für alle $\rho > 0$ existiert eine* **GP**-*Reduktion mit Parametern $(1, \rho)$ und $(1, \rho/\epsilon)$ von MAX-3SAT auf MAX-CUT (zur Zeit $\epsilon \approx 1.014$).*

■

8.1 Grundlagen zur semidefiniten Programmierung

In der semidefiniten Programmierung werden lineare Fuktionen mit einer affin-linearen Kombination von positivsemidefiniten symmetrischen Matrizen als Nebenbedingung minimiert (maximiert). Allgemein sind semidefinite Programme Minimierungsprobleme der folgenden Form [VB94]:

$$\begin{aligned} &\min c^t x \\ \text{s.t.} \quad &z^t F(x) z \geq 0 \quad \forall z \in \mathbb{R}^n \end{aligned}$$

mit
$$F(x) = F_0 + \sum_{i=1}^{m} x_i F_i$$

Eingabe sind also die Zielfunktion $c \in \mathbb{R}^m$ und $m+1$ symmetrische positiv semidefinite Matrizen $F_0, ..., F_m \in \mathbb{R}^{n \times n}$. Dabei sind semidefinite Matrizen wie folgt definiert:

Definition 8.1 (positiv-(semi-)definite Matrizen)

Eine symmetrische Matrix $A \in \mathbb{R}^{n \times n}$ heißt positiv semidefinit (positiv definit), falls für alle $x \in \mathbb{R}^n \setminus \{0\}$ $x^t Ax \geq 0$ ($x^t Ax > 0$) gilt.

Erinnern wir uns an einige grundlegende Eigenschaften von positiv semidefiniten Matrizen.

Bemerkung 8.1 *Für eine symmetrische Matrix $A \in \mathbb{R}^{n \times n}$ sind folgende Eigenschaften äquivalent:*

1. *A ist positiv semidefinit.*
2. *Alle Eigenwerte von A sind nichtnegativ.*
3. *Es existiert eine untere Dreiecksmatrix B, die nur nichtnegative Diagonalelemente hat, so daß $A = BB^t$.*

Ein semidefinites Programm ist ein konvexes Optimierungsproblem, weil sowohl die Zielfunktion als auch die Nebenbedingungen konvex sind: Wenn $F(x)$ und $F(y)$ positivsemidefinit sind, dann gilt für alle $0 \leq \lambda \leq 1$

$$F(\lambda x + (1-\lambda)y) = \lambda F(x) + (1-\lambda)F(y) \text{ ist positivsemidefinit.}$$

Abbildung 8.1 veranschaulicht ein Beispiel für semidefinite Programmierung mit $x \in \mathbb{R}^2$ und $F_i \in \mathbb{R}^{7 \times 7}$.

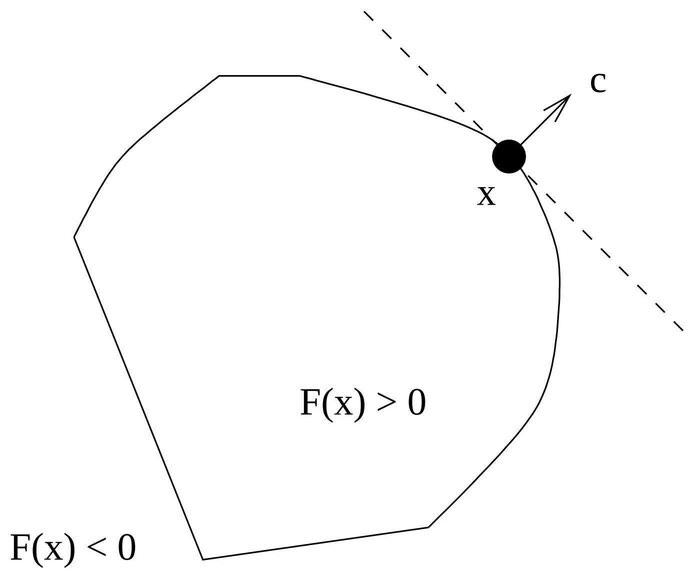

Abbildung 8.1: Beispiel für ein semidefinites Programm ($x \in \mathbb{R}^2$ und $F_i \in \mathbb{R}^{7\times 7}$).

Das geschlossene Polygon repräsentiert die Menge der zulässigen Punkte, d. h.

$$\{x \in \mathbb{R}^2 | F(x) \text{ ist positivsemidefinit. }\}$$

Anschaulich gesprochen ist hier das Ziel, eine Orthogonale auf c in c-Richtung möglichst weit zu bewegen, solange man in der zulässigen Menge bleibt. In unserem Fall ist der Optimalpunkt x_{opt} eindeutig, was im allgemeinen nicht notwendigerweise der Fall ist.

Die Definition von semidefiniter Programmierung erscheint auf dem ersten Blick sehr spezialisiert und konstruiert. Wir werden aber im Laufe dieses Kapitels sehen, daß sich viele numerische und kombinatorische Probleme als semidefinite Programme ausdrücken lassen. Als erstes Bespiel betrachten wir die lineare Programmierung (vgl. [S86]):

$$\min c^t x$$
$$\text{s.t.} \quad Ax + b \geq 0$$

Da ein Vektor $v \in I\!R^m$ genau dann komponentenweise nichtnegativ ist, wenn die symmetrische Matrix diag(v) positivsemidefinit ist, können wir das LP (mit $F(x) = \text{diag}(Ax + b)$) auch als semidefinites Programm formulieren, d. h.

$$F_0 = \text{diag}(b), \quad F_i = \text{diag}(a_i), \quad i = 1, ..., m,$$

wobei die a_i die Spaltenvektoren von A sind. Auf der anderen Seite kann semidefinite Programmierung auch als Erweiterung der linearen Programmierung betrachtet werden [A95].

Es stellte sich heraus, daß die semidefinite Programmierung wie die lineare Programmierung, sowohl theoretisch als auch praktisch effizient lösbar ist.

Theoretische Effizienz: Sie folgt aus der Konvexität. Somit ist es leicht möglich, einen Seperationsalgorithmus anzugeben und somit die (Ellipsoid-)Methode von Khachiyan anzuwenden (Grundlagen dazu sind in Kapitel 13 von [S86] zu finden). Dieser Algorithmus ist zwar polynomiell, aber kaum praktikabel. Alizadeh entwickelte [A95] polynomielle Interior-Point-Algorithmen, die ...

Praktische Effizienz: ... auch in der Praxis ein gutes Laufzeitverhalten haben und numerisch stabil sind.

Dies soll an dieser Stelle als Einführung zu semidefiniten Programmierung genügen. Für weitere Informationen sei an [S86, GLS88, A95, VB94] verwiesen.

8.1.1 Semidefinite Programme in der kombinatorischen Optimierung

In diesem Kapitel wollen wir kombinatorische Optimierungsprobleme mit Hilfe von semidefiniter Progammierung lösen. Der Ansatz scheint auf den ersten Blick ein wenig umständlich, da die semidefinite Programmierung ein numerischer Optimierungsansatz ist, der scheinbar nichts mit Kombinatorik gemeinsam hat.

Betrachte folgendes Standard-Semidefinites-Programm (SSDP):

$$
\begin{array}{lll}
(SSDP) & & \max \sum_{i,j=0}^{n} w_{i,j} x_{i,j} \\
 & \text{s.t.} & x_{i,j} \in P \text{ mit } P \text{ einem Polyeder. } (x_{i,j} \text{ erfüllt lineare Restriktionen}) \\
 & & X \text{ ist positiv semidefinit}
\end{array}
$$

Diese Definition entspricht der einführenden Beschreibung von semidefiniten Programmen.

Die Methode zum Approximieren unseres Problems vollzieht sich im wesentlichen in 3 Schritten:

1. Wir beschreiben das Problem als ganzzahliges numerisches Problem. Eine Standardmethode, die auf eine Vielzahl kombinatorischer Probleme anwendbar ist, ist die Beschreibung durch ein 0/1-Integer-LP. Als Beispiel betrachten wir hier Vertex-Cover:

$$
\begin{array}{lll}
(IP) & & \min \sum_{v \in V} c_v \\
 & \text{s.t.} & Ac \geq 1 \quad c \in \{0,1\}^V
\end{array}
$$

   mit $A \in \{0,1\}^{E \times V}$ die Inzidenzmatrix von A.

2. Da die 0/1-Integer-Programming NP-vollständig ist, haben wir allein durch die Formulierung (IP) noch keinen Gewinn. Wir können jedoch (im Fall von Vertex-Cover) folgende Relaxierung (RP) in polynomieller Zeit lösen:

$$
\begin{array}{lll}
(RP) & & \min \sum_{v \in V} c_v \\
 & \text{s.t.} & Ac \geq 1 \quad 0 \leq c_v \leq 1 \quad \forall v \in V
\end{array}
$$

3. Für die Lösung des kombinatorischen Problems brauchen wir in der Regel diskrete Werte, in unserem Fall also eine 0/1-Lösung. Die Lösung in Schritt 2 ist aber möglicherweise ein Vektor, dessen Komponeneten Werte annehmen, die echt zwischen 0 und 1 liegen. Wir müssen nun diese Werte geeignet auf 0 oder 1 runden, um eine Lösung für unser kombinatorisches Problem zu erhalten.

Der Ansatz in diesem Kapitel variiert ein wenig von dem für Vertex-Cover: Wir relaxieren unser Problem als $(SSDP)$ und erhalten als seine Lösung eine Matrix $X = (x_{i,j})_{i,j}$, mit der wir allein noch nichts anfangen können. Ziel wird es sein, jeder Lösungskomponente (z.B. einem Knoten) einen Vektor zuzuordnen (im Gegensatz zu unserem Beispiel, wo wir jedem Knoten eine Komponente eines Vektors zugeordnet haben). Mit der entsprechenden Formulierung des $(SSDP)$ erzwingen wir, daß diese Vektoren gewisse Eigenschaften besitzen, so daß wir im folgenden Rundungsschritt den Knoten diskrete Werte zuordnen können.

Ziel dieses Abschnittes ist es nun, aus der Matrix X effizient die benötigten Vektoren zu konstruieren.

Behauptung 8.1 *Gegeben eine Lsg. X von (SSDP) können wir in polynomieller Zeit eine Menge $\{v_1, ..., v_n\} \subseteq I\!R^n$ finden, so daß*

$$v_i^t v_j = x_{i,j} \quad \forall i, j = 1, ..., n.$$

Beweis: Gegeben eine positiv semidefinite Matrix $X = (x_{i,j})_{i,j}$, können wir mit Hilfe der Cholesky-Zerlegung eine $n \times n$-Matrix B konstruieren, die Bedingung 3 in Bemerkung 8.1 erfüllt.

Die Zerlegung von X: Betrachte folgendes Schema zur Berechnung von $X = (\sum_{k=1}^{n} b_{k,i} b_{k,j})_{i,j}$

$$
\begin{array}{c|c}
 & \begin{pmatrix} b_{1,1} & & & \\ 0 & \ddots & B & \\ \vdots & \ddots & \ddots & \\ 0 & \cdots & 0 & b_{n,n} \end{pmatrix} \\ \hline
\begin{pmatrix} b_{1,1} & 0 & \cdots & 0 \\ & \ddots & \ddots & \vdots \\ & B & \ddots & 0 \\ & & & b_{n,n} \end{pmatrix} & \begin{pmatrix} x_{1,1} & \cdots & \cdots & x_{1,n} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{n,1} & \cdots & \cdots & x_{n,n} \end{pmatrix}
\end{array}
$$

Es ergeben sich folgende Identitäten:

- $b_{1,1} = \sqrt{x_{1,1}}$
- $x_{i,1} = b_{1,1} b_{i,1} \Leftrightarrow b_{i,1} = \frac{x_{i,1}}{b_{1,1}}$
- Für alle $j = 2, ..., n$ ist

$$x_{j,j} = \sum_{k=1}^{j} b_{j,k}^2 \Leftrightarrow b_{j,j} = \sqrt{x_{j,j} - \sum_{k=1}^{j-1} b_{j,k}^2}$$

- Für alle $j = i+1, ..., n$ ist

$$\begin{aligned} x_{j,i} &= \sum_{k=1}^{n} b_{j,k} b_{i,k} \\ &= \sum_{k=1}^{i} b_{j,k} b_{i,k} \\ &= \sum_{k=1}^{i-1} b_{j,k} b_{i,k} + b_{j,i} b_{i,i} \\ &\Leftrightarrow \\ b_{j,i} &= \frac{1}{b_{i,i}} \left(x_{j,i} - \sum_{k=1}^{i-1} b_{j,k} b_{i,k} \right) \end{aligned}$$

-

$$x_{n,n} = \sum_{i=1}^{n} b_{i,n}^2 \Leftrightarrow b_{n,n} = \sqrt{x_{n,n} - \sum_{k=1}^{n-1} b_{j,k}^2}$$

Es genügt nun, die Werte in der richtigen Reihenfolge zu berechnen, was der Algorithmus 8.1 in $O(n^3)$ Schritten macht:

Algorithmus 8.1 (Cholesky-Dekomposition)

Eingabe : Eine positiv semidefinite Matrix X

Ausgabe : Eine untere Dreiecksmatrix B mit der Eigenschaft, daß $X = B^t B$

Schritt 1: Berechne $b_{1,1}$.

Schritt 2: Berechne für alle $i = 2, ..., n$, $b_{i,1}$.

Schritt 3: Für alle $i = 2, ..., n-1$

- Berechne $b_{i,i}$.
- Berechne für alle $j = i+1, ..., n$, $b_{j,i}$.

Schritt 4: Berechne $b_{n,n}$.

Wir erhalten $\{v_1, ..., v_n\}$ als die Menge der Spaltenvektoren von B^t. ■

8.2 Semidefinite Programme zur Approximation von MAX-CUT

Wir wollen zunächst MAX-CUT genau definieren:

Problem 8.1 (MAX-CUT)

Input: Ein Graph $G = (V, E)$ mit $V = \{1, ..., n\}$ und eine Gewichtsmatrix $w : V^2 \rightarrow I\!R_0^+$ mit

- $w(i, j) = 0$, wenn $\{i, j\} \notin E$
- $w(i, j) = w(j, i)$ für alle i, j
- $w(i, i) = 0$ für alle i

Task: Finde $V' \subseteq V$, so daß

$$w(V', V \setminus V') := \sum_{i \in V'} \sum_{j \in V \setminus V'} w(i, j)$$

maximal ist. Der Maximalwert wird mit $MAX\text{-}CUT^*(G)$ bezeichnet.

Zunächst werden wir zwei Lemmata beweisen, die bei der Analyse des randomisierten Rundens (siehe oben Punkt 3) hilfreich sind.

Definition 8.2 $S_n = \{x \in I\!R^n : \|x\| = 1.\}$

Lemma 8.1 *Sei $v, w \in S_n$. Dann gilt*

$$\Pr_{r \in_R S_n}[\mathrm{sgn}(v^t r) \neq \mathrm{sgn}(w^t r)] = \frac{\arccos(v^t w)}{\pi}$$

Beweis: Es gilt (Symmetrie)

$$\Pr_{r \in_R S_n}[\mathrm{sgn}(v^t r) \neq \mathrm{sgn}(w^t r)] = 2 \Pr_{r \in_R S_n}[v^t r \geq 0 \wedge w^t r < 0]$$

Setze $\phi := \arccos(v^t w)$. Die Mengen $R = \{r \in S_n | v^t r \geq 0 \wedge w^t r < 0\}$ und $S_n \setminus R$ sind die Schnittmenge von S_n mit 2 Halbräumen, die zusammen einen Winkel von ϕ einschließen. Daraus folgt, daß das Maß von R $\phi/2\pi$ mal dem Maß von S_n ist. Also gilt

$$\begin{aligned} \Pr_{r \in_R S_n}[v^t r \geq 0 \wedge w^t r < 0] &= \phi/2\pi \\ &= \frac{\arccos(v^t w)}{2\pi} \end{aligned}$$

■

Lemma 8.2 *Für alle $-1 \leq y \leq 1$ gilt*

$$\frac{\arccos(y)}{\pi} \geq \frac{\alpha}{2}(1-y)$$

mit

$$\alpha := \min_{0<\phi<\pi} \frac{2}{\pi} \frac{\phi}{1-\cos\phi} > 0.87856$$

Beweis: Setze $y := \cos(\phi)$. Es ist nun zu zeigen, daß

$$\frac{2\phi}{\pi(1-\cos(\phi))} > 0.87856$$

ist:

$$\begin{aligned} f(\phi) &= \frac{\partial \frac{2\phi}{\pi(1-\cos(\phi))}}{\partial \phi} \\ &= 2/\pi \left(\frac{1}{1-\cos(\phi)} - \frac{\phi \sin(\phi)}{(1-\cos(\phi))^2} \right) \\ f'(\phi) &> 0 \end{aligned}$$

Bestimme nun die Nullstellen von f:

$$f(\phi) = 0 \Leftrightarrow \cos\phi + \phi \sin\phi = 1$$

■

Gemäß unserem Schema werden wir MAX-CUT als ganzzahliges numerisches Problem beschreiben. Sei dazu eine Instanz (G, w) von MAX-CUT gegeben. Das folgende quadratische Programm

$$\begin{array}{lrl} (Q) & & \max \frac{1}{2} \sum_{i<j} w_{i,j}(1 - v_i^t v_j) \\ & \text{s.t.} & v_i \in S_1 = \{-1, 1\} \quad \forall i \in V \end{array}$$

ist äquivalent zu MAX-CUT, wenn wir als V' die Menge der Knoten i nehmen, für die gilt daß $v_i = -1$ ist. Wir relaxieren nun (Q) indem wir nicht mehr in S_1, sondern in S_n rechnen und somit folgendes Programm (P) erhalten:

$$\begin{array}{lrl} (P) & & \max \frac{1}{2} \sum_{i<j} w_{i,j}(1 - v_i^t v_j) \\ & \text{s.t.} & v_i \in S_n \quad \forall i \in V \end{array}$$

Nehmen wir nun an, daß wir (P) lösen können. Beachte, daß $\max(P) \geq \max(Q)$. Wir benötigen eine Methode, um aus den gewonnenen n-dimensionalen Vektoren v_i Werte aus $\{-1, 1\}$ zu berechnen, die einen *guten* Schnitt in G definieren. Betrachte folgenden Algorithmus:

Algorithmus 8.2 (Semidefinite-MAX-CUT)

Eingabe : Eine Instanz (G, w) von MAX-CUT

Ausgabe : Ein Teilmenge $V' \subseteq V$, die einen Schnitt definiert

Schritt 1: Löse (P) mit Ergebnis $\{v_1, ..., v_n\}$

Schritt 2: Wähle $r \in_R S_n$

Schritt 3: Setze $V' = \{i | v_i^t r \geq 0\}$

Wir wählen eine zufällige Hyperebene und teilen damit den Lösungsraum in 2 Halbräume auf. Die Aufteilung der Knotenmenge wird gemäß der Aufteilung der korrespondierenen Vektoren durch die Hyperebenen gewählt. Der folgende Satz liefert uns eine Gütegarantie für den Algorithmus 8.2:

Satz 8.2 ([GW94]) *Sei $W := w(V', V \setminus V')$ die durch Algorithmus 8.2 definierte Zufallsvariable. Dann gilt*

$$\begin{aligned} \mathrm{E}[W] &\geq \alpha/2 \sum_{i<j} w_{i,j}(1 - v_i^t v_j) \qquad (8.1) \\ &\geq \alpha w MAX\text{-}CUT^*(G) \end{aligned}$$

mit

$$\alpha := \min_{0<\phi<\pi} \frac{2}{\pi} \frac{\phi}{1 - \cos\phi} > 0.87856.$$

Beweis: Aus der Linearität des Erwartungswertes folgt

$$\begin{aligned} \mathrm{E}[W] &= \sum_{i<j} w_{i,j} \Pr_{r \in S_n}[\mathrm{sgn}(v_i^t r) \neq \mathrm{sgn}(v_j^t r)] \\ \text{(Lemma 8.1)} \quad &= \sum_{i<j} w_{i,j} \frac{\arccos(v_i^t v_j)}{\pi} \\ \text{(Lemma 8.2)} \quad &\geq \frac{\alpha}{2} \sum_{i<j} w_{i,j}(1 - v_i^t v_j). \end{aligned}$$

■

Es ist ist leicht zu sehen, daß man (P) als semidefinites Programm ausdrücken kann: Wenn wir eine positiv semidefinite Matrix $X = B^t B$ haben, dann bedeutet die Restriktion $x_{i,i} = 1$, daß die Spaltenvektoren von B aus S_n sind, und somit ist (mit Beh. 8.1) (P)

äquivalent zu folgendem semidefiniten Programm (P'):

$$
\begin{array}{lrl}
(P') & & \max \frac{1}{2} \sum_{i<j} w_{i,j}(1 - x_{i,j}) \\
 & \text{s.t.} & x_{i,i} = 1 \text{ für alle } i \in V \\
 & & X \text{ ist positiv definit}
\end{array}
$$

Satz 8.3 ([GW94]) *Der Algorithmus 8.2 kann derart implementiert werden, daß für alle* ϵ *ein Schnitt mit erwartetem Gewicht*

$$\mathrm{E}[W] \geq (\alpha - \epsilon) w MAX\text{-}CUT^*(G)$$

gefunden wird. Die Laufzeit ist polynomiell in n *und* $\log(1/\epsilon)$.

Beweis: In Schritt 1 erhalten wir mit den in Abschnitt 8.1 beschriebenen Methoden eine relaxierte Lösung $\{v_1, ..., v_n\}$ mit Wert $\bar{w}$, für die

$$\bar{w} \geq \alpha w MAX\text{-}CUT^*(G) - \epsilon \geq (\alpha - \epsilon) w MAX\text{-}CUT^*(G)$$

gilt. Wie im Beweis von Satz 8.2 erhalten wir die angegebene Schranke. Die Laufzeit von Schritt 1 folgt aus den Ausführungen des Abschnitts 8.1. ∎

Wir haben an einem ersten Beispiel gesehen, daß man mit semidefiniter Programmierung maßgeblich verbesserte Ergebnisse erreichen kann. Vor kurzem wurde von Feige, Karpinski und Langberg [FKL00] verbesserte Approximationsalgorithmen, die auf semidefiniter Programmierung basieren, für breschränkten Grad MAX-CUT Probleme konstruiert. Die korrespondierenden A.R.'s für MAX-CUT für Graphen von maximalen Grad 3 und MAX-CUT für 3-regulären Graphen sind entsprechend 1.0858 und 1.0823. Beide Probleme sind $MAX\text{-}SNP$-hart. Die erste explizite untere Schranke für die A.R. für das letztere Problem wurde in der Arbeit von Berman und Karpinski [BK98] konstruiert.

Kapitel 9

Explizite Reduktionen

In diesem Kapitel werden wir explizite **GP**-Reduktionen kennenlernen. Zur Vereinfachung betrachten wir hier häufig *normierte* Probleme. Als Besipiel betrachten wir dazu das $MAX\text{-}SAT$ Problem.

Sei f eine Formel in konjunktiver Normalform

$$f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$$

$MAX\text{-}SAT(f)$ war definiert als

$$MAX\text{-}SAT(f) = \max_{x \in \{0,1\}^n} |\{i | c_i(x) = 1\}|.$$

Wir führen nun die *normierte* Variante $\|MAX\text{-}SAT(f)\|$ als

$$\|MAX\text{-}SAT(f)\| = \frac{MAX\text{-}SAT(f)}{m}$$

ein. Man beachte, daß $0 \leq \|MAX\text{-}SAT(f)\| \leq 1$ gilt.

9.1 Expander Graphen

Wir werden in diesem Abschnitt eine sehr nützliche Klasse von Graphen kennenlernen, die sogenannten *Expander Graphen*.

Definition 9.1 *Ein Graph $G = (V, E)$ wird (n, d, c)-Expander genannt, wenn*

- *er n Knoten hat,*
- *der Maximalgrad eines Knoten d ist und*
- *für alle Teilmengen $W \subseteq V$ mit $|W| \leq n/2$ gilt*

  $$|N(W)| \geq c|W|.$$

  $N(W)$ ist dabei die Nachbarschaft von W in $V \setminus W$.

Solche Graphen werden z.B. bei der Konstruktion von Sortiernetzwerken oder bei der Konstruktion von Graphen mit hohem Zusammenhang und geringen Grad benötigt. Sie sind von großer Bedeutung bei der Realisierung von Netzwerken für Parallelrechner. Die Beweis der Existenz von solchen Expandern ist nicht trivial aber auch nicht schwer einzusehen [M73]. Die Konstruktion von Expandern ist schwieriger. Höhepunkt einer Reihe von Arbeiten [M73], [GG81], [LPS86] ist folgendes Resultat. Ein Graph heißt *d-regulär* genau dann wenn $\forall v \in V$ $Grad(v) = d$.

Satz 9.1 ([LPS88]) *Es existiert ein n_0, so daß für alle Primzahlen $p \equiv 1 (\mathrm{mod}\ 4)$ eine Familie von $d = (p+1)$-regulären Graphen $(G_n)_{n \geq n_0}$ existiert, die $(n, p, c(p))$-Expander sind. Wir können diese Graphen in Zeit $n^{O(1)}$ konstruieren.*

Wir werden diese Expander Graphen für die Konstruktion von Reduktionen benötigen.

Korollar 9.1 ([LPS88]) *Es existiert ein n_0, so daß wir für alle $n \geq n_0$ in Zeit $n^{O(1)}$ einen 14-regulären Graph G_n auf $n(1 + o(1))$ Knoten erzeugen können, so daß für alle $S \subseteq V(G)$ die Anzahl der Kanten zwischen S und seinem Komplement $\bar{S}$ größer als $\min\{|S|, |\bar{S}|\}$ ist.*

9.2 Die Reduktion von $\|MAX\text{-}3SAT\| \leq_{\mathbf{GP}} \|5\text{-}Occ\text{-}3SAT\|$

Lemma 9.1 *Für alle $\delta > 0$ existiert eine* **GP***-Reduktion von $\|MAX\text{-}3SAT\|$ auf $\|29\text{-}Occ\text{-}3SAT\|$ mit Parametern $(1,(1-\delta)^{-1})$ und $(1,(1-\delta/43)^{-1})$.*

Beweis: Sei I eine Instanz von $MAX\text{-}3SAT$ mit n Variablen $y_1, ..., y_n$ und m Klauseln. Wir definieren m_i als die Anzahl der Klauseln, in denen y_i vorkommt. N definieren wir als

$$N = \sum_i m_i$$

Weil wir eine 3SAT-Formel haben, gilt $N \leq 3m$. Sei im Folgenden n_0 die Konstante die wir in Kapitel 9.1 bei der Konstruktion von Expander Graphen definiert haben. O.B.d.A. ist jedes $m_i > n_0$, da die n_0-fache Replikation jeder einzelnen Klausel nichts am Anteil der erfüllten Klauseln ändert.

Wir konstruieren nun eine Instanz $\tau(I)$ von $29\text{-}Occ\text{-}3SAT$.

- Ersetze y_i durch m_i Kopien $y_i^1, ..., y_i^{m_i}$, indem wir für das j-te Auftreten y_i^j benutzen.
- Konstruiere den Expander Graphen G_{m_i}.
- Für jede Variable y_i und jede $j, l \leq m_i$ mit $\{j, l\} \in E(G_{m_i})$ konstruiere neue Klauseln

  $$(y_i^l \vee \neg y_i^j), \quad (y_i^j \vee \neg y_i^l)$$

  $$|E(G_{m_i})| = 14 \cdot m_i/2 = 7 \cdot m_i.$$

  Es gibt $14 \cdot m_i$ neue Klauseln.

Diese Formel hat somit $14N$ neue Klauseln und m alte Klauseln und jede Variable taucht in genau 28 neuen und einer alten Klausel auf.

Eine optimale Belegung von $\tau(I)$ erfüllt alle neuen Klauseln. Angenommen es wird eine neue Klausel von y_i nicht erfüllt. Dann haben die Variablen $y_i^1, ..., y_i^{m_i}$ nicht alle den gleichen Wert.

Betrachte diese Variablen als Knoten von G_{m_i}. Die Werte dieser Variablen definiert uns eine Partition S, $\bar{S}$ der Knotenmenge. O.B.d.A. ist $|S| \leq m_i/2$. Aus dem Korollar 9.1 folgt, daß mindestens $|S| + 1$ Kanten aus S herausgehen. Jede dieser Kanten entspricht einer nicht erfüllten neuen Klausel.

Drehe die Werte für die Variablen in S herum. Wir erfüllen nun $|S| + 1$ neue Klauseln mehr und höchstens $|S|$ alte Klauseln weniger als vorher. Das ist ein Widerspruch zur Optimalität. Also erfüllt eine optimale Belegung von $\tau(I)$ alle neuen Klauseln. Somit haben wir

$$\|MAX\text{-}3SAT(I)\| = 1 \Longrightarrow \|29\text{-}Occ\text{-}3SAT(\tau(I))\| = 1$$

Nehmen wir nun an, daß keine Belegung mehr als $(1-\delta)m$ Klauseln von I erfüllen kann. Dann können in der neuen Formel niemals mehr als $14N + (1-\delta)m$ Klauseln erfüllt werden. Der Anteil der nicht erfüllten Klauseln ist also

$$\begin{aligned} 1 - \frac{14N + (1-\delta)m}{14N + m} &= \frac{\delta m}{14N + m} \\ (N \leq 3m) \qquad &\geq \frac{\delta m}{42m + m} \\ &= \delta/43 \end{aligned}$$

Daraus folgt

$$\|MAX\text{-}3SAT(I)\| < (1-\delta) \Longrightarrow \|29\text{-}Occ\text{-}3SAT(\tau(I))\| < 1 - \delta/43$$

■

Satz 9.2 *Für alle $\delta > 0$ existiert eine* **GP**-*Reduktion von $\|29\text{-}Occ\text{-}3SAT\|$ auf $\|5\text{-}Occ\text{-}3SAT\|$ mit Parametern $(1, (1-\delta)^{-1})$ und $(1, (1-\delta/29)^{-1})$.*

BEWEIS: Ersetze den Expander der letzten Konstruktion durch einen Kreis: Wenn eine Variable l-mal auftaucht, dann ersetze sie durch l neue Variablen und füge die neuen Klauseln entsprechend dem Kreis auf l Knoten ein. Jede Variable erscheint nun in 4 neuen und einer alten Klausel. Wenn nun ein $1 - \delta$-Anteil der alten Formel erfüllt wird, dann wird höchstens ein $1 - \delta/29$-Anteil der neuen Formel erfüllt. ■

9.3 Die Reduktion von $\|5\text{-}Occ\text{-}3SAT\| \leq_{\mathbf{GP}} LABEL\ COVER$

Wir definieren $LABEL\ COVER$ wie folgt. Eingabe ist

1. Ein regulärer[1] bipartiter Graph $G = (V_1, V_2, E)$.
2. Eine natürliche Zahl N, die die Menge $\{1, ..., N\}$ der *Labels* definiert.
3. Für jede Kante $e \in E$ eine partielle Funktion $\Pi_e : \{1, ..., N\} \to \{1, ..., N\}$.

Ein *Labelling* ist durch eine nicht-leere Menge von Labels für jeden Knoten von G definiert. Wir sagen ein Labelling *überdeckt* eine Kante $e = \{u, v\}$ (mit $u \in V_1$ und $v \in V_2$), wenn für jedes Label a_2 von v ein Label a_1 von u existiert, so daß

$$\Pi_e(a_1) = a_2$$

Aufgabe ist es, ein Labelling zu finden, das ein Label pro Knoten vergibt und den Anteil der überdeckten Kanten maximiert.

Satz 9.3 *Für alle $\epsilon > 0$ existiert eine* **GP***-Reduktion von $\|5\text{-}Occ\text{-}3SAT\|$ auf $LABEL\ COVER$ mit Parametern $(1, (1-\epsilon)^{-1})$ und $(1, (1-\epsilon/3)^{-1})$.*

BEWEIS: Gegeben eine Instanz I von $5\text{-}Occ\text{-}3SAT$ konstruieren wir wie folgt eine Instanz $G = (V_1, V_2, E)$ von $LABEL\ COVER$.

- In V_1 haben wir für jede Klausel und
- in V_2 für jede Variable einen Knoten.
- Wir haben eine Kante, wenn die entsprechende Variable in der jeweiligen Klausel vorkommt.

[1] Ein bipartiter Graph wird regulär genannt, wenn zwei Konstanten d_1, d_2 existieren, so daß der Grad der Knoten auf der linken Seite d_1 und auf der rechten Seite d_2 ist.

- Die Anzahl N der Labels ist 8.
- Wir betrachten für einen Knoten in V_1, in dessen Klausel die Variablen x_i, x_j, x_k vorkommen, das Label als einen binären Vektor (b_1, b_2, b_3) bezüglich der Belegung $x_i = b_1, x_j = b_2, x_k = b_3$.
- Für die Knoten aus V_2 haben wir nur die Labels 0 oder 1 (Alle anderen 6 Labels sind Dummy-Labels).

Nun beschreiben wir Π_e. Sei $e = \{u, v\} \in E$, wobei u der Klausel C und v der Variable x_i entspricht. Sei nun j die Stelle an der x_i in C auftritt. Wir definieren

$$\Pi_e(b_1, b_2, b_3) = b_j \iff b_1, b_2, b_3 \text{ erfüllt } C.$$

- Für jeden Knoten dürfen wir nur ein Label verwenden. Also ist das Labelling der Knoten in V_2 eine Wahrheitsbelegung der Variablen.
- Das Labelling der Knoten in V_1 definiert uns eine Belegung der Variablen in den entsprechenden Klauseln (die die Klauseln erfüllen).
- Eine Kante wird überdeckt, wenn die Belegung der Variable auf beiden Seiten übereinstimmt und die Klausel erfüllt wird.

Wenn alle Kanten überdeckt werden, haben wir eine erfüllende Belegung. Wenn keine Belegung mehr als einen $(1-\epsilon)$-Anteil der Klauseln erfüllen kann, dann existiert auch kein Labelling, das mehr als einen $(1-\epsilon/3)$-Anteil der Kanten überdeckt. ■

9.4 Die Reduktion von *LABEL COVER* $\leq_{\mathbf{GP}}$ *MAX-CLIQUE*

Satz 9.4 *Für alle $c, \rho > 0$ existiert eine* **GP**-*Reduktion mit Parametern $(c/|E|, \rho)$ und (c, ρ) von LABEL COVER auf MAX-CLIQUE .*

Beweis: Sei $G = (V_1, V_2, E), N, \Pi$ eine Instanz von *LABEL COVER*. Wir konstruieren eine Instanz $G' = (V', E')$ von *MAX-CLIQUE* wie folgt. Für eine Kante e und Labels a_1 und a_2, mit $\Pi_e(a_1) = a_2$ nennen wir das Tripel (e, a_1, a_2) ein *Labelling Szenario*. Für $e = \{u, v\}$ interpretieren wir das Labelling Szenario als Zuordnung von a_1 zu u und a_2 zu v. Zwei Szenarien werden *inkonsistent* genannt, wenn die zugehörigen Kanten einen Knoten gemeinsam haben, dem aber verschiedene Labels zugeordnet wurden.

- V' ist die Menge der *Labelling Szenarien*.
- E': Zwei Szenarien sind adjazent, wenn sie nicht inkonsistent sind.

Wir zeigen nun, daß die Größe der größten Clique in G' gleich der maximalen Anzahl der überdeckten Kanten in unserer Instanz von *LABEL COVER* ist.

Für jedes Labelling, daß K Kanten überdeckt, bilden die zugehörigen Szenarien eine Clique der Größe K, weil sie alle konsistent sind.

Sei nun eine Clique $S \subseteq V'$ der Größe K gegeben. Kein Paar von Szenarien in S kann inkonsistent sein. D.h. für jeden beliebigen Knoten in $V_1 \cup V_2$ existieren keine 2 Szenarien in S, die verschiedene Label zu diesem Knoten zuordnen. Nun benutzten wir diese Labels (wenn sie existieren) und erhalten ein (partielles) Labelling, das jede Kante, die in S betrachtet wird überdeckt. Trivialerweise kann dieses partielle Labelling zu einem vollständigen Labelling erweitert werden. ■

Kapitel 10

Die NP-Härte von Approximationsproblemen

Rufen wir uns noch einmal die Notationen aus Kapitel 1.3 in Erinnerung. Ein *Max(Min)-Problem* war eine Menge P von Instanzen $I = (F, c)$. F war dabei die Menge der zulässigen Lösungen und $c : F \to I\!R$ eine Kostenfunktion, die uns für jede zulässige Lösung $s \in F$ die Kosten $c(s)$ angibt.

Wir sagen, daß ein Optimierungsproblem X *(approximativ) NP-hart* für den *A.R.* α (bzgl. deterministischer [randomisierter] Reduzierbarkeit) ist, wenn die Existenz eines polynomielle [randomisierten] Approximationsalgorithmus für X mit *A.R.* α impliziert, daß $P = NP$ [$RP = NP$].

Nehmen wir an wir haben einen Approximationsalgorithmus $\mathcal{A}$ für P. Für eine Instanz $x \in P$ definieren wir $\tilde{y} = c(\mathcal{A}(x))$. $\tilde{y}$ sind also die Kosten der Lösung, die uns $\mathcal{A}$ erzeugt. Wir verlangen, daß

$$\alpha = \frac{1}{1-\delta} \text{ für } \delta > 0,$$

die *Approximationsgüte* von $\mathcal{A}$ ist. Aus Kapitel 1.3 wissen wir, daß für Maximierungsprobleme

$$\frac{|\overbrace{c(\mathcal{A}(x))}^{=\tilde{y}} - OPT(x)|}{\max\{OPT(x), c(\mathcal{A}(x))\}} \leq \delta$$

gilt. Das heißt, $\tilde{y} \geq \frac{OPT(x)}{\alpha}$. Wir sagen auch, $\mathcal{A}$ approximiert P mit der Approximationsgüte (mit $A.R.$) $\alpha = \alpha(n)$, wenn $\tilde{y} \geq \frac{OPT(x)}{\alpha}$.

Wie können wir nun beweisen, daß ein Maximierungsproblem P_2 approximationshart ist? Nehmen wir an, daß eine polynomielle Reduktion τ von SAT auf ein Maximierungsproblem P_1 existiert, so daß für jede Formel g gilt

$$\begin{aligned} g \in SAT \quad &\Rightarrow \quad OPT(\tau(g)) \geq l_1 \\ g \notin SAT \quad &\Rightarrow \quad OPT(\tau(g)) < \frac{l_1}{\rho_1} \end{aligned}$$

Nehmen wir nun eine **GP**-Reduktion ϕ (aus Definition 1.8) mit Parametern (l_1, ρ_1) und (l_2, ρ_2) und bilden die Komposition $\phi \circ \tau$ (die trivialerweise auch wieder in polynomieller Zeit berechenbar ist). $\phi \circ \tau$ ergibt nun eine Reduktion von SAT auf P_2, die uns folgende Eigenschaften sichert:

$$\begin{aligned} g \in SAT \quad &\Rightarrow \quad OPT(\phi(\tau(g))) \geq l_2 \\ g \notin SAT \quad &\Rightarrow \quad OPT(\phi(\tau(g))) < \frac{l_2}{\rho_2} \end{aligned}$$

Mit anderen Worten zeigt uns $\phi \circ \tau$, daß ein $A.R.$ ρ_2 für P_2 NP-hard ist. Denn:

$$g \in SAT \equiv c(\phi(\tau(g))) \geq \frac{l_2}{\rho_2}.$$

Analog definieren wir die **GP**-Reduktion für *Minimierungsprobleme*.

10.1 Ein Beispiel für Promiseprobleme: das Unique Clique Problem

Das Unique Clique Problem gehört zu den sogenannten *Promiseproblemen*. Ein Problem ist ein Promiseproblem, wenn neben der Eingabe irgendein "Versprechen" mit der Eingabe assoziiert ist. Wir sagen ein Algorithmus löst ein Promiseproblem, wenn er zu jeder Eingabe, die die versprochene Bedingung erfüllt, eine richtige Antwort ausgibt. Bei Eingaben, die diese Bedingung nicht erfüllen ist es jedoch möglich, daß der Algorithmus nicht korrekt antwortet. Natürlich ist das Versprechen nur von Bedeutung, wenn die versprochene Bedingung nicht mit den gleichen Ressourcen, wie sie der Algorithmus verwendet, verifiziert werden kann.

Nun kommen wir zur Definition eines Promiseproblems, und zwar des Unique Clique Problems.

Definition 10.1 *Das Unique Clique Problem ist ein Promiseproblem und wie folgt definiert:*

Eingabe: *Ein ungerichteter Graph* $G = (V, E)$ *und eine Zahl* $k \in I\!N$.

Versprechen: G *enthält höchstens eine Clique der Größe* k.

Frage: *Enthält* G *eine Clique der Größe* k?

Erinnern wir uns an den Begriff der üblichen polynomiellen Reduzierbarkeit, der in der Literatur benutzt wird.

Seien Σ *und* Γ *zwei endliche Alphabete. Eine Sprache* $A \subseteq \Sigma^*$ *heißt polynomzeit reduzierbar auf eine Sprache* $B \subseteq \Gamma^*$, *in Zeichen* $A \leq_P B$, *falls es eine totale Funktion* $f : \Sigma^* \to \Gamma^*$ *mit polynomieller (det.) Zeitkomplexität gibt, so daß für alle* $a \in \Sigma^*$ *gilt:*

$$a \in A \equiv f(a) \in B.$$

Offensichtlich ist $\leq_P$ eine transitive Relation, d.h. aus $A \leq_P B$ und $B \leq_P C$ folgt $A \leq_P C$.

Analog hierzu läßt sich auch die randomisierte Reduzierbarkeit definieren. So ist A randomisiert (Polynomzeit) reduzierbar auf B ($A \leq_{RP} B$), falls

$$\exists \text{ RTM M mit } \phi_M : \Sigma^* \to \Gamma^*,\ T_M(n) = n^{O(1)}, \text{ so daß } \forall x \in \Sigma^*[x \in A \Leftrightarrow \phi_M(x) \in B]$$

Im folgenden Satz 10.1 werden wir zeigen, daß das Cliquenproblem (CP) randomisiert reduzierbar auf das Unique Clique Problem (UCP) ist. Da CP NP-vollständig ist, gilt damit das interessante Korollar

Korollar 10.1 *Wenn es zur Lösung des Unique Clique Problems einen randomisierten polynomiellen Algorithmus gibt, dann gibt es für jedes Problem in NP einen solchen Algorithmus.*

Satz 10.1 *Das Cliquenproblem ist randomisiert reduzierbar auf das Unique Clique Problem, d.h., es gilt $CP \leq_{RP} UCP$.*

Beweis: Wir definieren $MAX\text{-}CLIQUE\ (G) := \max\{|V'| : V' \text{ ist eine Clique in } G\}$ und konstruieren eine effiziente randomisierte Reduktion $f : G = (V, E) \mapsto G' = (V', E')$, die wie folgt definiert ist:

- Eingabe: Ein ungerichteter Graph $G = (V, E)$ mit $|V| = n$ und eine Zahl $k \in \mathbb{N}$.
- Ausgabe: Ein ungerichteter Graph $G' = (V', E')$ mit $|V'| = (nk)^{O(1)}$, mit folgenden Eigenschaften:
  - Falls $MAX\text{-}CLIQUE\ (G) < k$, dann gilt $MAX\text{-}CLIQUE\ (G') < 2nk^2$.
  - Falls $MAX\text{-}CLIQUE\ (G) = k$, dann gibt es mit hoher Wahrscheinlichkeit eine ganze Zahl r, $2nk^2 \leq r \leq 2nk(k+1)$, so daß G' genau eine Clique der Größe r besitzt, d.h.

    $$\Pr[\exists r \in \mathbb{N} \text{ mit } 2nk^2 \leq r \leq 2nk(k+1), \text{ so daß } [\#_c^r(G') = 1]] \geq 1/2.$$

    $\#_c^r(G')$ ist dabei als die Anzahl der Cliquen der Größe r in G' definiert.

Eine Reduktion mit diesen Eigenschaften beweist den Satz. Denn bei einer Eingabe G für das Cliquenproblem können wir die Reduktion für jedes mögliche k $(1 \leq k \leq n)$ durchführen, und somit den Graphen G' erzeugen. Dann können wir den Algorithmus für das Unique Clique

Problem auf G' anwenden, wobei r entweder zufällig und gleichverteilt aus $[2nk^2, 2nk(k+1)]$ gewählt wird, oder jedes r in diesem Bereich überprüft wird.

Die Reduktion läßt sich nun wie folgt bewerkstelligen. Wir konstruieren zunächst bei gegebenem $G = (V, E)$ und $k \in I\!N$ ein Gewichtssystem (V, F_k, w), wobei

$$F_k = \{W|W \text{ ist eine Clique der Größe k von } G\}$$

und w eine randomisierte Funktion $w : V \to \{1, \ldots, 2n\}$ ist. Aus dem Isolationslemma (Lemma 3.1) folgt dann, daß dieses Gewichtssystem mit hoher Wahrscheinlichkeit ($\geq 1/2$) eindeutig ist. Der Graph G' wird nun wie folgt konstruiert: Wir ersetzen jeden Knoten $v \in V$ durch eine Clique C_v der Größe $2nk + w(v)$ und verbinden für jede Kante $(u, v) \in E$ alle Knoten in C_u mit allen Knoten in C_v. D.h.,

$$V' = \bigcup_{v \in V} \{v_i : 1 \leq i \leq 2nk + w(v)\}$$

$$E' = \{\{v_i, v_j\} : v \in V\} \cup \{\{u_i, v_j\} : \{u, v\} \in E\}.$$

Falls $C = \{v_1, \ldots, v_t\} \subset V$ nun eine Clique in G ist, so ist $C' = C_{v_1} \cup C_{v_2} \cup \ldots \cup C_{v_t} \subset V'$ einen Clique in G' der Kardinalität $2nkt + w(v_1) + \ldots + w(v_{t_1})$. Natürlich entspricht auch jede (maximale) Clique in G' einer Clique (oder einem einzigen Knoten) in G.

Nachdem die Reduktion nun vollständig beschrieben worden ist, folgt der Beweis aus den folgenden zwei Behauptungen.

Behauptung 1: Falls $MAX\text{-}CLIQUE\ (G) < k$, dann ist $MAX\text{-}CLIQUE\ (G') < 2nk^2$.

Diese Behauptung ergibt sich aus der Tatsache, daß die Größe einer Clique $C' \subset V'$ die zu einer Clique $C = \{v_1, \ldots, v_{k-1}\} \subset V$ korrespondiert, höchstens

$$2nk(k-1) + w(v_1) + \ldots + w(v_{k-1}) \leq$$

$$2nk(k-1) + 2nk(k-1) = 2n(k+1)(k-1) < 2nk^2$$

ist.

Behauptung 2: Falls $MAX\text{-}CLIQUE\ (G) = k$, dann gilt

$$\Pr[\exists r \in I\!N \text{ mit } 2nk^2 \leq r \leq 2nk(k+1), \text{ so daß } [\#_c^r(G') = 1]] \geq 1/2.$$

Aus dem Isolationslemma (Lemma 3.1) folgt, daß die maximal gewichtete Clique (unter allen Cliquen der Größe k in G) eindeutig ist. Diese Clique sei mit C, und deren korrespondierende Clique in G' mit C' bezeichnet. C' ist eine Clique der Größe $2nk^2 + \bar{w}(C)$. Offensichtlich ist $2nk^2 < 2nk^2 + \bar{w}(C) < 2nk(k+1)$. Für jede andere Clique $\bar{C} \subset V, \bar{C} \neq C$, der Größe k hat die korrespondierende Clique in G' die Kardinalität $2nk^2 + \bar{w}(\bar{C}) < 2nk^2 + \bar{w}(C)$. Wegen Behauptung 1 ist für jede Clique in G der Größe $< k$ die korrespondierende Clique in G' kleiner als $2nk^2 + \bar{w}(C)$. Da es in G keine Cliquen der Größe $> k$ gibt, gibt es auch keine Clique größer als C' in G'. Deswegen ist C' die eindeutige Maximum Clique in G'. ■

10.2 Approximationshärte von $MAX\text{-}CLIQUE$

In diesem Abschnitt wollen wir zeigen, daß es NP-hart ist, eine Approximation an die größte Clique in einem gegebenen Graphen G zu finden. D.h. wir wollen folgenden Hauptsatz beweisen, der auf Arora, Lund, Motwani, Sudan und Szegedy [ALM$^+$92] zurückgeht.

Satz 10.2 (Hauptsatz) *Es gibt ein $\epsilon > 0$, so daß die Approximation von $MAX\text{-}CLIQUE$ mit A.R. n^ϵ NP-hart ist.*

Wir benötigen eine modifizierte Form von $MAX\text{-}SAT$: Sei f eine Formel in konjunktiver Normalform

$$f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$$

$MAX\text{-}SAT(f)$ war definiert als

$$MAX\text{-}SAT(f) = \max_{x \in \{0,1\}^n} |\{i | c_i(x) = 1\}|.$$

Wir führen nun die *normierte* Variante $\|MAX\text{-}SAT(f)\|$ als

$$\|MAX\text{-}SAT(f)\| = \frac{MAX\text{-}SAT(f)}{m}$$

ein. Man beachte, daß $0 \leq \|MAX\text{-}SAT(f)\| \leq 1$ gilt.

Zum Beweis unseres Hauptsatzes benutzen wir folgendes Lemma von [ALM$^+$92], daß wir in Kapitel 2.2 beweisen werden.

Lemma 10.1 *Wenn für MAX-$3SAT$ ein PTAS existiert, so folgt $P = NP$*

Wir benötigen aber noch weitere Vorüberlegungen.

Satz 10.3
Für alle $l, \epsilon > 0$ existiert eine GP-Reduktion von $\|MAX\text{-}3SAT\|$ auf $MAX\text{-}CLIQUE$ mit den Parametern $(l, 1+\epsilon)$ und $(l\frac{N}{3}, 1+\epsilon)$, wobei N polynomiell in n ist.

Beweis: Wir beweisen diese Aussage durch eine Modifikation der Reduktion von $3SAT$ auf $CLIQUE$ (siehe dazu auch [K72]). Sei g dazu eine Formel in konjunktiver Normalform mit genau 3 Literalen pro Klausel:

$$g(x_1, ..., x_n) = c_1 \wedge c_2 \wedge ... \wedge c_m.$$

Wir konstruieren nun eine **GP**-Reduktion $\phi : g \mapsto G = (V, E)$ auf $MAX\text{-}CLIQUE$:

$$\begin{aligned} V &= \{v_{i,j} | 1 \leq i \leq m, 1 \leq j \leq 3\} \\ E &= \{\{v_{i_1,j_1}, v_{i_2,j_2}\} | i_1 \neq i_2 \wedge c_{i_1,j_1} \neq \neg c_{i_2,j_2}\} \end{aligned}$$

Wenn nun eine Knotenteilmenge $C \subseteq V$ eine Clique von G ist, induziert diese eine konsistente Wahrheitsbelegung von $x_1, ..., x_n$, da jede Variable in höchstens einer Belegung in C vorkommt. Pro Klausel c_i existiert außerdem höchstens ein Knoten in C, der ein Literal repräsentiert, das c_i erfüllt. Andersherum erfüllt ein Knoten $v_{i,j} \in C$ auch die Klausel c_i. Somit sind $|C|$ Klauseln von g erfüllt. Es folgt also mit $N = 3m$,

$$\begin{aligned} \|MAX\text{-}3SAT(g)\| = l \quad &\Rightarrow \quad MAX\text{-}CLIQUE\ (G) = lN/3 \\ \|MAX\text{-}3SAT(g)\| < \frac{l}{1+\epsilon} \quad &\Rightarrow \quad MAX\text{-}CLIQUE\ (G) < \frac{lN}{3(1+\epsilon)} \end{aligned}$$

■

Fassen wir noch mal den Satz 10.3 zusammen, so erhalten wir folgende Aussage: Für alle $\epsilon' > 0$ existiert eine GP-Reduktion ϕ mit Parametern $(l, 1+\epsilon')$ und $(l\frac{N}{3}, 1+\epsilon')$, s.d.

$$\phi : g = \bigwedge_{i=1}^{m} c_i \mapsto G = (V, E) \text{ mit } |V| = N = 3m$$

Es gilt dabei ferner

$$\begin{aligned} \|MAX\text{-}3SAT(g)\| = l \quad &\Rightarrow \quad MAX\text{-}CLIQUE\ (G) = lm = lN/3 \\ \|MAX\text{-}3SAT(g)\| < \frac{l}{1+\epsilon'} \quad &\Rightarrow \quad MAX\text{-}CLIQUE\ (G) < \frac{lm}{1+\epsilon'} = lN/3\frac{1}{1+\epsilon'}. \end{aligned}$$

Für das Entscheidungsproblem $3SAT$ müssen wir $l = 1$ setzen. Wir fixieren nun ϵ, so daß $1 + \epsilon = \frac{1}{1+\epsilon'}$. Dann ist das folgende Entscheidungsproblem ob 1. oder 2. gilt, als Konsequenz von Lemma 10.1 NP-hart.

1. $MAX\text{-}CLIQUE\ (G) \geq N/3 \longrightarrow MAX - 3SAT(g) \geq m$
2. $MAX\text{-}CLIQUE\ (G) < N/(3(1+\epsilon)) \longrightarrow MAX - 3SAT(g) < m/1 + \epsilon$

Da wir ϵ beliebig wählen können, erhalten wir mit Lemma 10.1 eine schwächere Ausage als in unserem Hauptsatz:

Satz 10.4 *Es existiert ein $\epsilon > 0$, so daß die Approximation von $MAX\text{-}CLIQUE$ mit A.R. $(1 + \epsilon)$ NP-hart ist.*

Um diese Aussage zu verschärfen, brauchen wir eine Konstruktion, die die Größe der maximalen Clique in G in einem viel stärkeren Maß von der Erfüllbarkeit der $3SAT$ Formel g abhängig macht. Wir führen dazu die sogenannten (n, k, α)-*Booster* ein:

Definition 10.2 ((n,k,α)-Booster) *Sei $n \in I\!N$ gegeben. Ein (n,k,α)-Booster $\mathcal{B}$ ist eine Menge $\mathcal{B} \subseteq P_k(\{1,...,n\})$ von k-elementigen Teilmengen von $\{1,...,n\}$, so daß für alle $A \subseteq \{1,...,n\}$*

$$(\rho-\alpha)^k \leq \underbrace{\frac{|\{B|B \in \mathcal{B}, B \subseteq A\}|}{|\mathcal{B}|}}_{=r_{\mathcal{B}}} \leq (\rho+\alpha)^k$$

für $\rho = |A|/n$ gilt. $r_{\mathcal{B}}$ wird BOOSTER-RATIO von $\mathcal{B}$ genannt.

Als Beispiel betrachten wir $\mathcal{B} = P_k(\{1,...,n\})$. Für $\alpha \approx 0$ ist $\mathcal{B}$ ein (n,k,α)-Booster: Sei $A \subseteq \{1,...,n\}$ mit $|A| = \rho n$. Der BOOSTER-RATIO $r_{\mathcal{B}}$ ist dann $\frac{\binom{\rho n}{k}}{\binom{n}{k}} \approx \frac{(\rho n)^k}{n^k} = \rho^k$. Die Größe von $\mathcal{B}$ ist $\binom{n}{k} = \Theta(n^k)$. Somit folgt, daß wir nur für konstantes k diesen einfachen Booster ist polynomieller Zeit konstruieren können. Alon, Feige, Widgderson und Zuckerman [AFWZ95] zeigen, daß aber auch für $k = O(\log n)$ (n,k,α)-Booster in polynomieller Zeit konstruiert werden können:

Satz 10.5 (BOOSTER-SATZ [AFWZ95]) *Sei $k = O(\log n)$ und $\alpha > 0$. Dann kann man in polynomieller Zeit einen (n,k,α)-Booster $\mathcal{B}$ mit polynomieller Größe berechnen.*

Mit diesem Werkzeug, können wir unser Nichtapproximierbarkeitsresultat erheblich verbessern. Wir werden mit Hilfe von (n,k,α)-Boostern unseren Graphen G geeignet *aufblähen*. Setze $(1-\beta) = (1+\epsilon)$.

Definition 10.3 (BOOSTER-PRODUKT (BP) von Graphen) *Sei $G = (\{1,...,n\},E)$ ein Graph und $\mathcal{B}$ ein (n,k,α)-Booster. Das Booster-Produkt $G_{\mathcal{B}}$ ist dann der folgende Graph:*

$$G_{\mathcal{B}} = (\mathcal{B}, \{\{s_i,s_j\}|s_i,s_j \in \mathcal{B},\ s_i \cup s_j \text{ ist eine Clique in } G\})$$

Dann steht die Größe der maximalen Clique von G mit der Größe der maximalen Clique von $G_{\mathcal{B}}$ in folgendem Verhältnis.

Lemma 10.2 *Sei G ein Graph auf N Knoten und $G_{\mathcal{B}}$ das Booster-Produkt für einen (N,k,α)-Booster $\mathcal{B}$. Dann gilt*

$$\gamma_1 \leq \textit{MAX-CLIQUE}\ (G_{\mathcal{B}}) \leq \gamma_2$$

für

$$\begin{aligned}\gamma_1 &= \left(\frac{\textit{MAX-CLIQUE}\ (G)}{N} - \alpha\right)^k |\mathcal{B}| \\ \gamma_2 &= \left(\frac{\textit{MAX-CLIQUE}\ (G)}{N} + \alpha\right)^k |\mathcal{B}|\end{aligned}$$

Beweis: Sei $C \subseteq \{1,...,N\}$ eine maximale Clique der Größe t in G. Nach der Definition von (N,k,α)-Boostern, gilt für die Anzahl u der Mengen $X \in \mathcal{B}$, für die gilt, daß $X \subseteq C$,

$$(t/N-\alpha)^k|\mathcal{B}| \leq u \leq (t/N+\alpha)^k|\mathcal{B}|.$$

Nach der Definition von $G_{\mathcal{B}}$ bilden alle diese X eine Clique von $G_{\mathcal{B}}$. Wenn nun $B \subseteq \mathcal{B}$ eine maximale Clique von $G_{\mathcal{B}}$ ist, dann ist

$$C = \bigcup_{W \in B} W \text{ ist eine Clique mit höchstens } \textit{MAX-CLIQUE}\ (G) \text{ Knoten in } G$$

■

Nach Satz 10.5 können wir in polynomieller Zeit einen $(N, \log N, \beta/9)$-Booster $\mathcal{B}$ konstruieren. Also können wir auch in polynomieller Zeit den BP-Graphen $G_{\mathcal{B}} = (\mathcal{B}, E')$ konstruieren. Aus Lemma 10.2 folgt, daß

$$\gamma_1 \leq \textit{MAX-CLIQUE}\ (G_{\mathcal{B}}) \leq \gamma_2$$

mit

$$\begin{aligned}\gamma_1 &= \left(\frac{\textit{MAX-CLIQUE}\ (G)}{N} - \beta/9\right)^{\log N} |\mathcal{B}| \\ &\geq ((3-\beta)/9)^{\log N}|\mathcal{B}|\end{aligned}$$

und

$$\begin{aligned}\gamma_2 &= \left(\frac{\textit{MAX-CLIQUE}\ (G)}{N} + \beta/9\right)^{\log N} |\mathcal{B}| \\ &< ((3+2\beta)/9)^{\log N}|\mathcal{B}|.\end{aligned}$$

Da $|\mathcal{B}|$ polynomiell in m wächst, erhalten wir eine GAP von $|\mathcal{B}|^\epsilon$ für ein $\epsilon \approx 1/5$. ■

Kapitel 11

Der Beweis von $NP = PCP(\log n, 1)$

Dieses Kapitel beschreibt den Beweis des bahnbrechenden Resultats von [ALM$^+$92], daß

$$NP = PCP(\log n, 1).$$

Genauer gesagt, zeigen wir hier die nicht-triviale Inklusion $NP \subseteq PCP(\log n, 1)$. Der Beweis ist wie folgt aufgebaut:

1. Zunächst zeigen wir, daß $NP \subseteq PCP(n^{O(1)}, 1)$. Das heißt, es existieren für Sprachen aus NP große Beweise π, die mit konstant vielen Bitqueries randomisiert auf Richtigkeit überprüft werden können.

2. In Abschnitt 11.2 zeigen wir dann, daß $NP \subseteq PCP(\log n, \operatorname{poly}\log n)$ ist, mit der Besonderheit, daß der Tester nur $O(1)$ Segmente der Länge $\operatorname{poly}\log n$ nachfragt. Wenn wir die Anzahl der Queries in nachgefragten Paketen oder Segmenten messen würden, kämen wir also mit konstant vielen Nachfragen aus.

3. In Abschnitt 11.3 werden wir nun Beweise für eine NP-vollständige Sprache derart rekursiv kodieren, daß in jeder Rekursion immer nur $O(1)$ Pakete des Beweises nachgefragt werden. Die Beweisgröße wird dabei kleiner und die Größe der Pakete ist durch eine

Funktion in Abhängigkeit der Paketgröße des Beweissystems im vorherigen Rekursionsschritt bestimmt. Wenn wir nun mit einem $PCP(\text{poly}(n), 1)$-Beweissystem anfangen, dann können wir es nach endlich vielen Rekursionsschritten auf ein $PCP(\log n, 1)$-Beweissystem reduzieren.

Dies war eine sehr grobe Skizze des recht komplizierten Beweises, der im folgenden ausführlicher behandelt wird.

11.1 ,,Parity based encoding"

Wir beginnen mit der folgenden schwächeren Version des PCP-Theorems.

Satz 11.1 $NP \subseteq PCP(\text{poly}(n), 1)$

BEWEIS:
Wir werden zeigen, daß 3SAT$\in PCP(n^3, 1)$ gilt.

Sei $f(X_1, ..., X_n)$ eine 3SAT-Formel über n Variablen. Nehmen wir nun an, es gibt eine erfüllende Belegung $a = a_1...a_n$ von f.

Wir werden a so kodieren, daß der resultierende Code exponentielle Länge in $|a|$ hat. Da alle exponentiell vielen Stellen zueinander ,,passen" müssen, wird die Wahrscheinlichkeit, daß der Beweis den Test täuscht, sehr klein sein. Man sieht später, daß der Beweis nur an $O(1)$ Stellen untersucht werden muß, damit der Tester mit hoher Sicherheit garantieren kann, daß der Beweis korrekt ist.

Definiere für beliebige Vektoren $x \in \text{GF}(2)^l$ und $y \in \text{GF}(2)^m$, $x \circ y := (x_i y_j) \in \text{GF}(2)^{l \times m}$. Mit unserer Belegung a sei nun

$$b := a \circ a \in \text{GF}(2)^{n \times n} \text{ und } c := a \circ b \in \text{GF}(2)^{n \times n \times n}.$$

Man erhält mit a, b, c als Koeffizienten lineare Funktionen $A(x), B(y), C(z)$ wie folgt:

$$A : \mathrm{GF}(2)^n \to \mathrm{GF}(2) \quad , \quad A(x) := \langle a, x \rangle = \sum_{i=1}^{n} a_i x_i$$
$$B : \mathrm{GF}(2)^{n\times n} \to \mathrm{GF}(2) \quad , \quad B(y) := \langle b, y \rangle = \sum_{i=1}^{n}\sum_{j=1}^{n} b_{ij} y_{ij}$$
$$C : \mathrm{GF}(2)^{n\times n\times n} \to \mathrm{GF}(2) \quad , \quad C(z) := \langle c, z \rangle = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} c_{ijk} z_{ijk}$$

Der Beweis soll nun aus drei Tabellen $A := (A(x))_{x\in \mathrm{GF}(2)^n}, B := (B(y))_{y\in \mathrm{GF}(2)^{n\times n}}, C = (C(z))_{z\in \mathrm{GF}(2)^{n\times n\times n}}$ bestehen.

Angenommen, der Beweis besteht aus den Tabellen $\tilde{A}, \tilde{B}, \tilde{C}$, die angeblich eine erfüllende Belegung von f kodieren. Ziel ist es, einen $(poly(n), 1)$-beschränkten Tester zu konstruieren, der Def. 2.4 genügt. Er muß zwei Eigenschaften des Beweises überprüfen:

1. Die Tabellen $\tilde{A}, \tilde{B}, \tilde{C}$ kodieren (gemäß obiger Konstruktion) eine Belegung $\tilde{a}$ von f ;

2. $\tilde{a}$ erfüllt auch tatsächlich die 3SAT-Formel f.

Test für Punkt 1:

Betrachte folgenden Algorithmus:

Eingabe: 2 Konstanten $k_1, k_2 \in I\!N$ und Lesezugriff auf die Tabellen $\tilde{A}, \tilde{B}, \tilde{C}$.

Ausgabe: Akzeptiere genau dann, wenn der Beweis **Test 1** und **Test 2** besteht.

Test 1: Wiederhole die folgenden 3 Tests k_1-mal :

1. Teste für $x, x' \in_R \mathrm{GF}(2)^n$,

$$\tilde{A}(x) + \tilde{A}(x') = \tilde{A}(x + x').$$

2. Teste für $y, y' \in_R \mathrm{GF}(2)^{n\times n}$,

$$\tilde{B}(y) + \tilde{B}(y') = \tilde{B}(y + y').$$

3. Teste für $z, z' \in_R \mathrm{GF}(2)^{n\times n\times n}$,

$$\tilde{C}(z) + \tilde{C}(z') = \tilde{C}(z + z').$$

Test 2: Wiederhole die folgenden 2 Tests k_2-mal :

1. Gilt für $x, y \in_R \mathrm{GF}(2)^n - \{0\}$,

$$\text{SC-A}(x) \cdot \text{SC-A}(y) = \text{SC-B}(x \circ y)$$

2. Gilt für $x \in_R \mathrm{GF}(2)^n - \{0\}, y \in_R \mathrm{GF}(2)^{n\times n} - \{0\}$,

$$\text{SC-A}(x) \cdot \text{SC-B}(y) = \text{SC-C}(x \circ y)$$

Mit den Unterprogrammen,

SC-A $(x \in \mathrm{GF}(2)^n)$

Wähle $r \in_R \mathrm{GF}(2)^n$;

Return$(\tilde{A}(r) + \tilde{A}(x - r)) = \tilde{A}(x)$, falls $\tilde{A}$ linear.

SC-B $(y \in \mathrm{GF}(2)^{n\times n})$

Wähle $r \in_R \mathrm{GF}(2)^{n\times n}$;

Return$(\tilde{B}(r) + \tilde{B}(y - r)) = \tilde{B}(y)$, falls $\tilde{B}$ linear.

SC-C $(z \in \mathrm{GF}(2)^{n\times n\times n})$

Wähle $r \in_R \mathrm{GF}(2)^{n\times n\times n}$;

Return$(\tilde{C}(r) + \tilde{C}(z - r)) = \tilde{C}(z)$, falls $\tilde{C}$ linear.

Definition 11.1 *$f, g : D \to R$ seien zwei Funktionen über endlichen Mengen D, R. Der Abstand $\Delta(f, g)$ zwischen f und g sei wie folgt definiert:*

$$\Delta(f, g) := \Pr_{x \in_R D}[f(x) \neq g(x)] = \frac{|\{x \in D | f(x) \neq g(x)\}|}{|D|}$$

(f, g) heißt δ-nah, wenn $\Delta(f, g) \leq \delta$.

Lemma 11.1 *Für gegebene Konstanten $0 \leq \delta, p \leq 1$ existieren Konstanten k_1, k_2, so daß der obige Algorithmus mit Wahrscheinlichkeit von mindestens p verwirft, wenn keine Belegung a von f existiert (und somit auch keine induzierten A, B, C) , für die $(A, \tilde{A}), (B, \tilde{B}) und (C, \tilde{C})$ alle δ-nah sind.*

BEWEIS:

Betrachte zunächst folgendes Lemma von Blum et al. [BLR90]:

Lemma 11.2 *Sei $\tilde{g} : F \to F'$ eine Funktion über zwei endlichen Körpern F und F'. Wenn*

$$\Pr_{x,y \in_R F}[\tilde{g}(x) + \tilde{g}(y) \neq \tilde{g}(x+y)] \leq \frac{\delta}{2},$$

dann existiert eine lineare Funktion $g : F \to F'$, so daß $(g, \tilde{g})$ δ-nah ist.

■(Lemma 11.2)

Angenommen, es gibt keine lineare Funktion, die δ-nah an $\tilde{A}$ liegt. Dann folgt aus dem obigen Lemma, daß

$$\Pr_{x,y \in_R \mathrm{GF}(2)^n}[\tilde{A}(x) + \tilde{A}(y) \neq \tilde{A}(x+y)] > \frac{\delta}{2}.$$

Analog für $\tilde{B}, \tilde{C}$. Daraus folgt, daß nach bestandenem Test mit hoher Wahrscheinlichkeit lineare Funktionen mit Koeffizienten $a \in \mathrm{GF}(2)^n, b \in \mathrm{GF}(2)^{n \times n}, c \in \mathrm{GF}(2)^{n \times n \times n}$ existieren, so daß $(a, \tilde{A}), (b, \tilde{B}), (c, \tilde{C})$ δ-nah aneinander liegen. Wir werden nun zeigen, daß mit beschränkter Fehlerwahrscheinlichkeit $b = a \circ a$ und $c = b \circ a$ garantiert werden kann, wenn auch Test 2 erfolgreich war.

Lemma 11.3 *Sei $x \in$ GF(2)$^n \setminus \{0\}$. Dann existieren genau 2^{n-1} zu x orthogonale Vektoren $y \in$ GF(2) (Bzw. genau 2^{n-1} Vektoren $y' \in$ GF(2)$^n \setminus \{0\}$, für die $x^t y' = 1$ gilt).*

BEWEIS: Sei $x \in \mathrm{GF}(2)^n \setminus \{0\}$. O.B.d.A. sei $x_n = 1$. Man nehme nun ein $y \in \mathrm{GF}(2)^n$ und ordne y eindeutig $y' = (y'_i)$ mit

$$y'_i = \begin{cases} y_i & , \quad i < n \\ 1 - y_i & , \quad i = n \end{cases}$$

zu. Nimm an, daß $y_n = 0$. Dann gilt

$$x^t y = \sum_{i=1}^{n-1} x_i y_i \neq \sum_{i=1}^{n-1} x_i y_i + 1 = x^t y'$$

Analoges gilt für den Fall $y_n = 1$. Auf jeden Fall existiert eine Bijektion zwischen den zu x orthogonalen und nichtorthogonalen Vektoren, womit die Behauptung folgt. ■(Lemma 11.3)

Aus Lemma 11.3 folgt, daß für $\alpha, \beta \in \mathrm{GF}(2)^n, \alpha \neq \beta \quad \Pr_{y \in_R \{0,1\}^n \setminus \{0\}}[\alpha^T y \neq \beta^T y] = \frac{1}{2}$ gilt. Da für ein $V \in \mathrm{GF}(2)^{n \times n}$ $x^T V = (x^T v^i)_i$, folgt dann, daß für $X, Y \in \mathrm{GF}(2)^{n \times n}, X \neq Y$

$$\Pr_{x \in_R \{0,1\}^n - \{0\}}[x^T X \neq x^T Y] = \Pr[x^T(x^1 - y^1) \neq 0 \vee ... \vee x^T(x^n - y^n) \neq 0] \geq \frac{1}{2}$$

gilt. Kombiniert man beide Ergebnisse, erhält man für $X, Y \in \mathrm{GF}(2)^{n \times n}, X \neq Y$

$$\Pr_{x,y \in_R \{0,1\}^n - \{0\}}[x^T X y \neq x^T Y y] \geq \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}.$$

Mit $X := a^T a$ und $Y = (b_{ij})$ folgt, daß (unter der Annahme,daß $\tilde{A}, \tilde{B}$ linear sind) der erste Test in **Test 2** mit Wahrscheinlichkeit größer oder gleich $\frac{1}{4}$ negativ ist, wenn $b \neq a \circ a$ (Beachte, daß $x a^T a y^T = (a x^T)(a y^T)$ und $\sum_{i,j} b_{ij} x_i y_j = \sum_i x_i \sum_j b_{ij} y_j = x B y^T$). Gleiches Argument gilt für $c = a \circ b$.

Was ist, wenn $\tilde{A}, \tilde{B}, \tilde{C}$ nicht gleich, sondern nur δ-nah an den linearen Funktionen A,B,C mit Koeffizienten a, b, c liegen? Es muß sichergestellt werden, daß die beim **Test 2** erforderlichen Werte korrekt sind. Betrachte das Unterprogramm SC-A. Es gilt wegen Lemma 11.2:

$$\Pr_r[\tilde{A}(x - r) \neq A(x - r)] \leq \delta \text{ und } \Pr_r[\tilde{A}(r) \neq A(r)] \leq \delta$$

$$\Rightarrow \Pr_r[\tilde{A}(x - r) \neq A(x - r) \vee \tilde{A}(r) \neq A(r)] \leq 2\delta$$

Da $x_1 + x_2 \neq y_1 + y_2$ impliziert, daß $x_1 \neq y_1 \vee x_2 \neq y_2$ folgt

$$\Pr_r[\tilde{A}(x-r) + \tilde{A}(r) \neq A(x-r) + A(r)] \leq 2\delta.$$

Also ist das Ergebnis von SC-A mit Wahrscheinlichkeit größer oder gleich $(1-2\delta)$ richtig (Gleiches Argument gilt für SC-B,SC-C). Also ist die Gesamtfehlerwahrscheinlichkeit, wenn beide Tests akzeptieren, durch die Konstante $2 \cdot \frac{1}{4}(1-2\delta)$ nach oben beschränkt. Durch konstant viele Iterationen kann jede beliebige Schranke p erreicht werden. $\Longrightarrow$ Lemma 11.1.

■(Lemma 11.1)

Test für Punkt 2.

Wir wissen zwar, daß mit hoher Wahrscheinlichkeit $\tilde{A}, \tilde{B}, \tilde{C}$ eine Belegung $\tilde{a}$ von f kodieren, jedoch nicht, ob $\tilde{a}$ f auch wirklich erfüllt.

f hat als 3SAT-Formel die Form: $f(x) = \bigwedge_{j \in J} C_j(x)$ mit den Klauseln

$$\begin{aligned} C_j(x) &= (x_{j1}^{\alpha_{j1}} \vee x_{j2}^{\alpha_{j2}} \vee x_{j3}^{\alpha_{j3}}) = \\ \text{(De Morgan)} &= \neg \underbrace{(x_{j1}^{1-\alpha_{j1}} \wedge x_{j2}^{1-\alpha_{j2}} \wedge x_{j3}^{1-\alpha_{j3}})}_{=:C'_j(x)} \\ \text{Mit } y^\alpha &:= \begin{cases} y & , \quad \alpha = 0 \\ \neg y & , \quad \alpha = 1 \end{cases} \end{aligned}$$

Arithmetisiere die C'_j wie folgt:

$$(x_{j1}^{\beta_{j1}} \wedge x_{j2}^{\beta_{j2}} \wedge x_{j3}^{\beta_{j3}}) = C'_j(x) \mapsto \hat{C}_j(x) := (\beta_{j1} + x_{j1})(\beta_{j2} + x_{j2})(\beta_{j3} + x_{j3})$$

Beachte dabei, daß über GF(2) $\quad 1 - x = 1 + x$ gilt. Aus der Konstruktion folgt dann, daß

$$f(a) = 1 \Leftrightarrow \hat{C}(a) := (\hat{C}_j(a))_{j \in J} = 0$$

Wenn $\hat{C}(a)$ nicht der Nullvektor ist, folgt wieder nach Lemma 11.3, daß

$$\Pr_{r \in_R \{0,1\}^{|J|} \setminus \{0\}}[< \hat{C(a)}, r > \neq 0] \geq \frac{1}{2}$$

gilt. Für beliebiges $r \in \{0,1\}^{|J|}$ hat $< \hat{C}(a), r >$ folgende Form:

$$\begin{aligned}
< \hat{C}(a), r > &= \sum_{j \in J} r_j \hat{C}_j(a) = \\
&= \sum_{j \in J} r_j (\beta_{j1} + a_{j1})(\beta_{j2} + a_{j2})(\beta_{j3} + a_{j3}) = \\
(*) &= \sum_{j \in J} r_j \begin{matrix} (\beta_{j1}\beta_{j2}\beta_{j3} + \beta_{j1}\beta_{j3}a_{j2} + \beta_{j2}\beta_{j3}a_{j1} + \beta_{j3}a_{j1}a_{j2} + \\ + \beta_{j1}\beta_{j2}a_{j3} + \beta_{j1}a_{j2}a_{j3} + \beta_{j2}a_{j1}a_{j3} + a_{j1}a_{j2}a_{j3}) \end{matrix} = \\
&= \underbrace{\sum_{j \in J} r_j \beta_{j1}\beta_{j2}\beta_{j3}}_{=:\gamma} + \\
&\quad + \sum_{i=1}^{n} a_i \underbrace{\Big(\sum_{l: r_l \text{ Koeff. v.} a_i \text{ in } (*)} r_l \Big) \Big(\sum_{m_1, m_2: \beta_{m1}\beta_{m2} \text{Koeff. v. } a_i \text{ in } (*)} \beta_{m1}\beta_{m2} \Big)}_{=:\phi_i} + \\
&\quad + \sum_{i,j=1}^{n} a_i a_j \underbrace{\Big(\sum_{l: r_l \text{ Koeff. v. } a_i a_j \text{ in } (*)} r_l \Big) \Big(\sum_{m: \beta_m \text{ Koeff. v. } a_i a_j \text{ in } (*)} \beta_m \Big)}_{=:\psi_{ij}} + \\
&\quad + \sum_{i,j,k=1}^{n} a_i a_j a_k \underbrace{\Big(\sum_{l: r_l \text{ Koeff. v. } a_i a_j a_k \text{ in } (*)} r_l \Big)}_{=:\omega_{ijk}} = \\
(**) &= \gamma + A(\phi) + B(\psi) + C(\omega)
\end{aligned}$$

Beachte dabei, daß man $\gamma, \phi, \psi, \omega$ in polynomieller Zeit berechnen kann. Es genügt also, den Ausdruck $(**)$ zu berechnen und genau dann zu akzeptieren, wenn er Null ergibt. Wir können jedoch nicht sicher sein, daß die Werte stimmen,da $\tilde{A}, \tilde{B}, \tilde{C}$ nur $\delta - nah$ an A, B, C liegen. Wie zuvor können wir die Selbstreduzierung von linearen Funktionen zur Hilfe nehmen und erhalten mit Wahrscheinlichkeit $\geq (1 - 2\delta)$ jeweils das richtige Ergebnis. Somit machen wir, wenn wir akzeptieren, mit Wahrscheinlichkeit kleiner oder gleich $\frac{1}{2}(1 - 6\delta)$ einen Fehler.

Resultieren wir:

- Die Gesamtfehlerwahrscheinlichkeit ist durch eine Konstante nach oben beschränkt;
- alle Tests sind in polynomieller deterministischer Zeit berechenbar;

- da f eine 3SAT-Formel ist, folgt $|J| \le \binom{n}{3} = O(n^3)$. Also ist die Anzahl der Zufallsbits $O(n^3)$ und
- die Anzahl der Fragen an den Beweis ist konstant.

Somit folgt, daß 3SAT $\in PCP(n^3, 1)$. ■(Satz 11.1)

11.2 Segmentierung der Fragen

Im nächsten Unterabschnitt werden wir einen randomisiert überprüfbaren Beweis (für Sprachen aus NP) benötigen, der mit konstant vielen Fragen auf Korrektheit getestet werden kann. In unserem Fall ist die Länge der Fragen jeweils $\operatorname{poly}\log(n)$. Man kann diese Eigenschaft auch als Segmentierung des Beweises betrachten, wobei der Tester nur konstant viele Segmente des Beweises untersucht.

Satz 11.2 *Für jede Sprache aus NP gibt es ein PCP-Beweissystem, so daß der Tester*

- $O(\log(n))$ *Zufallsbits benötigt und*
- $O(1)$ *Segmente der Länge* $\operatorname{poly}\log(n)$ *nachfragt.*

Zur Vorbereitung des Beweises von Satz 11.2 betrachte das folgendende schwächere Resultat von Babai et al.:

Satz 11.3 ([BFLS91])

$$NP = PCP(\log n, \operatorname{poly}\log n)$$

Bemerkung 11.1 *Wenn f eine 3SAT-Formel ist und w eine erfüllende Belegung von f, dann ist die Länge $l = |\pi|$ des $PCP(\log n, \operatorname{poly}\log n)$-Beweises π, $l := n^{O(1)}$.*

Im folgenden sei $d := \lceil \frac{\log n}{\log \log n} \rceil$, $I := \{1, ..., \lceil \log n \rceil\}$. Da

$$\lceil \log n \rceil^{\lceil \frac{\log n}{\log \log n} \rceil} \geq \log n^{\frac{\log n}{\log \log n}} = 2^{\log \log n \frac{\log n}{\log \log n}} = n$$

gilt, kann man eine Zeichenkette x über $\{0, 1\}$ der Länge n auch als Abbildung x von I^d nach $\{0, 1\}$ betrachten. Unser Ziel ist, diese Abbildung in ein Polynom (mit beschränktem Grad) über einem endlichen Körper einzubetten. Sei also $q \in [(k \log l)^4, 2(k \log l)^4]$ eine Primzahlpotenz für $k \in I\!N$ eine Konstante. Es gilt nun, daß $Q := \text{GF}(q) \supseteq I$.

Lemma 11.4 (Low-Degree-Erweiterung) *Sei $f : H^m \to F$ mit $H \subseteq F$ und F ein endlicher Körper. Dann existiert ein eindeutiges m-variates Polynom $\hat{f}$ über F, so daß*

- *der Grad von $\hat{f}$ in jeder Variable $\leq |H| - 1$ ist. Also ist* $\deg \hat{f} \leq m(|H| - 1)$.
- $\hat{f}(x) = f(x) \quad \forall x \in H^m$.

$\hat{f}$ wird auch die Low-Degree-Erweiterung von f genannt.

Beweis:

Sei $u \in H$ und

$$L_u(x) := \frac{\prod_{i \in H \setminus \{u\}} (x - i)}{\prod_{i \in H \setminus \{u\}} (u - i)} = \begin{cases} 1 & , \quad x = u \\ 0 & , \quad x \neq u \end{cases}$$

L_u ist ein univariates Polynom mit Grad $|H| - 1$. Nun können wir $\hat{f}$ wie folgt definieren:

$$\hat{f}(x) := \sum_{u=(u_1,...,u_m) \in H^m} f(u) \prod_{i=1}^{m} L_{u_i}(x_i)$$

Die Eindeutigkeit folgt aus der Tatsache, daß $\hat{f}$ höchstens $m(|H|-1)$ Nullstellen über F hat.

■((Lemma 11.4))

Beweis: (von Satz 11.2)

Sei π der Beweis des PCP Systems von Satz 11.3. Mit obigen Vorbemerkungen können wir π auch als Abbildung $\pi : I^d \to \{0,1\}$ schreiben. Das neue Beweis-Orakel bestehe nun aus folgenden Tabellen:

- T_{points} sei die Wertetabelle der Low-Degree-Erweiterung $\hat{\pi} : Q^d \to Q$ von π. Es gilt $\deg \hat{\pi} \leq |I|d$. Beachte ferner, daß

  $$q^d = O(\log^{O(1)\cdot d}(l)) = O(l^{O(1)}) = n^{O(1)}$$

  ist. Somit sind in T_{points} $n^{O(1)}$ Einträge der Länge $\operatorname{poly}\log(n)$.

- T_{lines} sei eine Tabelle, so daß $T_{lines}(a,b)$ ein univariates Polynom ist, das $\hat{\pi}(a+ib)$ als Funktion in i beschreibt. [T_{lines} hat $q^{2d} = \operatorname{poly}(n)$ Einträge. Die Länge der Einträge ist $q \log q = \log^{O(1)} n \log\log n = \operatorname{poly}\log(n)$.]

- Später wird der Tester $r = (a_0, a_1, ..., a_k) \in_R Q^d \times (I^d)^k$ wählen (Bedarf an Zufallsbits ist $O(\log \operatorname{poly} n) = O(\log n)$) und eine Funktion $p_r : Q \to Q^d$ konstruieren, so daß $p_r(i) = a_i, (i = 1, .., k)$ und die einzelnen Komponenten $(p_r)_j$ Polynome vom Grad $\leq k$ sind.

  Der Beweisende soll nun eine Tabelle T_p konstruieren, die für jede Wahl von $r = (a_0, ..., a_k)$ die Werte von $\hat{\pi} \circ p_r$ enthält. $\hat{\pi} \circ p_r$ ist ein Polynom vom Grad $\leq kd|I|$. [$n^{O(1)}$ Einträge, der Länge $\operatorname{poly}\log(n)$. (Siehe letzter Punkt)]

Der Test, ob der kodierte Beweis auch korrekt ist, verläuft wie folgt:

Schritt 1:

1. Wähle $a, b \in_R Q^d$ und $i \in_R Q$;
2. Teste, ob $T_{points}(a+ib) = T_{lines}(a,b)(i)$;
3. Wenn Test erfolgreich, so fahre fort, sonst verwerfe .

Schritt 2: Wähle $r = (a_0, a_1, ..., a_k) \in_R Q^d \times (I^d)^k$ und konstruiere p_r. Teste, ob $p_r(1), ..., p_r(k)$ den Test von Satz 11.3 bestanden hätten.

Schritt 3: Wähle $t \in_R Q$ und teste, ob $T_{p_r}(t) = T_{points} \circ p_r(t)$.

Insgesamt werden $O(1)$ Einträge (der Länge poly $\log n$) der Tabellen untersucht und $O(\log n)$ Zufallsbits benötigt. Es bleibt die Korrektheit des Testalgorithmus zu zeigen.

Beh:

$$\begin{aligned} & \Pr_{r,a,b,i,t}[T_{points}(a+ib) \neq T_{lines}(a,b)(i) \vee T_{p_r}(t) \neq T_{points} \circ p_r(t)] \leq \delta \\ \Rightarrow \quad & \Pr_{r,t}[T_{p_r} \neq T_{points} \circ p_r \wedge T_{p_r}(t) = T_{points} \circ p_r(t)] \leq 3\delta \end{aligned}$$

Die Aussage ist äquivalent zu der Aussage, daß wenn die Antworten von T_{p_r} für ein r nicht mit den Antworten von π (bei Eingaben $a_1, ..., a_k$) übereinstimmen und T_{points} nah ein einem Polynom liegt, der Beweis durch Test 3 mit konstanter Wahrscheinlichkeit fällt.

Beweis: (der Beh.)

Es gilt also, daß

$$\Pr_{(a,b,i) \in_R (Q^d)^2 \times Q}[T_{points}(a+ib) \neq T_{lines}(a,b)(i)] \leq \delta$$

(Satz 11.4 $\Rightarrow$) Es existiert ein Polynom g vom Grad $\leq |I|d$, so daß

$$\Pr_{x \in_R Q^d}[T_{points}(x) \neq g(x)] \leq 2\delta$$

Also erhalten wir:

$$\begin{aligned} & \Pr_{r,t}[T_{p_r} \neq T_{points} \circ p_r \wedge T_{p_r}(t) = T_{points} \circ p_r(t)] \leq \\ \leq \quad & \Pr_{r,t}[T_{p_r} \neq g \circ p_r \wedge T_{p_r}(t) = g \circ p_r(t)] + 2\delta \leq \\ \leq \quad & \underbrace{\frac{k|I|d}{q}}_{\leq \delta \text{ für } k \text{ groß genug}} +2\delta \quad \left(\text{Beachte } \frac{k \log n \frac{\log n}{\log\log n}}{k^4 \log^4 n} = \frac{1}{k^3 \log^2 n \log\log n}\right) \end{aligned}$$

Womit Satz 11.2 bewiesen wäre. ■((Satz 11.2))

11.2.1 Der Low-Degree-Test

Bemerkung 11.2 *Wenn* $\Pr_{x\in X}[\psi(x) \neq \phi(x)] \leq \delta$ *, dann folgt aus der Markovschen Ungleichung,*

$$\Pr_{x\in X}\left[\frac{|\{x|\psi(x) = \phi(x)\}|}{|X|} \geq (1-\epsilon)\right] \geq 1 - \underbrace{\Pr\underbrace{\left[\frac{|\{x|\psi(x) = \phi(x)\}|}{|X|} \leq (1-\epsilon)\right]}_{\frac{|\{x|\psi(x)\neq\phi(x)\}|}{|X|}\geq\epsilon}}_{\leq\frac{\delta}{\epsilon}} \geq 1 - \frac{\delta}{\epsilon}$$

■

Für $\hat{x}, \hat{h} \in F^n$ sei $P_{\hat{x},\hat{h}}(y)$ ein Polynom vom Grad d in y, das die Anzahl der gemeinsamen Punkte mit $f(\hat{x} + y\hat{h})$ maximiert.

Satz 11.4 *Sei* $f : F^n \to F$ *gegeben, so daß*

$$\Pr_{\hat{x},\hat{h}\in_R F^n}[f(\hat{x}) = P_{\hat{x},\hat{h}}(0)] \geq 1 - \delta$$

Dann existiert ein Polynom $g : F^n \to F$ *mit* $\deg(g) \leq d$*, so daß*

$$\Pr_{\hat{x}\in_R F^n}[f(\hat{x}) = g(\hat{x})] \geq 1 - 2\delta.$$

Beweis: via Lemma 11.5-11.8. ■

Bemerkung 11.3 *Es gilt folgende Äquivalenz:*

$$\Pr_{\hat{x},\hat{h}\in_R F^n}[f(\hat{x}) = P_{\hat{x},\hat{h}}(0)] \geq 1 - \delta \Leftrightarrow \Pr_{\hat{x},\hat{h}\in_R F^n, y\in_R F}[f(\hat{x} + y\hat{h}) = P_{\hat{x},\hat{h}}(y)] \geq 1 - \delta$$

Beweis: Die Rückrichtung ist klar. Umgekehrt maximiert $P_{\hat{x},\hat{h}}(y) = P_{\hat{x}+y\hat{h},\hat{h}_1}(0)$ die Anzahl der gemeinsamen Punkte mit $f(\hat{x} + y\hat{h} + 0\hat{h}_1)$. Also folgt

$$\Pr_{\hat{x},\hat{h}_0,\hat{h}_1,y}[\underbrace{f(\hat{x} + y\hat{h}_0 + 0\hat{h}_1)}_{=f(\hat{x}+y\hat{h}_0)} = \underbrace{P_{\hat{x}+y\hat{h}_0,\hat{h}_1}(0)}_{=P_{\hat{x},\hat{h}_0}(y)}] \geq 1 - \delta$$

■

Definition 11.2 *Sei $M = (m_{yz})_{yz \in F}$ eine Matrix. Dann heißt das Polynom*

- *in z, das die Anzahl der gemeinsamen Punkte mit der y-ten Reihe in M maximiert, $R_y^M(z)$.*
- *in y, das die Anzahl der gemeinsamen Punkte mit der z-ten Spalte in M maximiert, $C_z^M(y)$.*

Betrachte folgendes Lemma ohne Beweis:

Lemma 11.5 (Arora, Safra [AS92b])
Es gibt ein $\bar{\epsilon}$, so daß $\forall \epsilon < \bar{\epsilon}$ bei gegebener Matrix $M = (m_{yz})_{yz \in F}$ mit der Eigenschaft

$$\Pr_{y,z}[m_{yz} = R_y^M(z)] \geq 1 - \epsilon \text{ und } \Pr_{y,z}[m_{yz} = C_z^M(y)] \geq 1 - \epsilon$$

ein bivariates Polynom $Q(y, z)$ mit $\deg(Q) \leq d$ in y, z, so daß

$$\Pr_y[R_y^M \equiv Q(y,)] \geq 1 - \epsilon \text{ und } \Pr_z[C_z^M \equiv Q(, z)] \geq 1 - \epsilon.$$

■

Definiere nun g wie folgt:

$$g \equiv \mathrm{majority}_{\hat{h} \in F^n}\{P_{\hat{x},\hat{h}}(0)\} \Longleftrightarrow [g(\hat{x}) = \bar{z} \Leftrightarrow |\{\hat{h}|P_{\hat{x},\hat{h}}(0) = \bar{z}\}| \geq |\{\hat{h}|P_{\hat{x},\hat{h}}(0) = z\}| \forall z \neq \bar{z}]$$

Wie oben sei $P'_{\hat{x},\hat{h}}(y)$ ein Polynom vom Grad d in y, das die Anzahl der gemeinsamen Punkte mit $g(\hat{x} + y\hat{h})$ maximiert.

Lemma 11.6

$$\Pr_{\hat{x} \in_R F^n}[f(\hat{x}) = g(\hat{x})] \geq 1 - 2\delta.$$

Beweis: Sei $G_{\hat{x}} := \{\hat{h}|g(\hat{x}) = P_{\hat{x},\hat{h}}(0)\}$ und $F_{\hat{x}} := \{\hat{h}|f(\hat{x}) = P_{\hat{x},\hat{h}}(0)\}$.
Es gilt:

$$g(\hat{x}) \neq f(\hat{x}) \Leftrightarrow |\underbrace{G_{\hat{x}} \cap F_{\hat{x}}}_{F^n \setminus (\bar{G}_{\hat{x}} \cup \bar{F}_{\hat{x}})}| = 0 \Rightarrow |\bar{G}_{\hat{x}} \cup \bar{F}_{\hat{x}}| \geq 1$$

Also gilt

$$\begin{array}{rcl} \Pr_{\hat{x}}[g(\hat{x}) \neq f(\hat{x})] & \leq & \Pr_{\hat{x}}[|\bar{G_{\hat{x}}} \cup \bar{F_{\hat{x}}}| \geq 1] \leq \\ \text{(Markowsche Ungleichung)} & \leq & \mathrm{E}_{\hat{x}}[|\bar{G_{\hat{x}}} \cup \bar{F_{\hat{x}}}|] \leq \\ & \leq & \mathrm{E}_{\hat{x}}[|\bar{G_{\hat{x}}}|] + \mathrm{E}_{\hat{x}}[|\bar{F_{\hat{x}}}|] \leq 2\delta \end{array}$$

■

Lemma 11.7

$$\Pr_{\hat{h}_1,\hat{h}_2}[P_{\hat{x},\hat{h}_1}(0) = P_{\hat{x},\hat{h}_2}(0)] \geq 1 - \frac{4\delta}{\epsilon}$$

Beweis: Sei $\hat{h}_1, \hat{h}_2 \in F^n$ und $M = (m_{yz})$ mit $m_{yz} = f(\hat{x} + y\hat{h}_1 + z\hat{h}_2)$ gegeben. Aus Bem 11.3 und der Tatsache, daß $\hat{x} + y\hat{h}_1, \hat{h}_2$ zufällig gewählt und unabhängig sind, folgt

$$\begin{array}{rcl} \Pr_{\hat{h}_1,\hat{h}_2 \in_R F^n}[f(\hat{x} + y\hat{h}_1 + z\hat{h}_2) = P_{\hat{x}+z\hat{h}_2,\hat{h}1}(y)] & \geq & 1 - \delta \\ \Pr_{\hat{h}_1,\hat{h}_2 \in_R F^n}[f(\hat{x} + y\hat{h}_1 + z\hat{h}_2) = P_{\hat{x}+z\hat{h}_1,\hat{h}2}(z)] & \geq & 1 - \delta \end{array}$$

(Bem. 11.2) $\Longrightarrow$ Die Annahme, daß

$$\Pr_{y,z}[\underbrace{m_{yz} = R_y^M(z)}_{f(\hat{x}+y\hat{h}_1+z\hat{h}_2)=P_{\hat{x}+y\hat{h}_1,\hat{h}_2}(z)}] \geq 1 - \epsilon \text{ und } \Pr_{y,z}[\underbrace{m_{yz} = C_z^M(y)}_{(\ldots)}] \geq 1 - \epsilon$$

ist mit Fehlerwahrscheinlichkeit $\leq \frac{2\delta}{\epsilon}$ richtig.

(Lemma 11.5) $\Longrightarrow$ Es existiert ein bivariates Polynom $Q(y, z)$ mit $\deg(Q) \leq d$, so daß mit

$$Y_1 := \{y | R_y^M \equiv Q(y,)\} \text{ und } Z_1 := \{z | C_z^M \equiv Q(, z)\},$$

$|Y_1|, |Z_1| \geq (1 - \epsilon)|F|$ gilt.

Seien $y, z \in_R F$. Da $\hat{x} + y\hat{h}_1, \hat{h}_2$ zufällig gewählt und unabhängig sind, gilt

$$\begin{array}{rcl} \Pr_{\hat{h}_1,\hat{h}_2 \in_R F^n}[f(\hat{x} + y\hat{h}_1) = P_{\hat{x}+y\hat{h}_1,\hat{h}2}(0)] & \geq & 1 - \delta \\ \Pr_{\hat{h}_1,\hat{h}_2 \in_R F^n}[f(\hat{x} + z\hat{h}_2) = P_{\hat{x}+z\hat{h}_2,\hat{h}1}(0)] & \geq & 1 - \delta \end{array}$$

Sei nun

$$Y_2 := \{y | f(\hat{x} + y\hat{h}_1) = P_{\hat{x}+y\hat{h}_1,\hat{h}2}(0)\} \text{ und } Z_2 := \{z | f(\hat{x} + z\hat{h}_2) = P_{\hat{x}+z\hat{h}_2,\hat{h}1}(0)\}$$

Da $\frac{|Y_2|}{|F|} \geq 1 - \delta$ gilt, folgt nach Bem. 11.2

$$\Pr_{\hat{h}_1,\hat{h}_2}[|Y_2| \geq (1-\epsilon)|F|] \geq 1 - \frac{\delta}{\epsilon}$$

Analog erhält man $\Pr_{\hat{h}_1,\hat{h}_2}[|Z_2| \geq (1-\epsilon)|F|] \geq 1 - \frac{\delta}{\epsilon}$
Mit Fehlerwahrscheinlichkeit $\leq \frac{4\delta}{\epsilon}$ kann man nun annehmen, daß

$$\frac{|F \setminus Y_2|}{|F|} + \frac{|F \setminus Y_1|}{|F|} \leq 2\epsilon \text{ und } \frac{|F \setminus Z_2|}{|F|} + \frac{|F \setminus Z_1|}{|F|} \leq 2\epsilon$$

Da $2\epsilon \geq \frac{1}{|F|}(|\bar{Y_1}| + |\bar{Y_2}|) \geq \frac{1}{|F|}(|\bar{Y_1} \cup \bar{Y_2}|) = \frac{1}{|F|}(|F| - |Y_1 \cap Y_2|)$ gilt, folgt

$$|Y_1 \cap Y_2| \geq (1-2\epsilon)|F| \text{ und analog } |Z_1 \cap Z_2| \geq (1-2\epsilon)|F|$$

bei gleicher Fehlerwahrscheinlichkeit.

Für $y \in Y_1 \cap Y_2$ folgt dann $P_{\hat{x}+y\hat{h}_1,\hat{h}2}(z) = Q(y,z)$ und $m_{y0} = P_{\hat{x}+y\hat{h}_1,\hat{h}2}(0)$
$\Rightarrow m_{y0} = Q(y,0)$. Wenn nun $(1-2\epsilon) > \frac{1}{2} \Rightarrow P_{\hat{x},\hat{h}_1}(y) \equiv C_0^M \equiv Q(y,z)\,|_{z=0}$. Analog gilt dann $P_{\hat{x},\hat{h}_2}(z) \equiv R_0^M \equiv Q(y,z)\,|_{y=0}$. Daraus folgt dann, mit WK $\leq \frac{4\delta}{\epsilon}$

$$P_{\hat{x},\hat{h}_1}(0) = Q(0,0) = P_{\hat{x},\hat{h}_2}(0).$$

$\Longrightarrow \quad \Pr_{\hat{h}_1,\hat{h}_2}[P_{\hat{x},\hat{h}_1}(0) = P_{\hat{x},\hat{h}_2}(0)] \geq 1 - \frac{4\delta}{\epsilon}$ ∎

Lemma 11.8

$$\forall \hat{x}, \hat{h} \textit{ gilt } g(\hat{x}) = P'_{\hat{x},\hat{h}}(0)$$

Beweis:

Seien $\hat{h}_1, \hat{h}_2 \in_R F^m$ und definiere die Matrix $M = (m_{yz})$ mit

$$m_{y0} = g(\hat{x} + y\hat{h}) \text{ und } m_{yz} = f(\hat{x} + y\hat{h} + z(\hat{h}_1 + y\hat{h}_2)) \quad (z \geq 1)$$

Es gilt

$$\begin{aligned}
\Pr_{\hat{h}_1,\hat{h}_2}[f(\hat{x}+y\hat{h}+z(\hat{h}_1+y\hat{h}_2)) \neq g(\hat{x}+y\hat{h}+z(\hat{h}_1+y\hat{h}_2))] &\leq \delta \\
\Pr_{\hat{h}_1,\hat{h}_2}[f(\hat{x}+y\hat{h}+z(\hat{h}_1+y\hat{h}_2)) \neq P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(z)] &\leq \delta \\
(\text{Lemma 11.7}) \Rightarrow \Pr_{\hat{h}_1,\hat{h}_2}[g(\hat{x}+y\hat{h}+z(\hat{h}_1+y\hat{h}_2)) \neq P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(z)] &\leq \frac{4\delta}{\epsilon}
\end{aligned}$$

Insgesamt ergibt dies

$$\Pr_{\hat{h}_1,\hat{h}_2}[m_{yz} = f(\hat{x}+y\hat{h}+z(\hat{h}_1+y\hat{h}_2)) = g(\hat{x}+y\hat{h}+z(\hat{h}_1+y\hat{h}_2)) = P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(z)] \geq 1-\delta_1$$

mit $\delta_1 := 2\delta + \frac{4\delta}{\epsilon}$. Wie bei Lemma 3 folgt mit Bem. 11.2, daß

$$\Pr_{y,z}[m_{yz} = R_y^M(z)] \geq 1-\epsilon \text{ und } \Pr_{y,z}[m_{yz} = C_z^M(y)] \geq 1-\epsilon$$

zutrifft mit Wahrscheinlichkeit $\geq 1-\delta_2$ $(*)$ (mit $\delta_2 = \frac{\delta_1+\delta}{\epsilon}$). (Lemma 11.5) $\Longrightarrow$ Es existiert ein bivariates Polynom $Q(y,z)$ mit $\deg(Q) \leq d$, so daß mit

$$Y_1 := \{y | R_y^M \equiv Q(y,)\} \text{ und } Z_1 := \{z | C_z^M \equiv Q(,z)\}$$

$|Y_1|, |Z_1| \geq (1-\epsilon)|F|$ gilt.

Beh:

1. $\Pr_{\hat{h}_1,\hat{h}_2}[m_{y0} = Q(y,0)] > \frac{1}{2}$
2. $\Pr_{\hat{h}_1,\hat{h}_2}[m_{00} = Q(0,0)] \geq 1-\delta_1-\delta_2$

zu 1.: Für beliebiges $y \in F$ gilt

$$\Pr_{h_1,h_2}[(m_{y0} =)g(\hat{x}+y\hat{h}) = P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(0)] \geq 1-\delta_1$$

Aus Bem 11.2 folgt dann, daß mit Wahrscheinlichkeit $\geq 1-\delta_2$ für einen Anteil von mindestens $(1-\epsilon)$ der y gilt, daß

$$m_{y0} = P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(0)$$

Für $y \in Y_1$ gilt jedoch $P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(0) = Q(y,0)$. Wenn $1-2\epsilon > \frac{1}{2}$, folgt Beh. 1 :

$$\Pr_{\hat{h}_1,\hat{h}_2}[m_{y0} \neq Q(y,0)] \leq \Pr_{\hat{h}_1,\hat{h}_2}[m_{y0} \neq P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(0) \vee P_{\hat{x}+y\hat{h},\hat{h}_1+y\hat{h}_2}(0) \neq Q(y,0)] \leq 2\epsilon$$

zu 2.: Analog zeigt man, daß für $z \in Z_1$, $\Pr_{\hat{h}_1,\hat{h}_2}[P_{\hat{x},\hat{h}_1}(z) = m_{0z} = Q(0,z)] > \frac{1}{2}$ mit Wahrscheinlichkeit $\geq 1 - \delta_2$, so daß $P_{\hat{x},\hat{h}_1}(z) = Q(0,z)$. Setzt man $y = 0$ so erhält man

$$\Pr_{h_1,h_2}[(m_{00} =)g(\hat{x}) = P_{\hat{x},\hat{h}_1}(0)] \geq 1 - \delta_1$$

$$\Longrightarrow \Pr_{\hat{h}_1,\hat{h}_2}[m_{00} \neq Q(0,0)] \leq \delta_1 + \delta_2 \Rightarrow \text{ Beh. 2}$$

Insgesamt gilt nun

- Mit Wk $\leq \delta_2$ gilt $m_{y0} \neq Q(y,0)$.
- Mit Wk $\leq \delta_2 + \delta_1$ gilt $m_{00} \neq Q(0,0)$.

Wegen $(*)$ folgt dann:

$$(**) = \Pr_{\hat{h}_1,\hat{h}_2}[\underbrace{g(\hat{x}) = P'_{\hat{x},\hat{h}}(0)}_{(***)}] \geq 1 - 3\delta_2 - \delta_1$$

Da der Ausdruck $(***)$ jedoch unabhängig von $\hat{h}_1, \hat{h}_2$ ist, folgt, daß $(**) = 1$ (wenn $1 - 3\delta_2 + \delta_1 > 0$). ■

Beweis: (Satz 11.4)
(Lemma 11.6) $\Rightarrow \Pr_{\hat{x}\in_R F^n}[f(\hat{x}) = g(\hat{x})] \geq 1 - 2\delta$
(Lemma 11.8) $\Rightarrow g(\hat{x}) \equiv P'_{\hat{x},\hat{h}}(0)$ und $\deg(P') \leq d$
$\Longrightarrow$ die Behauptung von Satz 11.4. ■((Satz 11.4))

11.3 Rekursive Kodierung

Wir werden einen Beweis für eine NP-vollständige Sprache derart rekursiv kodieren, daß in jeder Rekursion die Eigenschaft erhalten bleibt, daß zum Testen nur $O(1)$ Fragen an den Beweis gestellt werden müssen. Als Basis verwenden wir das ,,*Parity based encoding*" vom vorletzten Unterabschnitt. Mit den Ergebnissen vom letzen Abschnitt können wir die Kodierungen so konstruieren, daß jeweils nur konstant viele Segmente mit konstant vielen Fragen getestet werden müssen. Wenn wir nun garantieren können, daß die Länge des Beweises sehr klein wird (sagen wir in der Ordnung von $\mathrm{poly}\log\log(n)$ in der Eingabelänge n), dann erhalten wir statt dem ursprünglichen Aufwand von $\mathrm{poly}(n)$ Zufallsbits, einen Aufwand von $\mathrm{poly}\log\log n = O(\log n)$. Zur Präzisierung werden wir nun die Notation von Kodierungsschemata zur Hilfe nehmen:

Definition 11.3 (Kodierungsschema) *Ein $l(n)$-Kodierungsschema E_ϕ für ein zweistelliges boolesches Prädikat $\phi(,)$ hat zwei Eingaben x und y, und generiert ein Tripel (X, Y, π) mit der Eigenschaft*

- *X, Y, π sind Binärstrings,*
- *X ist nur von x und Y nur von y abhängig,*
- *π darf von x, y und ϕ abhängen,*
- *$|(X, Y, \pi)| = O(l(n))$, mit $n = |x| + |y|$.*

Definition 11.4 *Zwei Tripel (X_1, Y_1, π_1) und (X_2, Y_2, π_2) liegen ϵ-XY-nah aneinander, falls für die Hammingdistanzen von X_i bzw. Y_i gilt, daß*

$$d(X_1, X_2) \leq \epsilon |X_1| \text{ und } d(Y_1, Y_2) \leq \epsilon |Y_1|.$$

Definition 11.5 (Distanz eines Kodierungschema)
Ein Kodierungschema E_ϕ hat die Distanz ϵ, wenn für jede Eingabe $(x_1, y_1) \neq (x_2, y_2)$ gilt, daß $E_\phi(x_1, y_1)$ und $E_\phi(x_2, y_2)$ nicht ϵ-XY-nah sind.

Definition 11.6 (gutartige Kodierungen) *Ein $l(n)$-Kodierungsschema E_ϕ mit Distanz ϵ wird gutartig genannt, wenn ein randomisierter Algorithmus T mit folgenden Eigenschaften existiert:*

- *T untersucht nur konstant viele Bits in* (X', Y', π'),
- $[T(E_\phi(x,y)) = \mathit{akzeptiere}]$, *wenn* $\phi(x,y) = 1$,
- *Es existiert ein* $\delta > 0$, *so daß*

$$\Pr[T((X', Y', \pi')) = \mathit{verwerfe}] \geq \delta \text{ ,wenn}$$

  - $\nexists x, y \quad [E_\phi(x,y)$ *und* (X', Y', π') *sind* $\frac{\epsilon}{4} -$ *XY-nah*] *oder*
  - $\exists x, y \quad [E_\phi(x,y)$ *und* (X', Y', π') *sind* $\frac{\epsilon}{4} -$ *XY-nah*] $\wedge\, \phi(x,y) = 0$.

Die Basis

Satz 11.5 *Es gibt ein gutartiges* $(2^{\mathrm{poly}(n)})$*-Kodierungsschema* $E_\phi^{(0)}$ *für ein in polynomieller (deterministischer) Zeit auswertbares Pädikat* ϕ. *Es werden dabei* $r_0(n) := n^{O(1)}$ *Zufallsbits benötigt.*

Beweis:
Sei $\phi(x,y)$ gegeben. Aus Cooks Theorem [C71] folgt, daß

$$\phi(x,y) = 1 \Leftrightarrow \exists z, f[\ f \text{ ist eine 3SAT-Formel }] : f(x,y,z) = 1$$

Dabei folgt aus der Konstruktion von f, daß $|z| = O(p^2(n))$ und $|f| = O(p^{O(1)}(n))$, wenn $\phi(x,y)$ in Zeit $p(n) \in n^{O(1)}$ evaluiert werden kann ($n = |x| + |y|$).
Konstruktion der Kodierung $E_\phi^{(0)}(x,y) = (X, Y, \pi)$**:** Sei

$$X := (xu^T)_{u \in \mathrm{GF}(2)^{|x|}} \text{ und } Y := (yv^T)_{v \in \mathrm{GF}(2)^{|y|}}$$

Wenn $a = xyz$ eine erfüllende Belegung von f ist, dann sei π die Kodierung von a wie im Beweis von Satz 11.1 (*parity based encoding*). Aus der Konstruktion folgt, daß $|XY\pi| = O(2^{O(p^6(n))}) = O(2^{\mathrm{poly}(n)})$ gilt.
Ein Testalgorithmus für $E_\phi^{(0)}(x,y)$**:**

1. Teste, ob π die Kodierung einer erfüllenden Belegung von f ist. Wende dabei den Algorithmus aus Satz 11.1 an. Wir benötigen dabei nur konstant viele Nachfragen an π. Die Fehlerwahrscheinlichkeit ist durch ein δ_1 nach oben beschränkt.

2. Teste, ob X, Y Kodierungen von x, y sind, so daß xy Präfix von a ist. Da alle inneren Produkte xu^T aus X auch in π enthalten sind, verläuft der Test wie folgt:

   (a) Teste wie bei Satz 11.1, ob X δ'-nah an einer linearen Funktion liegt.

   (b) Wähle $u, r_1, r_2 \in_R \mathrm{GF}(2)^{|x|}$ und teste, ob

   $$x(r_1 - u)^T + xr_1^T = \tilde{A}(r_2 - u) + \tilde{A}(r_2);$$

   (c) Akzeptiere genau dann, wenn Test (a) und (b) erfolgreich waren.

   Der Test für y verläuft analog. Wie bei Satz 11.1 erhält man für die Fehlerwahrscheinlichkeit eine obere Schranke δ .

Es folgt nun, daß $E_\phi^{(0)}$ ein gutartiges $(2^{\mathrm{poly}(n)})$-Kodierungsschema ist und $r_0 = O(n^3)$ gilt. ∎

Die Rekursion

Wie einige Male zuvor:

Sei $\phi(x, y)$ gegeben. Aus Cooks Theorem folgt, daß

$$\phi(x, y) = 1 \Leftrightarrow \exists z, f[\ f \text{ ist eine 3SAT-Formel }] : f(x, y, z) = 1$$

Dabei folgt aus der Konstruktion von f, daß $|z| = O(p^2(n))$ und $|f| = O(p^{O(1)}(n))$, wenn $\phi(x, y)$ in Zeit $p(n) \in n^{O(1)}$ evaluiert werden kann ($n = |x| + |y|$).

Bemerkung 11.4 *Betrachte folgende Abstraktion des segmentierten Beweises aus Abschnitt 11.2:*

1. *Der Beweis besteht aus konstant vielen Tabellen $T_1, ..., T_c$;*

2. *Jede Tabelle T_j hat $\mathrm{poly}(n)$-viele Einträge der Länge $\mathrm{poly}\log(n)$;*

3. *Die Tabellen T_1, T_2 kodieren x bzw. y mit Hilfe der Low-Degree-Erweiterung der Zeichenketten x, y. Aus dem Lemma von Schwartz [S80b] folgt, daß für die Tabellen von zwei verschiedenen Strings x_1 und x_2 gilt, daß der Anteil der gleichen Einträge durch eine Konstante nach oben beschränkt ist. Für y gilt dasselbe.*

4. *Es existiert eine Verifikationsprozedur, die die Tabelle nur an konstant vielen (sagen wir c') Einträgen untersucht und mit durch eine Konstante beschränkter Fehlerwahrscheinlichkeit testen kann, ob*

   - *T_1, T_2, Belegungen x, y zulässig kodieren und*
   - *ob ein z existiert, so daß $f(x, y, z) = 1$ ist.*

   Die Anzahl der benötigten Zufallsbits ist dabei $O(\log n)$.

Konstruiere nun $E^{(i)}$ rekursiv mit Hilfe von $E^{(i-1)}$ wie folgt:

- Bilde eine Tabelle $T_{overall}$, indem man für alle möglichen Folgen von Zufallsbits, die in der Verifikationsprozedur von Abschnitt 11.2 auftreten können [$2^{O(\log n)}$-viele], die Konkatenation aller bei den jeweiligen Zufallsfolgen ausgelesenden Einträge aus T_i, ($i = 1, ..., c$) einträgt ;

- Bilde c' Vergleichstabellen T'_j, ($j = 1, ..., c'$), die für jede mögliche Zufallsfolge [$2^{O(\log n)}$-viele] *zwei* Einträge ($T'_j(r)_1, T'_j(r)_2$) hat:

  1. die (bei der jeweiligen Zufallswahl) vom Tester beim j-ten Test ausgelesenen Beweisteile;
  2. Konkatenation aller bei der jeweiligen Zufallswahl ausgelesenen Einträge des Beweises [Beachte $T'_j(r)_2 = T_{overall}(r)$] .

Jeder Eintrag wird nun mit dem Kodierungsschema $E_{\psi_j}^{(i-1)}$ wie folgt kodiert:

- Die Einträge von $T_k, (k = 1, ..., c)$ werden mit $E_{\psi_1}^{(i-1)}$, $\psi_1(\sigma, 0) = 0$ kodiert, für alle Einträge σ in T_k (Die zweite Stelle ist hier nutzlos; später wird nur die Kodierung von σ benötigt) ;
- Die Einträge von $T_{overall}$ werden mit $E_{\psi_2}^{(i-1)}$,

  $$\psi_2(0, \sigma) = 1 \Leftrightarrow \sigma \text{ überzeugt den Tester aus Abschnit 11.2}$$

  für alle Einträge σ in $T_{overall}$;
- Die Einträge von $T'_k, (k = 1, ..., c')$ werden mit $E_{\psi_3}^{(i-1)}$,

  $$\psi_3(\sigma_1, \sigma_2) = 1 \Leftrightarrow \sigma_1 = \text{ dem } j\text{-ten Teil von } \sigma_2$$

  für alle Paare (σ_1, σ_2) von Einträgen in T'_k.

$E_\phi^{(i)} = (X, Y, \pi)$ ist die Konkatenation aller kodierten Einträge von $T_1, ..., T_c, T_{overall}, T'_1, ..., T'_{c'}$, so daß

- $X =$ Kodierung von T_1 ;
- $Y =$ Kodierung von T_2 ;
- $\pi =$ Rest .

Satz 11.6 *$E_\phi^{(i)}$ ist gutartig für $i \geq 0$. (Mit ϕ ein in polynomieller (deterministischer) Zeit auswertbares Pädikat). Wenn l_i die Länge der Kodierung $E_\phi^{(i)}$ und r_i die Anzahl der benötigten Zufallsbits in der Verifikationsprozedur ist, dann gilt:*

1. $l_i(n) = \text{poly}(n) \cdot l_{i-1}(\text{poly} \log n)$
2. $r_i(n) = O(\log n) + O(r_{i-1}(\text{poly} \log n)) + O(\log(l_{i-1}(\text{poly} \log n)))$

BEWEIS: Wir haben in der Kodierung poly(n) Einträge der Länge $l_{i-1}(\text{poly} \log n) \Rightarrow$ Beh. (1). Den Rest durch Induktion über i:

$i = 0$: Folgt aus Satz 11.5 .

$i - 1 \to i$:

- Nach Induktionsvoraussetzung hat $E^{(i-1)}$ Distanz $\epsilon > 0$. Wegen Punkt 3 in Bemerkung 11.4 unterscheiden sich die jeweiligen Tabellen für $x_1 \neq x_2$ in mehr als $\epsilon'|T_1|$ Einträgen für eine Konstante $\epsilon' > 0$. Daraus folgt, daß die Kodierung von x eine Distanz $\geq \min\{\epsilon, \epsilon'\}$ hat. Analoges gilt für y.
- Betrachte folgende Verifikationsprozedur:

  Schritt 1: Wähle $r \in_R \text{GF}(2)^{O(\log n)}$.

  Schritt 2: Teste mit dem Tester für $E^{(i-1)}_{\psi_2}$ im r-ten Eintrag der Kodierung von $T_{overall}$, ob ψ_2 erfüllt ist.
  ($\Leftrightarrow \exists a = a_1 ... a_{c'}$ [diese c' Strings überzeugen den Verifizierenden aus Bem. 11.4])
  Der Aufwand an Query-Bits ist nach I.V. $O(1)$ [an Zufallsbits $r_{i-1}(\text{poly} \log n)$].
  Die Fehlerwahrscheinlichkeit ist nach I.V. durch eine Konstante δ_1 beschränkt.

  Schritt 3: Teste mit dem Tester für $E^{(i-1)}_{\psi_3}$ in der r-ten Zeile der Kodierung von $T'_k, (k = 1, ..., c')$, ob ψ_3 erfüllt ist ($\Leftrightarrow b_j$ ist j-ter Teilstring von b) .
  Der Aufwand an Query-Bits ist nach I.V. $O(1)$ [an Zufallsbits $O(r_{i-1}(\text{poly} \log n))$].
  Die Fehlerwahrscheinlichkeit sei (wie bei Schritt 1) durch ein δ_2 beschränkt.

  Schritt 4: Teste, ob b_j in T'_j mit a_j, das in der Familie (T_i) zu finden ist, übereinstimmt. Da $E^{(i-1)}$ nach Induktionsvoraussetzung Distanz ϵ hat, genügt es die Kodierung von a_j und b_j an $O(1)$ Stellen zu vergleichen: Annahme $a_j \neq b_j$, dann

$$\Pr_{k \in_R \{1, ..., l_{i-1}(\text{poly} \log n)\}} [\underbrace{E^{(i-1)}(b_j)_k}_{=:B_j} \neq \underbrace{E^{(i-1)}(a_j)_k}_{=:A_j}] = \frac{d(B_j, A_j)}{|B_j|} > \epsilon$$

  Der Aufwand an Query-Bits ist $O(1)$ [an Zufallsbits $O(\log(l_{i-1}(\text{poly} \log(n))))$.
  Die Fehlerwahrscheinlichkeit ist durch δ_3 beschränkt.

Schritt 5: Teste analog, ob a in $T_{overall}$ gleich b in $T'_j, (j = 1, ..., c')$ ist. Die Fehlerwahrscheinlichkeit ist durch δ_4 beschränkt.

Wenn alle Tests erfolgreich waren, so gilt mit Wahrscheinlichkeit $\geq 1 - (\sum_{i=1}^{4} \delta_i)$:

- Es existieren zwei Strings x, y, die zulässig kodiert sind ;
- x, y erfüllen ϕ .

Daraus folgt, daß $E^{(i)}$ eine gutartige Kodierung ist.

∎

Korollar 11.1 *Für $E_\phi^{(2)}$ gilt:*

1. $l_2(n) = \text{poly}(n)$
2. $r_2(n) = O(\log n)$

Beweis: Aus Satz 11.5 folgt, daß $l_0(n) = 2^{\text{poly}(n)}, r_0(n) \in \text{poly}(n)$ ist. Daraus folgt:

$$l_1(n) = \text{poly}(n) \cdot 2^{\text{poly} \log n} = O(2^{\text{poly} \log n})$$

Also ist $l_2(n) = \text{poly}(n) \cdot \underbrace{O(2^{\text{poly} \log \log n})}_{=O(2^{O(\log n)})} \in \text{poly}(n)$. Ferner gilt:

$$r_1(n) = O(\log n) + O(\text{poly} \log n) + O(\log 2^{\text{poly} \log n}) = \text{poly} \log n,$$

$$r_2(n) = O(\log n) + O(\text{poly} \log \log n) + O(\log 2^{\text{poly} \log \log n}) = O(\log n).$$

∎

Kapitel 12

Die Approximationskomplexität von Zählproblemen

In diesem Kapitel wollen wir uns einem ganz neuen Problemkreis zuwenden. Wir stellen die Frage nach der Anzahl von Lösungen zu gewissen Problemen. Diese Fragestellung erweitert das Entscheidungsproblem, bei dem nur nach der Existenz einer Lösung gefragt ist.

Zählprobleme lassen sich nicht direkt mit Entscheidungsproblemen vergleichen. So ist das Ergebnis eines Zählalgorithmus eine natürliche Zahl. In [V79a] wird ein geeignetes Berechnungsmodell für Zählprobleme eingeführt.

Definition 12.1 (Counting TM) *Eine Counting TM (kurz CTM) M ist eine nichtdeterministische Turing Maschine, die außerdem ohne zusätzlichen Berechnungsaufwand auf einem Extraband die Anzahl akzeptierender Berechnungsfolgen ausdruckt. Damit berechnet diese TM eine Funktion*

$$f_M : \Sigma^* \to I\!N.$$

Mit Hilfe dieser Definition können wir nun Komplexitätsklassen definieren.

Definition 12.2 ($\#\mathcal{P}$) *Mit $\#\mathcal{P}$ bezeichnen wir alle Probleme, die mit einer polynomiell zeitbeschränkten Counting TM berechnet werden können.*

Anhand der Definition ist klar, daß die zugehörigen Zählprobleme von Problemen aus NP in $\#P$ liegen. Wie bei den Entscheidungsproblemen interessieren uns auch hier die schwersten Probleme der Klasse. Dazu müssen wir zunächst einen geeigneten Reduktionsbegriff einführen.

Definition 12.3 (Orakel-Counting TM) *Eine Orakel-Counting TM mit dem Orakel X ist eine deterministische TM M, die einen speziellen Orakelzustand in ihrer endlichen Kontrolle besitzt. Geht M in diesen Zustand über, so wird der Inhalt eines speziellen Arbeitsbandes, des Orakelbandes, als Eingabe für das Orakel X verwendet, welches das Ergebnis, d.h. die Anzahl der Lösungen des Problems X auf das Orakelband schreibt. Mit diesem Ergebnis kann M dann weiterarbeiten. Wir bezeichen die Klasse der so in polynomiell beschränkter Zeit berechenbaren Probleme mit $\mathcal{P}^X$.*

Damit können wir eine geeignete Reduktion wie folgt definieren.

Definition 12.4 *Ein Zählproblem Y ist auf X reduzierbar, falls $Y \in \mathcal{P}^X$. Eine Zählproblem X ist $\#\mathcal{P}$ -hart, wenn*

$$\#\mathcal{P} \subseteq \mathcal{P}^{\mathcal{X}}$$

Ein Zählproblem X ist $\#\mathcal{P}$ -vollständig, wenn X $\#\mathcal{P}$ -hart ist und X in $\#\mathcal{P}$ liegt.

Kandidaten für $\#\mathcal{P}$-vollständige Zählprobleme sind Zählprobleme, die aus NP-vollständigen Entscheidungsproblemen hervorgehen. Ob alle diese Zählprobleme wirklich $\#\mathcal{P}$-vollständig sind, ist ein offenes Problem. Valiant gibt in [V79b] ein erstes nichttriviales $\#\mathcal{P}$-vollständiges Zählproblem an.

Satz 12.1 *Die Berechnung der Permanente einer $n \times n$ Matrix mit Einträgen aus $\{0, 1\}$, d.h. die Berechnung der Anzahl Perfekter Matchings eines bipartiten Graphen, ist $\#\mathcal{P}$ -vollständig.*

Dieser Satz ist sehr erstaunlich, da das zugehörige Entscheidungsproblem, d.h. der Test auf die Existenz von Perfekt Matching, in polynomieller Zeit möglich ist. Ausgehend von dem $\#\mathcal{P}$ -vollständigen Problem der Permanentenberechnung hat Valiant noch eine Reihe weiterer $\#\mathcal{P}$ -vollständiger Probleme gefunden (siehe [V79a]). Ein weiteres erstaunlicherweise $\#P$ -vollständiges Problem ist das *monotone 2-DNF Zählproblem*.

Definition 12.5 (monotone 2-DNF Zählproblem) *Gegeben sei eine boolesche Formel f in disjunktiver Normalform, so daß keine Klausel mehr als zwei Literale enthält. Außerdem gebe es keine negierten Literale, d.h. die Formel ist monoton. Bezeichne $\#f$ die Anzahl der erfüllenden Belegungen*

$$\#f := |\{\bar{x} \mid f(\bar{x}) = 1\}|.$$

Das monotone 2-DNF Zählproblem ist die Berechnung von $\#f$.

Satz 12.2 *Das monotone 2-DNF Zählproblem ist $\#\mathcal{P}$ -vollständig.*

Bemerkung 12.1 *Das monotone DNF Erfüllungsproblem ist trivialerweise in $\mathcal{P}$.*

Wir sehen also, daß viele wichtige und grundlegende Zählprobleme so schwer sind, daß wir keine effizienten polynomiellen Lösungen erwarten dürfen. Was kann uns aus diesem Dilemma retten?

- Einschränkungen auf spezielle Klassen von zu zählenden Objekten.
- Approximative anstatt exakter Lösungen.

Im Falle von DNF-Formeln zeigt Satz 12.2, daß Einschränkungen hier nicht unbedingt zum Ziel führen. Daher werden Algorithmen entwickelt, um die Anzahl erfüllender Lösungen einer booleschen Formel in disjunktiver Normalform approximativ zu bestimmen. Dafür müssen wir die Eigenschaften solcher approximativer Algorithmen genauer definieren.

Definition 12.6 (ϵ-Approximationsalgorithmus) *Ein Algorithmus A für das Zählproblem f ist ein ϵ-Approximationsalgorithmus, falls für die Ausgabe y von A*

$$(1-\epsilon)\cdot \#f \leq y \leq (1+\epsilon)\cdot \#f$$

gilt. Die Eingabegrößen des Algorithmus sind die Eingabegröße des Zählproblems f und der Kehrwert von ϵ.

Definition 12.7 ((ϵ,δ)-Approximationsalgorithmus) *Ein Algorithmus A für das Zählproblem f ist ein (ϵ,δ)- Approximationsalgorithmus, falls für die Ausgabe y von A*

$$\Pr[(1-\epsilon)\cdot \#f \leq y \leq (1+\epsilon)\cdot \#f] \geq 1-\delta$$

gilt. Die Eingabegrößen des Algorithmus sind die Eingabegröße des Zählproblems f, der Kehrwert von ϵ und $\log(1/\delta)$.

Einige Approximationsalgorithmen mit polynomieller Laufzeit haben wir bereits kennengelernt (Siehe auch Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther) [K91]). Wir beginnen im nächsten Abschnitt mit einem Härteresultat für das $\#2SAT$ Problem. Dazu wollen wir einen zu den bisherigen Approximationsalgorithmen verwandten Approximationsalgorithmus für Zählprobleme einführen.

Definition 12.8 *Ein Algorithmus A für das Zählproblem f ist ein Approximationsalgorithmus mit A.R. α, wenn für die Ausgabe y*

$$\max\{y/\#f, \#f/y\} \leq \alpha$$

gilt.

In Abschnitt 12.2 entwickeln wir einen (ϵ,δ)-Approximationsalgorithmus für das DNF-Zählproblem. Er ist der Arbeit [KLM89] entnommen.

12.1 Approximationshärte von $\#2SAT$

Wir widmen uns jetzt dem Status von $\#2SAT$ und der äquivalenten Zählprobleme $\#IS$ und $\#CLIQUE$. Es gilt

Satz 12.3 ([GJ79], S. $260 - 261$) $2SAT \in P$.

Ist die Zählfunktion $f : \{0,1\}^* \longrightarrow I\!N$ für $\#2SAT$ auch in P? Die Antwort ist **Nein**, wie folgender Satz zeigt.

Satz 12.4 ($\#2SAT$) *Die Approximation von* $\log(\#2SAT)$ *mit A.R.* $O(n^\epsilon)$ *ist* NP*-hart, für ein* $\epsilon > 0$.

Wir werden eine Reduktion von $MAX\text{-}CLIQUE$ auf $\#2SAT$ konstruieren und dabei das Approximationshärteresultat aus Kapitel 10 verwenden:

Beweis: Gegeben sei ein Graph $G = (V, E)$ mit Knotenmenge $V = \{1, 2, \ldots, n\}$. Wir konstruieren eine $2 - KNF$-Formel $f = \bigwedge_{\{i,j\} \notin E} (\overline{x_i} \vee \overline{x_j})$, $(\overline{x} := \neg x)$. Für eine Belegung $s \in \{0,1\}^n$ von f definieren wir $V_s = \{i | s_i = 1\}$.

Bevor wir den Beweis fortsetzen benötigen wir die folgenden Lemmata:

Lemma 12.1 $f(s) = 1 \Longleftrightarrow V_s$ *ist eine Clique von* G

Beweis: ($\Longleftarrow$) V_s ist eine Clique von G. Für alle $\{i,j\} \notin E$ gilt $\{i,j\} \not\subseteq V_s$. Daher ist $s_i = 0 \vee s_j = 0$ und somit $f(s) = 1$.

($\Longrightarrow$) Sei $f(s) = 1$. Angenommen $\{i,j\} \notin E$ und $s_i = 1$ und $s_j = 1$.
Dann gilt: $(\overline{x_i} \vee \overline{x_j})(s) = 0 \Longrightarrow f(s) = 0$ (Widerspruch) ■

Aus diesem Lemma folgt: Wenn $f(s) = 1$ und $i, j \in V_s$, dann ist auch $\{i,j\} \in E$. V_s ist eine Clique von G.

Lemma 12.2 *Wenn MAX-$CLIQUE$ $(G) = k$, dann gilt*

$$O(n^k) \geq \#f = |\{s|f(s) = 1\}| \geq 2^k$$

Beweis: Via Lemma 12.1: Die Anzahl der Cliquen von G ist gleich $\#f$. Da die Größe jeder Clique in G höchstens k ist, ist die Anzahl der Cliquen von G

$$\#f \leq [\tbinom{n}{k} + \tbinom{n}{k-1} + \ldots + \tbinom{n}{0}] = O(n^k)$$

Sei C die MAX-$CLIQUE$ von G. Dann ist auch jede Clique $C' \subseteq C$ eine Clique von G. Also ist die Anzahl der Cliquen von G mindestens 2^k.

■

Mit den beiden Lemmata können wir den Beweis von Satz($\#2SAT$) jetzt fortsetzen:

$$\begin{aligned}
2^k &\leq \#f \leq O(n^k) \\
\Longleftrightarrow k &\leq \log(\#f) \leq O(k \log n) \\
\Longrightarrow \frac{\log(\#f)}{\log n} &\leq O(k) \leq O(\log(\#f))
\end{aligned}$$

Vorausgesetzt $\tilde{y}$ approximiere $\log(\#f)$ mit $A.R.\ n^{\epsilon'}$. Dann gilt

$$\frac{\tilde{y}}{\log(\#f)} \leq n^{\epsilon'} \qquad (12.1)$$

$$\frac{\log(\#f)}{\tilde{y}} \leq n^{\epsilon'} \qquad (12.2)$$

Aus (12.1) folgt

$$\begin{aligned}
\frac{\tilde{y}}{n^{\epsilon'}} &\leq \log(\#f) \\
\frac{\tilde{y}}{n^{\epsilon'} \log n} &\leq O(k) \\
\frac{\tilde{y}}{k} &\leq n^{\epsilon'} \log n
\end{aligned}$$

Aus (12.2) erhalten wir

$$\frac{k}{\tilde{y}} \leq O(n^{\epsilon'}).$$

Als Ergebnis erhalten wir eine $O(n^{\epsilon'} \log n)$-Approximation von k und folglich auch eine $O(\log n)$-Approximation für $\log(\#2SAT)$. Die Lösung für $\log(\#2SAT)$ impliziert also eine $O(n^{\epsilon'} \log n)$-Approximation von $MAX\text{-}CLIQUE$. Durch geeignete Wahl von $\epsilon > \epsilon'$, erhalten wir einen Widerspruch mit Hauptsatz 10.2. ■

Korollar 12.1 *Es gibt ein $\epsilon > 0$, so daß die Approximation von $\#2SAT$, $\#IS$ und $\#CLIQUE$ mit A.R. 2^{n^ϵ} NP-hart ist.*

Beweis: Via Beweis von Satz 12.4 und Hauptsatz 10.2. Angenommen $\tilde{y}$ ist eine polynomielle Approximation von $\#f$ mit $A.R.$ 2^{n^ϵ}, für alle $\epsilon > 0$. Dann gilt:

$$\frac{\tilde{y}}{\#f} \leq 2^{n^\epsilon} \tag{12.3}$$

$$\frac{\#f}{\tilde{y}} \leq 2^{n^\epsilon} \tag{12.4}$$

Durch Logarithmieren von (12.3) und Teilen durch $\log(\#f)$ erhalten wir

$$\frac{\log(\tilde{y})}{\log(\#f)} - 1 \leq n^\epsilon$$

Für (12.4) existiert eine analoge Umformung.

Mit Satz($\#2SAT$) folgt ein Widerspruch. ■

Mit $\#kIS$ bezeichnen wir die Einschränkung von $\#IS$ auf Graphen mit Maximalgrad k. Dyer et al. [DFJ98] haben gezeigt, daß für $\#kIS$ (mit $k \geq 25$) wahrscheinlich kein $PTAS$ existieren kann, während $\#4IS$ einen $PTAS$ besitzt [LV97]. Somit haben wir die offene Frage, was für $4 < k < 25$ gilt.

12.2 (ϵ, δ)-Approximationsalgorithmus für #DNF

In diesem Abschnitt werden wir einen probabilistischen Zählalgorithmus kennenlernen. Das Prinzip dieses Algorithmus ist einfach:

Sei $G = \{\bar{x} \mid f(\bar{x}) = 1\}$ und $U \supseteq G$. Der Algorithmus besteht aus N unabhängig ausgeführten Versuchen. Jeder dieser Versuche besteht aus den folgenden Schritten.

1. Wähle zufällig ein Element $\tilde{x}$ aus U. Zufällig bedeutet dabei unabhängig und gemäß einer uniformen Verteilung.

2. Teste ob $\tilde{x} \in G$, d.h. ob $f(\tilde{x}) = 1$.

3. Setze die Ausgabe
$$Y := \begin{cases} |U| & \text{falls } f(\tilde{x}) = 1 \\ 0 & \text{sonst .} \end{cases}$$

Damit ist das Ergebnis eines Schrittes eine Zufallsvariable Y. Der Erwartungswert von Y ist

$$E[Y] = \frac{1}{|U|} \sum_{x \in U} |U| \cdot f(x) = \sum_{x \in G} f(x) = |G|.$$

Damit gilt auch für die Ausgabe $\tilde{Y} = \frac{1}{N} \sum_{i=1}^{N} Y_i$ des Zählalgorithmus

$$E[\tilde{Y}] = |G|,$$

d.h. der Mittelwert der Ergebnisse der einzelnen Versuche ist eine geeignete Zufallsvariable für die Anzahl der erfüllenden Belegungen. Satz 3.1 (Satz von Bernstein) gab Auskunft über die minimale Anzahl von Versuchen, so daß diese Zufallsvariable eine (ϵ, δ)-Approximation darstellt:

Sei $\mu = \frac{|G|}{|U|}$. Dann ist der obige Algorithmus ein (ϵ, δ)-Approximationsalgorithmus, wenn die Anzahl der Versuche N größer gewählt wird als

$$\frac{1}{\mu} \cdot \frac{4}{\epsilon^2} ln(\frac{2}{\delta}).$$

Der Beweis dieses Satzes ist in Kapitel 3 und im Vorlesungsskript *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen)* von M. Karpinski (ausgearbeitet von K. Werther) [K91] zu finden.

Im folgenden sei G nun die Menge aller Einsstellen von f. Nun wollen wir versuchen eine geeignete Obermenge U zu konstruieren. Diese Obermenge U muß die folgenden Eigenschaften besitzen, damit der Satz 3.1 von Bernstein eine polynomielle Laufzeit garantiert.

- $|U|/|G|$ muß polynomiell in der Eingabegröße sein.
- Der Wert $|U|/|G|$ muß in polynomieller Zeit berechenbar sein.
- Es muß in polynomieller Zeit möglich sein, Elemente von U gleichverteilt zu wählen.

Die Menge $U = \{0,1\}^n$ aller Vektoren der Länge n ist nicht geeignet, da der Quotient $|U|/|G|$ für DNF-Formeln mit nur wenigen erfüllenden Belegungen nicht polynomiell beschränkt ist. Die Menge U muß also kleiner gewählt werden. Um diese Wahl durchzuführen, beachten wir, daß f in disjunktiver Normalform gegeben ist

$$f = c_1 \vee c_2 \vee \cdots \vee c_m.$$

f wird also genau dann erfüllt, wenn mindestens eine Klausel c_i erfüllt wird. Bezeichne D_i die Menge der Erfüllenden für die Klausel c_i

$$D_i = \left\{ (\bar{x}_1 \dots \bar{x}_n) \mid \bar{x}_k = \begin{cases} 1 & x_k \text{ kommt als positives Literal vor} \\ 0 & x_k \text{ kommt als negatives Literal vor} \\ 1 \text{ oder } 0 & x_k \text{ kommt nicht vor} \end{cases} \right\}.$$

Die Kardinalität von D_i ist $|D_i| = 2^{n - \#\text{Variablen in } c_i}$. Außerdem gilt

$$G = \bigcup_{i=1}^{m} D_i,$$

d.h. G ist die Vereinigung von bekannten Mengen. Mit Hilfe der D_i's können wir jetzt die Kardinalität von G beschränken:

$$\max_{1\leq i\leq m} |D_i| \leq |G| \leq \sum_{i=1}^{m} |D_i| \leq m \max_{1\leq i\leq m} |D_i| \leq m|G|.$$

Wir wählen also U als die direkte Summe der D_i. $U = \bigoplus_{i=1}^{m} D_i$ ist eine Obermenge der Menge $\tilde{G} = \{(i,s) \mid c_i(s) = 1 \text{ und } c_j(s) = 0 \text{ für alle } j < i\}$, die offensichtlich gleichmächtig zu G ist. Die Menge U erfüllt auch die Bedingungen des Satzes von Bernstein, da das Verhältnis $|U|/|\tilde{G}|$ durch m beschränkt ist. Wir können die Menge $\tilde{G}$ auch als Menge der Einsstellen der folgenden Funktion interpretieren:

$$\varphi(s,i) = \begin{cases} 1 & i \text{ ist die kleinste Nummer einer Klausel mit } c_i(s) = 1 \\ 0 & \text{sonst.} \end{cases}$$

Die Definitionsmenge von φ ist gerade U und $\tilde{G}$ die Menge der Einsstellen von φ. Es sei bemerkt, das wir für die Korrektheit des Algorithmus nur die Eigenschaft von φ benötigen, daß $\sum_{i=1}^{m} \varphi(s,i) = 1$ für alle $s \in \tilde{G}$ gilt. Die ersten beiden Anforderungen für den (ϵ,δ)-Approximationsalgorithmus haben wir erfüllt. Nun müssen wir noch zeigen, wie Elemente aus U mittels einer Gleichverteilung gewählt werden können. Dies geschieht folgendermaßen:

- Zuerst wähle eine Klausel mit Wahrscheinlichkeit $|D_i|/|U|$.
- Wähle eine erfüllende Belegung dieser Klausel mit Wahrscheinlichkeit $1/|D_i|$.

Mit Hilfe der obigen Wahl wird jedes Element von U mit gleicher Wahrscheinlichkeit $1/|U|$ gewählt. Damit haben wir einen (ϵ,δ)-Approximationsalgorithmus gewonnen.

Algorithmus 3 (ϵ,δ)-Approximationsalgorithmus

Eingabe: DNF-Formel f mit m Termen, Fehlertoleranz ϵ, Fehlerwahrscheinlichkeit δ.

Schritt 1: Berechne die Anzahl der erfüllenden Belegungen $|D_k|$ der k-ten Klausel.

Schritt 2: Berechne $A_i = \sum_{k=1}^{i} |D_k|$ für $0 \leq i \leq m$

Schritt 3: Setze $\tilde{Y} := 0$.

Schritt 4: Führe die folgende Schleife $N = m \cdot \frac{4\ln(2/\delta)}{\epsilon^2}$ mal durch

a) Wähle i mit Wahrscheinlichkeit $|D_i|/|U|$. Dies geschieht durch Wahl von $j \in \{1 \dots |U|\}$ mit Wahrscheinlichkeit $1/|U|$ und binäre Suche des i's mit $A_{i-1} < j \leq A_i$.

b) Wähle erfüllende Belegung s der Klausel c_i, d.h s mit $c_i(s) = 1$.

c) Setze $k := 0$.

d) Suche die erste Klausel c_k mit $c_k(s) = 1$.

e) Wenn $k = i$ ist, so setze $Y := |U|$, sonst setze $Y := 0$.

f) Setze $\tilde{Y} := \tilde{Y} + Y$.

Ausgabe: $\tilde{Y}/N$.

Bemerkung 12.2 *Die Schritte 4 a) und 4 b) wählen ein $(i, s) \in U$ mit Wahrscheinlichkeit* $1/|U|$.

Satz 12.5 *Der obige Algorithmus ist ein (ϵ, δ)-Approximationsalgorithmus und hat eine Laufzeit von*

$$O\left(\frac{nm^2 \ln(1/\delta)}{\epsilon^2}\right).$$

Beweis: Der Satz 3.1 von Bernstein sichert uns bei richtig gewähltem N die (ϵ, δ)-Eigenschaft, falls in Schritt 4 e) die Zufallsvariable Y den Erwartungswert $E[Y] = |G|$ hat. Sei $\operatorname{cov}(s) = \{i \mid c_i(s) = 1\}$ die Menge aller Klauseln, die von s erfüllt werden. Dann gilt

$$\sum_{i \in \operatorname{cov}(s)} \varphi(s, i) = 1,$$

da es für jedes $s \in G$ nur ein i mit $\varphi(s, i) = 1$ gibt. Daraus folgt, daß $E[Y] = \sum_{(s,i)} \varphi(s, i) = \sum_s \sum_{i \in \operatorname{cov}(s)} \varphi(s, i) = \sum_s 1 = |G|$ ist. Daher liefert der Algorithmus eine (ϵ, δ)-Approximation.

Die Laufzeit ergibt sich aus der Anzahl der Schleifendurchläufe von Schritt 4 und dem teuersten Schritt in dieser Schleife. Dies ist die Bestimmung des i^*, für welches $\varphi(i^*, s) = 1$ gilt. Dazu muß für jedes $i \leq i^*$ überprüft werden, ob s die Klausel c_i erfüllt. Da jede solche Überprüfung $O(n)$ Zeit kostet und das gesuchte i^* von der Größenordnung $O(m)$ ist, ergibt sich die obige Laufzeit des Algorithmus. ■

Mit dem Algorithmus 3 haben wir einen (ϵ, δ)-Approximationsalgorithmus gewonnen. Dabei ist die Berechnung der Funktion $\varphi(i, s)$ mit einer Laufzeit von $O(nm)$ sehr teuer. Für die Korrektheit des Algorithmus benötigen wir nur die Eigenschaft

$$\sum_{i \in \mathrm{cov}\,(s)} \varphi(s, i) = 1 \qquad (*).$$

Wir werden nun eine randomisierte Funktion φ' konstruieren, die die Eigenschaft $(*)$ besitzt, deren Berechnungzeit im Erwartungswert aber wesentlich günstiger ist.

12.2.1 Selbstjustierender Zählalgorithmus

Wir werden die Funktion φ des Zählalgorithmus durch eine Zufallsvariable φ' ersetzen, die nur von der erfüllenden Belegung s abhängig ist und für jedes s den Erwartungswert $E[\varphi'(s)] = 1/|\mathrm{cov}\,(s)|$ hat. Damit ergibt sich, daß φ' die Bedingung $(*)$ erfüllt. Ein einzelner Versuch sieht dann genauso aus wie oben:

- Wähle $(i, s) \in U$.
- Berechne $\varphi'(s)$.
- Setze $Y = |U| \cdot \varphi'(s)$.

Für den Erwartungswert von Y gilt

$$E[Y] = \sum_{s} \sum_{i \in \mathrm{cov}\,(s)} \varphi'(s) = |G|.$$

Damit ist der obige Versuch für einen (ϵ, δ)-Approximationsalgorithmus geeignet. Die Berechnung von $\varphi'(s)$ geschieht folgendermaßen.

1.) $t(s) := 0$

2.) wiederhole bis $s \in D_j$

  a) $t(s) := t(s) + 1$

  b) wähle $j \in 1 \ldots m$ mit Wahrscheinlichkeit $1/m$.

3.) $\varphi'(s) := t(s)/m$

Die Zeit zur Berechnung der Zufallsvariable $\varphi'(s)$ ist also auch eine Zufallsvariable.

Lemma 12.3 *Der Erwartungswert für $t(s)$ ist*

$$E[t(s)] = \frac{m}{|\mathrm{cov}\,(s)|}.$$

Beweis: $t(s)$ ist eine geometrisch verteilte Zufallsvariable mit Erfolgswahrscheinlichkeit $|\mathrm{cov}\,(s)|/m$. □

Damit ergibt sich $E[\varphi'(s)] = 1/|\mathrm{cov}\,(s)|$. Also ist die obige Prozedur für einen (ϵ, δ)-Approximationsalgorithmus geeignet.

Nun wollen wir den durchschnittlichen Zeitverbrauch der obigen Prozedur untersuchen. Dazu sei eine Aktion eine Schleifeniteration der Prozedur. Gesucht ist also der Erwartungswert für die Anzahl t von ausgeführten Iterationen pro Versuch. Die Kosten für jede dieser Aktion kann durch die Kosten für die zufällige Wahl von $(i, s) \in U$, der zufälligen Wahl von $j \in \{1 \ldots m\}$ und des Testes, ob $s \in D_j$ ist, abgeschätzt werden. Im Falle des DNF-Zählproblems sind diese Kosten $O(n + \log|U| + \log m)$. Die Anzahl der Klauseln m ist durch 2^n beschränkt, die Größe von U durch $m2^n$. Im allgemeinen kann also $\log|U|$ größer sein als $O(n)$. Wir betrachten jedoch nur Eingaben, für die U klein, also $|U| = O(n)$ ist. Für die anderen Eingaben ist $|U|/|G|$ für $U = \{0,1\}^n$ und $G = \{s \mid f(s) = 1\}$ polynomiell beschränkt. Daher sind die Kosten, die innerhalb einer Schleifeniteration auftreten, von der Größenordnung $O(n)$.

Lemma 12.4 *Sei $\mu = |G|/|U|$. Dann ist*

$$E[t] = m\mu.$$

Beweis: Es gilt

$$\begin{aligned} E[t] &= \frac{1}{|U|} \sum_{(s,i)\in U} E[t(s)] \\ &= \frac{1}{|U|} \sum_{s\in G} \sum_{i\in \operatorname{cov}(s)} E[t(s)] \\ &= \frac{1}{|U|} \sum_{s\in G} m = m\mu. \end{aligned}$$

■

Damit haben wir einen (ϵ, δ)-Approximationsalgorithmus. Angewandt auf das DNF-Zählproblem ist er aber leider nicht schneller als der alte Algorithmus, da wir $E[t]$ nur durch m abschätzen können. Daher beträgt auch hier die Laufzeit $O(nm^2)$. Wir können aber trotzdem Vorteil aus der neuen Berechnung von φ' ziehen.

Nehmen wir an, wir könnten μ berechnen und folglich die Anzahl der Versuche mit

$$N(\mu) = \frac{c\ln(1/\delta)}{\mu\epsilon^2}$$

angeben, wobei c eine geeignete Konstante ist. Dann ist die Anzahl der insgesamt ausgeführten Aktionen $T(\mu)$ eine Zufallsvariable mit Erwartungswert

$$E[T(\mu)] = N(\mu) \cdot E[t] = \frac{cm\ln(1/\delta)}{\epsilon^2}.$$

Dieser Erwartungswert ist um den Faktor m geringer als die obere Schranke, die wir berechnet haben. Man beachte, daß $E[T(\mu)]$ von μ unabhängig ist.

Sei $T = cm\ln(1/\delta)\frac{1}{\epsilon^2}$ mit Konstante c. Wir werden den Algorithmus nun T Aktionen lang ausführen. Die Anzahl der Versuche, die während dieser Aktionen beendet wurden, ist nun eine Zufallsvariable N_T in Abängigkeit von T. Wir werden zeigen, daß für geeignet gewähltes c mit hoher Wahrscheinlichkeit genügend Versuche durchgeführt worden sind. Intuitiv wird dieses deutlich, da der Erwartungswert von N_T ungefähr gleich $T/E[t] = N(\mu)$ ist. Der Algorithmus justiert sich also selbst, da sowohl Versuche mit wenigen als auch mit vielen Aktionen ausgeführt werden. Bei einer hinreichend großen Anzahl von Versuchen wird die Anzahl von ausgeführten Aktionen mit hoher Wahrscheinlichkeit sehr nahe an diesem Erwartungswert liegen. Damit sieht unser selbstjustierender Zählalgorithmus wie folgt aus.

Algorithmus 4 Selbstjustierender Zählalgorithmus

Eingabe: DNF-Formel f mit m Termen, Fehlertoleranz ϵ, Fehlerwahrscheinlichkeit δ

Schritt 1: Setze $G := 0$, $N_T := 0$,

Schritt 2: Setze $T := 8(1+\epsilon)m\ln(2/\delta)/\epsilon^2$.

Schritt 3: Führe die Schleife durch bis $G > T$

- a) Wähle (s, i) zufällig aus U.
- b) $t(s) := 0$.
- c) Wiederhole bis $s \in D_j$
  - i) Wenn $G > T$ gehe zur Ausgabe.
  - ii) Wähle zufällig $j \in \{1 \dots m\}$ mit Wahrscheinlichkeit $1/m$.
  - iii) $t(s) := t(s) + 1$.
  - iv) $G := G + 1$.
- d) Setze $N_T := N_T + 1$
- e) Setze $\varphi'(s) := t(s)/m$
- f) Setze $Y_{N_T} := |U|\varphi'(s)$

Schritt 4: $\tilde{Y} := \frac{1}{N_T}\sum_{i=1}^{N_T} Y_i$

Ausgabe: $\tilde{Y}$

Dieser Algorithmus hat die (ϵ, δ)-Eigenschaft und Laufzeit $O(nm)$, da eine Aktion in $O(n)$ Zeit durchführbar ist.

Satz 12.6 *Sei*

$$\tilde{Y} = \frac{T \cdot |U|}{mN_T}$$

die Schätzung des Algorithmus. Diese Schätzung ist eine (ϵ, δ)-Approximation, falls bei $\epsilon < 1$

$$T = \frac{8(1+\epsilon)m\ln(2/\delta)}{\epsilon^2}$$

gewählt wird.

Zum Beweis dieses Satzes müssen wir wieder ein wenig Vorarbeit leisten. Für $k = 1, \ldots, m$ sei

$$R_k = \{s \in G \mid |\mathrm{cov}\,(s)| = k\}$$

und $r_k = |R_k|$. Dann ist $\sum_{k=1}^m r_k = |G|$ und $\sum_{k=1}^m k r_k = |U|$. Die Wahrscheinlichkeit, daß ein zufällig gewähltes Element aus G in R_k liegt, ist damit $kr_k/|U|$, d.h. die Wahrscheinlichkeit für alle $s \in R_k$ ist gleichgroß und nur von k abhängig. Daher können wir die Zufallsvariablen τ_k definieren, die die gleiche Verteilung haben wie $t(s)$ für $s \in R_k$. Auch diese Variablen haben eine geometrische Verteilung

$$Pr[\tau_k = j] = k/m(1 - k/m)^{j-1}$$

mit Erwartungswert m/k. Für $d = \lambda/m$ mit $\lambda \leq 1/2$ ergibt sich

$$\begin{aligned} E[e^{d\tau_k}] &= \sum_{j=1}^{\infty} e^{dj} Pr[\tau_k = j] \\ &= e^d \frac{k}{m} \sum_{j=1}^{\infty} (e^d(1 - k/m))^{j-1} \\ &= \frac{e^d k}{m(1 - (1 - k/m)e^d)}. \end{aligned}$$

Die Reihe konvergiert, da $(1 - k/m)e^d < 1$ für $d \leq 1/2m$ gilt.

Im folgenden seien $t_1, t_2, \ldots$ identisch verteilte Zufallsvariablen mit der gleichen Zufallsverteilung wie t. Die Zufallsvariable t_i ist die Anzahl der Aktionen im i-ten Versuch. Sei $S_l = \sum_{i=1}^l t_i$ die Anzahl von Aktionen nach dem l-ten Versuch und N_i die Anzahl der Versuche, die nach i Aktionen beendet sind. Es gilt nach Definition

$$N_i < l \iff S_l > i.$$

Lemma 12.5 *Sei* $0 \leq \lambda \leq 1/2$ *und* $d = \lambda/m$. *Dann gilt*

$$E[e^{dt}] \leq e^{(\lambda + 2\lambda^2)\mu} = e^{(md + 2(md)^2)\mu}.$$

Beweis: Es gilt

$$E[e^{dt}] = \frac{1}{|U|}\sum_{k=1}^{m} kr_k E[e^{d\tau_k}].$$

Wir kennen den Erwartungswert von $e^{d\tau_k}$. Wir setzen ihn in die Gleichung ein und erhalten

$$E[e^{dt}] = \frac{1}{|U|}\sum_{k=1}^{m} \frac{k^2 r_k}{m(e^{-d}-1+k/m)}.$$

Wir können nun die Summanden nach oben abschätzen. Dazu benutzen wir die Ungleichungen $1-d \le e^{-d}$ und daraus folgend $k/m - d \le e^{-d} - 1 + k/m$. Damit ergibt sich

$$\frac{k}{m(e^{-d}-1+k/m)} \le \frac{k}{m(k/m-d)} = 1 + \frac{d}{k/m-d}.$$

Setzen wir diese Abschätzung in die Ungleichung ein, erhalten wir

$$E[e^{dt}] \le 1 + \frac{1}{|U|}\sum_{k=1}^{m} \frac{kr_k d}{k/m-d}.$$

Aufgrund von $d = \lambda/m$ und $k \ge 1$ ergibt sich $k/m - d \ge \frac{k(1-\lambda)}{m}$. Also folgt

$$E[e^{dt}] \le 1 + \frac{1}{|U|}\sum_{k=1}^{m} \frac{r_k\lambda}{1-\lambda}.$$

Nun benutzen wir die Ungleichung $1/(1-\lambda) \le 1 + 2\lambda$ für $0 \le \lambda \le 1/2$. Außerdem verwenden wir die Definition der r_k mit der Gleichheit $\sum r_k = |G|$. Es folgt

$$E[e^{dt}] \le 1 + \frac{|G|}{|U|}\lambda(1+2\lambda) = 1 + \mu(\lambda + 2\lambda^2) \le e^{\mu(\lambda+2\lambda^2)}.$$

■

Aus diesem Lemma können wir das folgende Korollar ableiten.

Korollar 12.2 *Sei $\epsilon \le 2$. Dann gilt*

$$Pr[S_l > (1+\epsilon)m\mu l] \le e^{-\mu\epsilon^2 l/8}.$$

Beweis: Nach Lemma 3.4 gilt

$$Pr[S_l > (1+\epsilon)m\mu l] \le E[e^{d(S_l-(1+\epsilon)m\mu l)}]$$

und mittels Lemma 3.3 folgt

$$= E[e^{dt}]^l e^{-d(1+\epsilon)m\mu l}.$$

Dabei bezeichne t die Zufallsvariable mit der gleichen Verteilung wie die Zufallsvariablen t_i. Aus Lemma 12.5 ergibt sich eine Abschätzung für $E[e^{dt}]$

$$E[e^{dt}] \leq e^{(md+2(md)^2)\mu}.$$

Setzen wir nun für $d = \epsilon/4m$ ein, so folgt

$$\begin{aligned} Pr[S_l > (1+\epsilon)m\mu l] &\leq e^{(\epsilon/4+\epsilon^2/8)\mu l - \epsilon/4(1+\epsilon)\mu l} \\ &= e^{-\mu\epsilon^2 l/8}. \end{aligned}$$

■

Eine ähnliche Abschätzung gilt auch in die andere Richtung.

Lemma 12.6 *Sei* $0 \leq \lambda \leq 1/2$ *und* $d = \lambda/m$. *Dann gilt*

$$E[e^{-dt}] \leq e^{(-\lambda-\lambda^2)\mu} = e^{(-md-(md)^2)\mu}.$$

Korollar 12.3 *Sei* $\epsilon \leq 2$. *Dann gilt*

$$Pr[S_l > (1-\epsilon)m\mu l] \leq e^{-\mu\epsilon^2 l/8}.$$

Nun können wir die Korrektheit des selbstjustierenden Zählalgorithmus beweisen.

Beweis: von Satz 12.6

Sei

$$k_1 = \frac{8(1+\epsilon)\ln(2/\delta)}{\mu\epsilon^2(1+\epsilon)}$$

und

$$k_2 = \frac{8(1+\epsilon)\ln(2/\delta)}{\mu\epsilon^2(1-\epsilon)}.$$

Damit ist $T = k_1 m\mu(1+\epsilon) = k_2 m\mu(1-\epsilon)$. Wenn $k_1 \leq N_T \leq k_2$ gilt, so gilt auch

$$\frac{T}{k_2} \leq \frac{T}{N_T} \leq \frac{T}{k_1}$$

und damit
$$\frac{T|U|}{mk_2} \leq \tilde{Y} \leq \frac{T|U|}{mk_1}.$$
Nach Einsetzen von k_1 und k_2 ergibt sich
$$|G|(1-\epsilon) \leq \tilde{Y} \leq |G|(1+\epsilon)$$
und somit ist die Schätzung eine ϵ-Approximation. Es reicht also zu zeigen, daß
$$Pr[N_T > k_2] + Pr[N_T < k_1] \leq \delta$$
ist. Dies ergibt sich aber unmittelbar durch Benutzen der Korollare 12.2 und 12.3:
$$Pr[N_T < k_1] = Pr[S_{k_1} > T] = Pr[S_{k_1} > k_1 m\mu(1+\epsilon)] \qquad (*).$$
Durch Anwenden von Korollar 12.2 und Einsetzen von k_1 ergibt sich
$$(*) \leq e^{-\mu\epsilon^2 k_1/8} = e^{\ln(2/\delta)} = \delta/2.$$
Analoges gilt für den anderen Teil. ■

12.3 GF(q) Zählprobleme

Im vorigen Abschnitt haben wir das Problem untersucht, die Anzahl der Einsstellen einer booleschen Funktion zu berechnen, die in disjunktiver Normalform gegeben ist. Nun stellen wir die analoge Frage, was passiert, wenn die Funktion als Polynom über GF(2) gegeben ist. Diese Fragestellung läßt sich verallgemeinern auf die Frage der c-Stellen von Polynomen über beliebigen endlichen Körpern. Dies ist eine sehr wichtige Frage in der Geometrie, Algebra und algebraischen Geometrie.

12.3.1 Approximative Zählung der Einsstellen von GF(2) Polynomen

In diesem Abschnitt werden wir zeigen, daß wir den Approximationsalgorithmus für das DNF Zählproblem auch für das GF(2) Zählproblem verwenden können. Dazu werden wir den folgenden Satz beweisen.

Satz 12.7 ((**[KL93]**)) *Sei $C = \{c_i\}$ eine beliebige Menge von m paarweise verschiedenen, nichtleeren Klauseln über n Variablen. Bezeichne G die Menge der Einsstellen der Funktion $g = \bigvee c_i$ und F die Menge der Einsstellen der Funktion $f = \bigoplus c_i$, d.h wir sehen die Menge C einmal als Monome einer Funktion in diskunktiver Normalform und einmal als Monome eines Polynoms über* GF(2) *an. Dann gilt*

$$|G|/|F| \leq m.$$

Sei U die in Abschnitt 12.2 definierte Überdeckung von G. Dann gilt für das Verhältnis von $|U|$ und $|G|$

$$\frac{|U|}{|G|} \leq m.$$

Mit Hilfe von Satz 12.7 erhalten wir dann

$$\frac{|U|}{|F|} \leq m^2.$$

Diese Aussagen implizieren direkt den folgenden

Satz 12.8 *Es gibt einen (ϵ, δ)-Approximationsalgoritmus für das $GF(2)$ Zählproblem, der in Zeit*

$$O(nm^3 \ln(1/\delta)/\epsilon^2)$$

läuft.

Sei $X = \{x_1, \ldots, x_k\}$ eine Menge von Variablen und s eine Belegung der Variablen $\{x_1, \ldots x_n\}$. Dann definieren wir für jede Teilmenge X' von X die Belegung $s(X')$ durch

$$s(X')_i := \begin{cases} 0 & \text{falls } x_i \in X \setminus X' \\ s_i & \text{sonst} \end{cases}.$$

Lemma 12.7 *Sei s eine Variablenbelegung mit $g(s) = 1$ und X die Variablenmenge eines maximalen Terms c mit $c(s) = 1$. Dann gibt es wenigstens eine Variablenbelegung $s' = s(X')$ für ein $X' \subseteq X$, mit $f(s') = 1$, d.h. $s' = s(X')$ erfüllt eine ungerade Anzahl von Klauseln.*

BEWEIS: Sei $T = \{c_i \mid c_i(s) = 1\}$ die Menge von Klauseln, die von s erfüllt werden.

- Wenn $|T|$ ungerade ist, dann erfüllt $X' = X$ die Bedingung.
- Wenn $|T|$ gerade ist, so dann teilen wir T in die Mengen T' und T''. T' ist die Menge aller Klauseln, so daß mindestens eine Variable aus X in der Klausel vorkommt, T'' enthält die Klauseln, die keine Variablen mit X gemeinsam haben.
  - Falls $|T''|$ ungerade ist, dann wählen wir $X' = \emptyset$. Dadurch werden alle Klauseln in T'' erfüllt aber keine in T'.
  - Falls $|T''|$ und damit auch $|T'|$ gerade ist, so benutzen wir die folgende Konstruktion. Dazu sei
$$P(X') := \{c_i \mid c_i(s(X')) = 1\},$$
$$Q(X') := \{c_i \mid \mathrm{Var}\,(c_i) \cap X = X'\}$$
und $p(X')$ und $q(X')$ die Parität der zugehörigen Mengen. Aufgrund der Maximalität von c besteht $Q(X)$ nur aus dem Element c. Damit ist $q(X) = 1$. Wir sind nun interessiert an einem X' mit $p(X') = 1$. Zwischen den Mengen P und Q (und damit auch zwischen p und q) besteht der folgende, einfach zu beweisende Zusammenhang:
$$P(X') = \bigcup_{X'' \subseteq X'} Q(X'').$$
Dieser Zusammenhang läßt sich durch eine nicht singuläre obere Dreiecksmatrix ausdrücken. Ebenso lassen sich die Werte von p durch die Werte von q durch ein lineares invertierbares Gleichungssystem über GF(2) ausdrücken. Da $q(X) \neq 0$ ist, folgt also, daß es wenigstens ein X' mit $p(X') \neq 0$ gibt. ■

Lemma 12.8 *Es gibt eine Zuordnung $\varphi : G \to F$, so daß für alle $s \in F$ die Urbildmenge $\varphi^{-1}(s)$ höchstens m Elemente enthält.*

BEWEIS: Zu $s \in F$ sei t_s ein maximaler Term, der durch s erfüllt wird, d.h. es gibt keinen Term t mit $t(s) = 1$, dessen Variablenmenge die Variablenmenge von t_s enthält. Für die (höchstens) $m-1$ vielen Terme t, die nicht von s erfüllt werden, sei s_t die Belegung, in der alle Variablen, die in t vorkommen, auf 1 gesetzt sind, die aber sonst mit s übereinstimmt. Damit definieren wir die Urbildmenge $M_s = \varphi^{-1}(s)$ von s als

$$M_s = \{s\} \cup \{s_t \mid t(s) = 0\}.$$

Wir behaupten nun, daß $G = \cup_{s\in F} M_s$ gilt. Sei $\bar{s} \in G$ und s' die nach Lemma 12.7 konstruierte erfüllende Belegung von f. Nach dieser Konstruktion gilt $\bar{s} \in M_{s'}$. Daher ist φ die gewünschte Zuordnung.

Beweis: von Satz 12.7
In Lemma 12.8 haben wir ein Abbildung $\varphi : G \to F$ konstruiert, so daß für alle $s \in F$ die Urbildmenge $\varphi^{-1}(s)$ höchstens m Elemente enthält. Dann kann die Menge G höchstens m mal so groß sein wie F. ∎

Dieses Ergebnis ist sogar optimal. Betrachten wir für eine Zweierpotenz m das Polynom

$$g_T = \prod_{i\in T}(x_i \oplus 1) \prod_{i\notin T} x_i.$$

Wenn $|T| = \log m$ ist, so hat g_T genau m Terme und eine Einsstelle. Die entsprechende Funktion

$$f_T = \bigwedge_{i\in T}(x_i \vee 1) \bigwedge_{i\notin T} x_i.$$

in disjunktiver Normalform hat jedoch m Einsstellen.

Bis jetzt haben wir GF(2)-Polynome mit Termen betrachtet, deren Variablenmenge jeweils nichtleer war. Dies sind gerade diejenigen Polynomen, die nicht die Konstante 1 (= Term mit leerer Variablenmenge) enthalten. Für GF(2)-Polynome mit Konstante 1 gilt der folgende

Satz 12.9 *Sei p ein Polynom in den Variablen $\{x_1, \ldots x_n\}$ mit $p(0,\ldots,0) = 1$. Sei $U = \{0,1\}^n$ die Menge aller Einsetzungen für die Variablen und F die Menge aller Einsstellen von p. Dann gilt*

$$|U|/|F| \leq m.$$

Dieser Satz impliziert direkt einen Algorithmus für diese Polynome, der in Laufzeit

$$O(nm^2 \log(1/\delta)/\epsilon^2)$$

implementierbar ist.

12.3.2 Approximative Zählung von c-Stellen multilinearer Polynome über GF(q)

In diesem Abschnitt wollen wir die Anzahl von c-Stellen von multilinearen Polynomen über GF(q) bestimmen ([KL91]).

Definition 12.9 *Ein multivariates Polynom $f(x_1, \ldots, x_n)$ heißt multilinear, wenn $deg_{x_i} \leq 1$ für jedes i gilt.*

Definition 12.10 *Es bezeichne*

$$\#_c g = |\{(x_1, \ldots x_n) \in \mathrm{GF}(q)^n \mid g(x_1, \ldots x_n) = c\}|$$

die Anzahl der c-Stellen von $g \in \mathrm{GF}(q)[x_1, \ldots, x_n]$.

Lemma 12.9 *Sei $g \in \mathrm{GF}(q)[x_1, \ldots, x_n]$ ein multilineares nicht konstantes Polynom. Dann gilt für alle $c \in \mathrm{GF}(q)$:*

$$\#_c g \geq (q-1)^{n-1}.$$

Beweis: Wir beweisen das Lemma durch Induktion über n der Anzahl der Variablen.

$n = 1$ Sei $g(x) = ax + b$ mit $a \neq 0$. Dann hat die Gleichung $g(x) = s$ für jedes $s \in \mathrm{GF}(q)$ genau eine Lösung. Somit gilt

$$\#_c g = 1 = (q-1)^0.$$

$n \to n+1$ Wir stellen das Polynom g folgendermaßen dar:

$$g(x_1, \dots x_{n+1}) = x_{n+1}h(x_1, \dots x_n) + k(x_1, \dots x_n).$$

Dabei sind h und k multilineare Polynome in den Variablen $x_1, \dots, x_n$. Wir unterscheiden nun drei Fälle:

- $h(x_1, \dots, x_n) \equiv 0$
  In diesem Fall ist
  $$\#_c g = q\#_c k \geq q(q-1)^{n-1} > (q-1)^n.$$
- $h(x_1, \dots x_n) \equiv d \neq 0$
  Das Polynom g hat also die Gestalt $g(x_1, \dots x_{n+1}) = x_{n+1}d + k(x_1, \dots x_n)$. Für festes $(x_1, \dots x_n)$ haben wir ein lineares univariates Polynom, welches jeden Wert $c \in \mathrm{GF}(q)$ genau einmal annimmt unabhängig von dem Wert von $k(x_1, \dots, x_n)$. Für jede beliebige Belegung der Variablen $x_1, \dots, x_n$ hat $g(x_1, \dots, x_n, x_{n+1}) = c$ genau eine Lösung. Somit ist
  $$\#_c g = q^n > (q-1)^n.$$
- $h(x_1, \dots, x_n)$ ist nicht konstant. Nach Induktionsannahme gibt es mindestens $(q-1)^{n-1}$ Lösungen der Gleichung $h(x_1, \dots x_n) = d$. Für $d \neq 0$ ist gibt es für jede dieser Lösungen $(x_1, \dots, x_n)$ genau ein x_{n+1} mit $g(x_1, \dots, x_{n+1}) = c$. Damit gilt
  $$\#_c g \geq \sum_{d \neq 0} \#_d h \geq (q-1)(q-1)^{n-1} = (q-1)^n.$$

■

Bezeichne $\mathrm{Var}(t)$ die Variablenmenge eines Polynoms oder Terms t.

Satz 12.10 *Sei $f \in \mathrm{GF}(q)[x_1, \dots, x_n]$ ein multilineares Polynom und $c \in \mathrm{GF}(q)$. Bezeichne m die Anzahl der Terme von $f = \sum t_i$, $D(f) := \{s \in \mathrm{GF}(q)^n \mid \exists t_i \; t_i(s) \neq 0\}$ und $S_c(f) :=$*

$\{s \in \mathrm{GF}(q)^n \mid f(s) = c\}$. *Falls f keinen konstanten Term enthält oder der konstante Term gleich c ist, dann gilt*

$$\frac{|D(f)|}{|S_c(f)|} \leq (q-1)m.$$

Beweis: Wir partitionieren $D(f)$ in Mengen $D_{i,j}(f)$ und überdecken $S_c(f)$ durch eine gleiche Anzahl von Mengen $R_{i,j}(f)$. Durch eine Beziehung zwischen den Mengen $R_{i,j}(f)$ und$D_{i,j}(f)$ erhalten wir dann die gewünschte Aussage.

Zunächst überdecken wir $D(f)$ durch nicht notwendigerweise disjunkte Mengen $D_i(f)$:

$$D_i(f) := \{s \in \mathrm{GF}(q)^n \mid t_i(s) \neq 0,\ \neg\exists t_j > t_i \text{ mit } t_j(s) \neq 0\}.$$

Dabei bedeutet $t_i < t_j$, daß $\mathrm{Var}\,(t_i) \subset \mathrm{Var}\,(t_j)$ gilt.

$D_i(f)$ besteht also aus den Zuweisungen s, für die der Term t_i maximal unter allen Termen mit $t(s) \neq 0$ ist.

$D_i(f)$ partitionieren wir in $q^{n-\deg t_i}$ Mengen $D_{i,j}(f)$. $D_{i,j}(f)$ besteht aus all den Zuweisungen aus $D_i(f)$, die auf den Variablen, die nicht in t_i auftauchen, gleich sind. Die Größe von $D_{i,j}(f)$ ist durch die Anzahl von Zuweisungen s zu den Variablen in t_i mit $t_i(s) \neq 0$ beschränkt. Da jede der $\deg t_i$ vielen Variablen einen Wert ungleich 0 erhalten muß, sind dies $(q-1)^{\deg t_i}$ viele.

Zu den Mengen $D_{i,j}(f)$ definieren wir die Mengen $R_{i,j}(f)$ in der folgenden Weise. Die Zuweisungen zu den Variablen, die nicht in t_i auftauchen, sei so wie in $D_{i,j}(f)$, d.h. wir betrachten Polynome p , die aus einer partiellen Zuweisung der Variablen hervorgehen. Diese Polynome sind Polynome in den Variablen in $\mathrm{Var}\,(t_i)$. Wir erweitern nun die partielle Zuweisung, so daß p und damit f zu c evaluiert.

Aufgrund der Definition der Mengen $D_i(f)$, werden die Polynome, die durch partielle Zuweisung der Variablen aus f hervorgehen, nicht konstant. Daher können wir Lemma 12.9 auf die Polynome p anwenden und erhalten daher für alle i, j eine untere Schranke für die

Kardinalität der Mengen $R_{i,j}(f)$:

$$|R_{i,j}(f)| \geq (q-1)^{\deg t_i - 1} \geq \frac{|D_{i,j}(f)|}{q-1}.$$

Da die Mengen $R_{i,j}(f)$ die Menge $S_c(f)$ überdecken haben, wir die Ungleichung

$$|S_c(f)| \leq \sum_{i,j} |R_{i,j}(f)|.$$

Aufgrund der Konstruktion sind die Mengen $R_{i,j}(f)$ und $R_{i,k}(f)$ für $j \neq k$ disjunkt, da sie den Variablen außerhalb von $\mathrm{Var}\,(t_i)$ verschiedene Werte zuweisen. Daher kann jedes Element $s \in S_c(f)$ in höchstens m verschiedenen Mengen $R_{i,j}(f)$ auftauchen. Damit haben wir

$$m \cdot |S_c(f)| \geq \sum_{i,j} |R_{i,j}(f)|.$$

Zusammenfassend ergibt sich

$$\frac{|D(f)|}{|S_c(f)|} \leq \frac{\sum_{i,j} |D_{i,j}(f)|}{1/m \sum_{i,j} |R_{i,j}(f)|} \leq \frac{\sum_{i,j} (q-1)|R_{i,j}(f)|}{1/m \sum_{i,j} |R_{i,j}(f)|} = m(q-1).$$

■

Die Schranke des Satzes 12.10 ist scharf. Dies zeigt das Beispiel

$$f(x_1, \ldots, x_n) = \prod_{i=1}^{n} x_i.$$

Es gibt $(q-1)^n$ Belegungen, so daß (der einzige) Term von f ungleich 0 wird, aber genau $(q-1)^{n-1}$ viele Belegungen, so daß f den Wert c annimmt. Das Verhältnis ist genau $q-1$.

Wir haben nun wieder eine geeignete Überdeckung konstruiert, und können wieder dasselbe Schema für einen (ϵ, δ)-Approximationsalgorithmus anwenden.

Dieser Approximationsalgorithmus ist für den Fall der beliebigen Polynome über $\mathrm{GF}(q)$ durch Grigoriew und Karpinski [GK91] erweitert worden.

Kapitel 13

Neue Approximationsschemata

Wir haben in dem vorliegenden Skript verschiedene Probleme bezüglich ihrer Approximierbarkeit untersucht. Es wurden Approximationsalgorithmen für Optimierungsprobleme wie das *Problem des Handlungsreisenden*, *Vertex-Cover*, etc. entwickelt, die ein konstantes Approximationsverhältnis besitzen. Für einige Zählprobleme haben wir (ϵ, δ)-Approximationsschemata konstruieren können. Für andere Probleme haben wir gezeigt, daß sie gar nicht approximiert werden können (modulo hinreichend starker Bedingungen).

In Kapitel 1.3 haben wir für Optimierungsprobleme sogenannte *polynomzeitbeschränktes Approximationsschemata* definiert:

Ein polynomzeitbeschränktes Approximationsschema (kurz PTAS) $\mathcal{P}$ ist eine Familie $(\mathcal{P}_\epsilon, \epsilon > 0)$ von polynomiellen Algorithmen $\mathcal{P}_\epsilon$, mit Approximationsgüte $1 + \epsilon$.

Die *Approximationsgüte* war dabei wie folgt definiert:

Gegeben sei ein Optimierungsproblem P. Ein Approximationsalgorithmus $\mathcal{A}$ besitzt eine Approximationsgüte (ein Approximation Ratio A.R.) von α, wenn für jede Instanz $x \in P$

$$\max\left\{\frac{c(\mathcal{A}(x))}{OPT(x)}, \frac{OPT(x)}{c(\mathcal{A}(x))}\right\} \leq \alpha$$

gilt. Nun stellt sich die Frage, ob $PTAS$s für NP-harte Optimierungsprobleme überhaupt existieren können: Die Antwort ist **Ja**.

13.1 *PTAS* für das Euklidische Traveling Salesman und Steiner Tree Problem

Im allgemeinen ist das TSP-Entscheidungsproblem NP-vollständig. Selbst die Approximation des allgemeinen Problems ist NP-hart, wie wir schon in Kapitel 1.3 gesehen haben. Über andere verwandte Probleme wie z. B. Steiner Tree Probleme siehe auch [KZ97a], [KR98].

Einschränkungen auf Eingabeinstanzen verändern wie im vorhergehenden Abschnitt die Lage jedoch erheblich: Das $TSP-Metric$ ist zwar noch NP-hart, es existieren allerdings Approximationalgorithmen, die Lösungen konstruieren, deren Kosten höchstens das 2- bzw. 1.5-fache der Optimallösung betragen (siehe auch Kapitel 1.3). Papadimitrou und Yannakakis zeigten jedoch, daß das Problem MAX-SNP-hart ist [PY93].

Das euklidische TSP ($ETSP$) ist immer noch NP-hart. Arora konstruierte jedoch 1996 erstaunlicherweise einen $PTAS$ für das $ETSP$ und andere euklidische Optimierungsprobleme.

Satz 13.1 ([A96]) *Es gibt einen $PTAS$ für $ETSP$.*

BEWEISMETHODE:

Durch rekursive Teilung der euklidischen Ebene (des $\mathbb{R}^2$) und Definition einer konstanten Anzahl von Portalen an denen sich Übergänge der Tour durch die Teilung existieren, kann man eine polynomielle Anzahl von Touren isolieren, von denen eine mit hoher Wahrscheinlichkeit ϵ-nah an der optimalen Tour liegt. Man braucht schließlich nur noch alle diese Touren aufzuzählen und die beste auszuwählen. ■

Kapitel 14

Die MCMC - Methode

Wir beschreiben in diesem Kapitel die Monte Carlo - Markov Chain - Methode (kurz: MCMC-Methode), mittels derer man auch zahlreiche effiziente Zähl- und Sampling-Verfahren erhalten kann. Betrachten wir einführend folgendes Problem: Gegeben sei ein ungerichteter Graph $G = (V, E)$, wir wollen ein Random Matching M erzeugen, d.h. wir fragen nach einem probabilistischen Algorithmus, der uns Matchings gleichverteilt (*uniformly at random*) erzeugt. Es wird sich zeigen, dass wir einen solchen Algorithmus erhalten, indem wir eine geeignete Markov-Kette auf der Menge aller Matchings konstruieren und nachweisen, dass diese hinreichend schnell gegen die Gleichverteilung auf der Menge aller Matchings von G konvergiert. Wir beginnen mit einer Einführung über Markov-Ketten und behandeln dann ihre Konvergenzeigenschaften, insbesondere das sog. *Rapid Mixing*. Eine erfolgreiche Methode zum Erreichen von Rapid Mixing ist das sogenannte *Coupling*, bei dem man übergeht zu Markov-Ketten auf einem erweiterten State Space, deren Projektionen auf den ursprünglichen Zustandsraum die ursprüngliche Markov-Kette wiedergeben. Wir werden uns dabei auf das sog. *Markov'sche Coupling* beschränken. Bessere Ergebnisse können mit dem sog. *Path Coupling* erzielt werden, das z.B. von Bordewich, Dyer und Karpinski zum Samplen von unabhängigen Mengen (*Independent Sets*) und Färbungen in Hypergraphen verwendet wird ([BDK05],[BDK06]).

14.1 Markov - Ketten

Ein stochastischer Prozess ist eine Familie $(X_t)_{t\in I}$ von Zufallsvariablen. Falls $I = \mathbb{N}$ oder $I = \mathbb{Z}$ gilt, so spricht man von einem Prozess *mit diskretem Zeitparameter*. Auf solche werden wir uns in den folgenden Betrachtungen beschränken.

Definition 14.1 *Eine Folge $(X_t)_{t\in\mathbb{N}}$ von Zufallsvariablen X_t mit Werten in Ω heisst Markov-Kette mit Zustandsraum (State Space) Ω, falls für alle $t \in \mathbb{N}$ und $x_0, \ldots, x_t \in \Omega$ gilt:*

$$Pr[X_{t+1} = y | X_t = x_t, \ldots, X_0 = x_0] \ = \ Pr[X_{t+1} = y | X_t = x_t]$$

Falls dabei die rechte Seite der Gleichung noch unabhängig von t ist, so spricht man von einer zeithomogenen Markov-Kette (time-homogenious Markov Chain), d.h. es gibt dann eine Matrix $P = (P(x, y))_{x,y\in\Omega}$ so, dass für alle $t \in \mathbb{N}$ und alle $x, y \in \Omega$ gilt:

$$P(x, y) \ = \ Pr[X_{t+1} = y | X_t = x_t]$$

P heisst Transitionsmatrix der Markov-Kette.

Eine zeithomogene Markov-Kette kann man also spezifizieren durch Angabe der Anfangsverteilung X_0 (engl. *initial distribution*) und der Transitionsmatrix P. Dabei ist wohlgemerkt P eine sogenannte *stochastische Matrix*, d.h. für alle $x \in \Omega$ gilt $\sum_{y\in\Omega} P(x, y) = 1$. Die Transitionsmatrix einer zeithomogenen Markov-Kette beschreibt die Einzelschritt-Übergangswahrscheinlichkeiten. Dann gilt für das Matrix-Produkt $P^t = P \cdot \ldots \cdot P$, dass

$$P^t(x, y) \ := \ \begin{cases} I(x, y), & \text{falls } t = 0 \\ \sum_{y'\in\Omega} P^{t-1}(x, y') \cdot P(y', y), & \text{falls } t > 0 \end{cases}$$

die t-Schritt-Übergangswahrscheinlichkeiten beschreibt.

Man interessiert sich bei Markov-Ketten insbesondere dafür, ob diese gegen einen stabilen Zustand, d.h. eine Grenzverteilung konvergieren.

Definition 14.2 *Es sei* $(X_t)_{t\in\mathbb{N}}$ *eine zeithomogene Markov-Kette mit State Space* Ω *und Transitionsmatrix* P*. Dann heisst eine Verteilung* $\pi\colon\Omega\to[0,1]$ *stationäre Verteilung (stationary distribution), falls für alle* $y\in\Omega$ *gilt:*

$$\pi(y)=\sum_{x\in\Omega}\pi(x)\cdot P(x,y)$$

Fassen wir π als einen $|\Omega|$-dimensionalen Zeilenvektor auf, so können wir obige Bedingung schreiben als $\pi=\pi\cdot P$.

Es stellt sich nun die Frage nach hinreichenden Kriterien für die Existenz stationärer Verteilungen. Wir benötigen den Begriff der ergodischen Markov-Kette.

Definition 14.3 *Es sei* (X_t) *eine zeithomogene Markov-Kette mit Transitionsmatrix* P *und endlichem Zustandsraum* Ω*.*

(a) Die Markov-Kette heisst irreduzibel, falls für alle $x,y\in\Omega$ *ein* $t\in\mathbb{N}$ *existiert mit* $P^t(x,y)>0$*.*

(b) Die Markov-Kette heisst aperiodisch, falls $ggT\{t|\,P^t(x,x)>0\}=1$ *für alle* $x\in\Omega$ *gilt.*

(X_t) *heisst ergodisch, falls sie irreduzibel und aperiodisch ist.*

Für ergodische Markov-Ketten kann es höchstens eine stationäre Verteilung geben. Denn wenn π,π' zwei stationäre Verteilungen einer ergodischen Markov-Kette mit Transitionsmatrix P sind, so folgt $\pi\cdot P=\pi'\cdot P$ (wobei wir wiederum π,π' als Zeilenvektoren auffassen), und aus der Nicht-Singularität von P folgt $\pi=\pi'$.

Lemma 14.1 *Es sei* P *die Transitionsmatrix einer Markov-Kette mit State Space* Ω*, und* $\pi\colon\Omega\to[0,1]$ *sei eine Wahrscheinlichkeitsverteilung. Falls*

$$\forall x,y\in\Omega\;\;\pi(x)\cdot P(x,y)=\pi(y)\cdot P(y,x)$$

gilt, so ist π *eine stationäre Verteilung der Markov-Kette.*

BEWEIS: Es sei $y \in \Omega$, dann gilt

$$\begin{aligned}\sum_{x\in\Omega} \pi(x) \cdot P(x,y) &= \sum_{x\in\Omega} \pi(y) \cdot P(y,x) \\ &= \pi(y) \cdot \sum_{x\in\Omega} P(y,x) = \pi(y)\end{aligned}$$

Somit ist π stationär. ■

Dass ergodische Markov-Ketten tatsächlich stationäre verteilungen besitzen, besagt folgendes Theorem.

Satz 14.1 *Jede ergodische Markov-Kette besitzt eine stationäre Verteilung π. Ferner gilt für alle $x, y \in \Omega$ $\lim_{t\to\infty} P^t(x,y) = \pi(y)$.*

Wir führen obiges Beispiel fort. Es sei $G = (V, E)$ ein ungerichteter Graph, und es sei

$$\mathcal{M}(G) = \{M \subseteq E \mid M \text{ ist Matching in } G\}$$

Wir konstruieren eine Markov-Kette mit State Space $\mathcal{M}(G)$ wie folgt: Es ist die Anfangsverteilung X_0 gegeben durch

$$X_0(M_0) = 1,\ \ X_0(M') = 0 \text{ für alle } M' \in \mathcal{M}(G) \setminus \{M_0\}$$

für ein vorher gewähltes Matching M_0 in G. Wir beschreiben die Transitionsmatrix P implizit durch einen probabilistischen Algorithmus, der bei Eingabe von $M \in \mathcal{M}(G)$ für $M' \in \mathcal{M}(G)$ mit Wahrscheinlichkeit $P(M, M')$ das Matching M' ausgibt:

1. Mit Wahrscheinlichkeit $\frac{1}{2}$ setze $M' = M$ und gib M' aus.
2. Andernfalls wähle $e \in_R E$ und setze $M' := M\Delta\{e\}$.
3. Falls $M' \notin \mathcal{M}(G)$, so setze $M' := M$.
   Gib M' aus.

Auf diese Weise wird eine Markov-Kette definiert, und man kann zeigen, dass diese auch ergodisch ist: Sie ist irreduzibel, denn mit einer von 0 verschiedenen Wahrscheinlichkeit kann ein beliebiges Matching $M \in \mathcal{M}(G)$ durch iteriertes Anwenden von Schritt 2 auf Matching-Kanten in das leere Matching transformiert werden, und aus diesem erhält man ebenfalls jedes beliebige Matching $M' \in \mathcal{M}(G)$ mit einer von 0 verschiedenen Wahrscheinlichkeit durch iteriertes Anwenden von Schritt 2 auf noch nicht im jeweils aktuellen Matching befindliche Kanten von M'.

Die Markov-Kette ist aperiodisch, dies folgt direkt aus Schritt 1 des Algorithmus. Also ist sie ergodisch.

Ferner gilt: Die stationäre Verteilung dieser Markov-Kette ist die uniforme Verteilung auf $\mathcal{M}(G)$. Dies zeigt man wie folgt: Die stationäre Verteilung ist eindeutig, und die uniforme Verteilung (Gleichverteilung) auf $\mathcal{M}(G)$ ist stationär. Dies folgt aus Lemma 5.1, denn für die Gleichverteilung π auf $\mathcal{M}(G)$ gilt $\pi(M) = 1/|\mathcal{M}(G)|$ für alle $M \in \mathcal{M}(G)$, und es gilt $P(M, M') = P(M', M)$ für alle $M, M' \in \mathcal{M}(G)$.

14.2 Mixing-Zeiten

Um Markov-Ketten algorithmisch einzusetzen, z.B. um Objekte zu samplen, reicht die bloße Existenz einer stationären Verteilung nicht aus. Es muss zusätzlich gesichert sein, dass die Markov-Kette auch hinreichend schnell gegen diese konvergiert. Diese Eigenschaft von Markov-Ketten wird als *Rapid Mixing* bezeichnet. Im folgenden betrachten wir Techniken, mittels derer man die Mixing-Zeit von Markov-Ketten beschränken kann. Zunächst stellen wir in diesem Abschnitt die notwendigen Begriffe bereit.

Es sei (X_t) eine ergodische Markov-Kette mit abzählbarem State Space Ω, Transitionsmatrix P und Anfangsverteilung $X_0 = x \in \Omega$ (d.h. $X_0(x) = 1$ und $X_0(y) = 0$ für alle $y \in \Omega \setminus \{x\}$. Dann ist $P^t(x, \cdot)$ die Verteilung von X_t. Es sei π die stationäre Verteilung, insbesondere gilt also für alle $y \in \Omega$ $\lim_{t\to\infty} P^t(x, y) = \pi(y)$.

Definition 14.4 *Es seien π_1 und π_2 zwei Wahrscheinlichkeitsverteilungen auf Ω, und Ω sei* *abzählbar. Dann ist*

$$\|\pi_1 - \pi_2\|_{TV} \;:=\; \frac{1}{2}\sum_{x\in\Omega} |\pi_1(x) - \pi_2(x)| \;=\; \max_{A\subseteq\Omega} |\pi_1(A) - \pi_2(A)|$$

die Total Variation Distance von π_1, π_2*.*

Die Total Variation Distance von P^t zur stationären Verteilung ist monoton in t:

Lemma 14.2 *Die Total Variation Distance der Zeilen der t-Schritt-Transitionsmatrix* P^t *einer ergodischen Markov-Kette zur stationären Verteilung* $\|P^t(x,\cdot) - \pi\|_{TV}$ *ist eine monoton fallende Funktion von* t*.*

Die Mixing-Zeit (engl. *Mixing Time*) ist nun die Konvergenzrate gemessen in der Total Variation Distance.

Definition 14.5 *Die Mixing-Zeit* $\tau_x(\epsilon)$ *der ergodischen Markov-Kette* (X_t) *mit Transitionsmatrix* P *und stationärer Verteilung* π *ist definiert durch*

$$\tau_x(\epsilon) \;:=\; \min\{t \mid \|P^t(x,\cdot) - \pi\|_{TV} \;\le \epsilon\}$$

Ferner ist $\tau(\epsilon) := \max_{x\in\Omega} \tau_x(\epsilon)$*.*

14.3 Coupling und Rapid Mixing

Eine sehr erfolgreiche Methode zum Beschränken der Mixing-Zeit einer Markov-Kette ist das sogenannte *Coupling*. Die Grundidee besteht darin, von einer Markov-Kette auf einem State Space Ω überzugehen zu einer Markov-Kette auf $\Omega \times \Omega$. Diese ist dann häufig einfacher zu analysieren, und man kann Kriterien angeben, unter denen sich die Mixing-Zeit der ursprünglichen Markov-Kette beschränken lässt. Die einfachste Variante ist das sog. Markov'sche Coupling. Zu den fortgeschrittenen Methoden zählt das sog. *Path Coupling*. Dieses wurde von Bordewich, Dyer und Karpinski ([BDK05], [BDK06]) zum Samplen von unabhängigen Mengen und Färbungen in (uniformen) Hypergraphen verwendet.

Wir beginnen mit dem sogenannten Markov'schen Coupling.

Definition 14.6 *Es sei* (Z_t) *eine zeithomogene Markov-Kette mit State Space* Ω *und Transitionsmatrix* P*. Ein Markov'sches Coupling für* (Z_t) *ist eine Markov-Kette* (X_t, Y_t) *auf dem State Space* $\Omega \times \Omega$ *mit Transitionsmatrix* $\hat{P}$ *so, dass für alle* $x, y \in \Omega$ *gilt:*

$$\begin{aligned}\sum_{y' \in \Omega} \hat{P}((x,y),(x',y')) &= P(x,x') \\ \sum_{x' \in \Omega} \hat{P}((x,y),(x',y')) &= P(y,y')\end{aligned}$$

Ein *Markov'sches Coupling* ist also eine Markov-Kette auf $\Omega \times \Omega$ so, dass die Projektionen Markov-Ketten mit Transitionsmatrix P sind. Wenn nun mit hinreichend hoher Wahrscheinlichkeit nach Zeit t (d.h. t Schritten) die beiden Projektionen gleich sind, so liefert t eine Schranke für die Mixing-Zeit:

Lemma 14.3 *Es sei* (X_t, Y_t) *ein Markov'sches Coupling zur Markov-Kette* (Z_t)*.*
Es sei $t\colon [0,1] \to I\!N$ *so, dass für alle* $x, y \in \Omega$ *und* $\epsilon > 0$ *gilt:*

$$Pr[X_{t(\epsilon)} \neq Y_{t(\epsilon)} \mid X_0 = x, Y_0 = y] \;\leq\; \epsilon$$

Dann gilt für die Mixing-Zeit $\tau(\epsilon)$ *von* (Z_t)*, dass* $\tau(\epsilon) \leq t(\epsilon)$

Beweis: Es sei P die Transitionsmatrix von (Z_t). Für eine Teilmenge $A \subseteq \Omega$ setzen wir $P^t(x, A) = \sum_{a \in A} P^t(x, a) = Pr[X_t \in A]$. Es sei $X_0 = x \in \Omega$ (s.o.), und Y_0 sei gemäß der stationären Verteilung π von (Z_t) verteilt. Dann gilt für jedes $0 < \epsilon < 1$ und $t := t(\epsilon)$:

$$\begin{aligned}P^t(x, A) &= Pr[X_t \in A] \\ &\geq Pr[X_t = Y_t \;\wedge\; Y_t \in A]\end{aligned}$$

$$
\begin{aligned}
&= 1 - Pr[X_t \neq Y_t \,\vee\, Y_t \notin A] \\
&\geq 1 - (Pr[X_t \neq Y_t] \,+\, Pr[Y_t \notin A]) \\
&\geq Pr[Y_t \in A] - \epsilon \\
&= \pi(A) - \epsilon
\end{aligned}
$$

Weil hierbei $A \subseteq \Omega$ beliebig gewählt war, erhalten wir für die Total Variation Distance von P^t bei Startpunkten x bzw. y:

$$\|P^t(x,\cdot) - P^t(y,\cdot)\|_{TV} \;=\; \max_{B \subseteq \Omega} |P^t(x,B) - P^t(y,B)| \;\leq\; \epsilon$$

sowie $\|P^t(x,\cdot) - \pi\|_{TV} \leq \epsilon$. ■

14.3.1 Färben von Bounded Degree Graphen

Wir betrachten als Beispiel das Färben von Graphen mit maximalem Knotengrad Δ mit einer festen Anzahl q von Farben.

Definition 14.7 *Es sei $G = (V,E)$ ein Graph. Eine q-Färbung von G ist eine Abbildung $\lambda\colon V \to Q$ in eine Menge Q der Kardinalität $|Q| = q$, so dass gilt:*

$$\forall e = \{u,v\} \in E \;\; \lambda(u) \neq \lambda(v)$$

Im allgemeinen ist bereits die Frage nach der Existenz von q-Färbungen ein NP-vollständiges Problem:

Satz 14.2 *Das Entscheidungsproblem GRAPH COLORING (gegeben ein Graph $G = (V,E)$ und eine Zahl $q \in \mathbb{N}$, besitzt G eine q-Färbung ?) ist NP-vollständig.*

Beweis: Wir beweisen die NP-Vollständigkeit des eingeschränkten Problems 3-COLORING (Gegeben ein Graph $G = (V,E)$, besitzt G eine 3-Färbung ?). Dazu zeigen wir NAESAT$\leq_p$ 3-COLORING, wobei NAESAT (Not-All-Equal SAT) wie folgt definiert ist: Gegeben eine

Formel in 3-KNF $\varphi = C_1 \wedge \ldots \wedge C_m$, gibt es eine Belegung der Variablen so, dass für jede Klausel C_i gilt, dass nicht alle Literale der Klausel den gleichen Wahrheitswert haben ? Die Reduktion ist wie folgt: Zu gegebener Instanz $\varphi = C_1 \wedge \ldots \wedge C_m$ von NAESAT mit Variablen $x_1, \ldots, x_n$ sei $G_\varphi = (V, E)$ gegeben durch

$$\begin{aligned} V &= \{a\} \cup \{v_{i,0}, v_{i,1} | 1 \leq i \leq n\} \cup \{u_{j,l} | l \text{ Literal in } C_j,\ 1 \leq j \leq m\} \\ E &= \{\{a, v_{i,b}\} \mid 1 \leq i \leq n, b \in \{0,1\}\} \\ &\quad \cup \{\{u_{j,l}, u_{j,l'}\} \mid 1 \leq j \leq m,\ l, l' \in C_j, l \neq l'\} \\ &\quad \cup \{\{u_{j,x_i^b}, v_{i,b}\} \mid 1 \leq i \leq n,\ 1 \leq j \leq m,\ x_i^b \in C_j\} \end{aligned}$$

Es gilt: φ ist eine Ja-Instanz von NAESAT genau dann, wenn G 3-färbbar ist. ■

Wir bezeichnen mit Δ_G den maximalen Knotengrad von G. Brook's Theorem besagt, dass für den Fall $q \geq \Delta_G \geq 3$ unter der Voraussetzung, dass G nicht den vollständigen Graphen K_{Δ_G+1} als Zusammenhangskomponente besitzt, eine q-Färbung von G existiert, und diese kann dann auch in polynomieller Zeit konstruiert werden. Wir werden hier den Fall $q \geq 2\Delta_G + 1$ betrachten.

Es sei nun $G = (V, E)$ ein Graph mit $\Delta_G = \Delta$. Wir wählen als State Space die Menge $\Omega = \{\lambda \colon V \to \{1, \ldots, q\} | \forall e = \{u, v\} \in E\, \lambda(u) \neq \lambda(v)\}$ und X_0 gleich eine beliebige q-Färbung von G, die wir z.B. mit dem oben erwähnten Algorithmus konstruieren. Die Transitionsmatrix P beschreiben wir wieder implizit durch Angabe eines entsprechenden probabilistischen Algorithmus, der bei gegebener q-Färbung λ randomisiert eine neue q-Färbung λ' generiert:

1. Wähle $v \in_R V$.
2. Wähle $c \in_R \{1, \ldots q\} \setminus \lambda(N_G(v))$. Hierbei ist $N_G(v) = \{u \in V \mid \{u, v\} \in E\}$ die Nachbarschaft von v in G.
3. Setze $\lambda'(v) = c$ und $\lambda'(u) = \lambda(u)$ für alle $u \in V \setminus \{v\}$.

Offenbar ist die so definierte Markov-Kette aperiodisch, denn aus $\lambda(v) \in \{1, \ldots q\} \setminus \lambda(N_G(v))$ für alle $v \in V$ folgt direkt $P(\lambda, \lambda) > 0$. Die Markov-Kette ist auch irreduzibel, denn nach

Voraussetzung gilt insbesondere $q \geq \Delta + 2$, und wir können einen Graphen $G_r = (\Omega, E_r)$ definieren, der genau diejenigen Kanten $\{\lambda, \lambda'\}$ enthält, für die es genau einen Knoten $v \in V$ gibt mit $\lambda(v) \neq \lambda'(v)$. Zwei Färbungen vonb G haben Abstand 1 in diesem Graphen genau dann, wenn sie durch einmaliges Anwenden des obigen Algorithmus ineinander überführbar sind. Zu zeigen ist, dass G_r zusammenhängend. Für $\lambda, \lambda' \in \Omega$ sei $d_H(\lambda, \lambda') = |\{v \in V | \lambda(v) \neq \lambda'(v)\}|$ die Hamming-Distanz. Färbungen mit Hamming-Distanz 1 liegen also in derselben Zusammenhangskomponente von G_r. Gelte nun, daß alle Färbungen mit Hamming-Distanz $\leq j$ in derselben Zusammenhangskomponente von G_r liegen. Gelte $d_H(\lambda_1, \lambda_2) = j + 1$. Dann gibt es einen Nachbarn von λ_2 in G_r, der von λ_1 aus auf einem Weg in G_r erreichbar ist. Dies zeigt insgesamt die Irreduziblität der Markov-Kette.

Also ist die Markov-Kette ergodisch und besitzt eine eindeutige stationäre Verteilung. Wiederum ist diese die Gleichverteilung auf der Menge Ω.

Satz 14.3 *Falls $q \geq 2\Delta + 1$ gilt, so gilt für die Mixing-Zeit $\tau(\epsilon)$ der obigen Markov-Kette*

$$\tau(\epsilon) \leq \frac{q - \Delta}{q - 2\Delta} \cdot n \cdot \ln\left(\frac{n}{\epsilon}\right)$$

Wir wollen zum Beweis dieses Satzes das Lemma 5.3 anwenden. Dazu müssen wir ein Markov'sches Coupling zur obigen Markov-Kette definieren. Wir benötigen den Begriff der *joint distribution* zweier Zufallsvariablen.

Definition 14.8 *Es sei U eine Menge, und X, Y seien zwei Zufallsvariablen mit Werten in U. Dann heisst eine Wahrscheinlichkeitsverteilung p auf $U \times U$ joint distribution von X, Y, falls gilt:*

$$\forall u, v \in U \;\; p(u, v) = Pr[X = u \wedge Y = v]$$

Lemma 14.4 *Es sei U eine endliche Menge, $A, B \subseteq U$ und X, Y seien Zufallsvariablen mit Werten in U so, dass X und Y jeden Wert in U mindestens einmal annehmen. Dann gibt es eine joint distribution p von X, Y mit folgenden Eigenschaften:*

(a) X ist gleichverteilt auf A, und $Pr[X \notin A] = 0$.

(b) Y ist gleichverteilt auf B, und $Pr[Y \notin B] = 0$.

(c) $Pr[X = Y] = \frac{|A \cap B|}{\max\{|A|,|B|\}}$

Beweis: (von Satz 5.3). Wir konstruieren ein Coupling wie folgt (dabei ist $Q = \{1, \ldots, q\}$ die Menge der Farben). Gegeben $(\lambda_1, \lambda_2) \in \Omega \times \Omega$, konstruiert folgender probabilistischer Algorithmus ein neues Paar von Farben (λ_1', λ_2'):

1. Wähle $v \in_R V$.
2. Wähle
$$(c_x, c_y) \in (Q \setminus \lambda_1(N_G(v))) \times (Q \setminus \lambda_2(N_G(v)))$$
gemäß einer joint distribution wie in Lemma 14.4 mit $A = Q \setminus \lambda_1(N_G(v))$ und $B = Q \setminus \lambda_2(N_G(v))$.
3. Setze $\lambda_1'(v) = c_x, \lambda_1'(u) = \lambda_1(u)$ für $u \in V \setminus \{v\}$
und $\lambda_2'(v) = c_y, \lambda_2'(u) = \lambda_2(u)$ für $u \in V \setminus \{v\}$.

Es gilt offenbar

$$Pr[c_x = c_y] = \frac{|\,(Q \setminus \lambda_1(N_G(v))) \cap (Q \setminus \lambda_2(N_G(v)))\,|}{\max\{\,|Q \setminus \lambda_1(N_G(v))|,\ |Q \setminus \lambda_2(N_G(v))|\,\}}$$

Die Färbungen des Coupling nach t Schritten seien nun mit λ_1^t, λ_2^t bezeichnet. Es sei $A_t \subseteq V$ die Menge der Knoten v von G, für die $\lambda_1^t(v) = \lambda_2^t(v)$ gilt, und es sei $D_t := V \setminus A_t$ das sogenannte *Disagreement Set*. Offenbar gilt dann

$$|D_{t+1}| \in \{\,|D_t| - 1,\ |D_t|,\ |D_t| + 1\,\}$$

Zu gegebenem Knoten v bezeichne $d'(v)$ die Anzahl der Kanten inzident zu v, die einen Endpunkt in A_t und einen in D_t haben. Die drei Möglichkeiten für den Wert von $|D_{t+1}|$ werden nun einzeln analysiert. Exemplarisch betrachten wir den Fall $|D_{t+1}| = |D_t| + 1$. Damit dies eintreten kann, muss der Knoten v, der in Schritt 1 gewählt wird, in A_t liegen, und es muss $c_x \neq c_y$ gelten. Insbesondere folgt daraus, dass

$$\begin{array}{ll} |Q \setminus \lambda_1(N_G(v))| - |\,(Q \setminus \lambda_1(N_G(v))) \cap (Q \setminus \lambda_2(N_G(v)))\,| & \leq d'(v) \\ |Q \setminus \lambda_2(N_G(v))| - |\,(Q \setminus \lambda_1(N_G(v))) \cap (Q \setminus \lambda_2(N_G(v)))\,| & \leq d'(v) \\ |Q \setminus \lambda_1(N_G(v))|,\ |Q \setminus \lambda_2(N_G(v))| & \geq q - \Delta \end{array}$$

Dies liefert

$$\begin{aligned} Pr[c_x = c_y \mid v] &\geq \frac{\max\{\,|Q \setminus \lambda_1(N_G(v))|,\ |Q \setminus \lambda_2(N_G(v))|\,\} - d'(v)}{\max\{\,|Q \setminus \lambda_1(N_G(v))|,\ |Q \setminus \lambda_2(N_G(v))|\,\}} \\ &\geq 1 - \frac{d'(v)}{q - \Delta} \end{aligned}$$

und somit insgesamt

$$\begin{aligned} Pr[|D_{t+1}| = |D_t| + 1] &= \frac{1}{n} \sum_{v \in A_t} Pr[c_x \neq c_y \mid v] \\ &\leq \frac{1}{n} \sum_{v \in A_t} \frac{d'(v)}{q - \Delta} = \frac{m'}{(q - \Delta)n}, \end{aligned}$$

wobei m' die Zahl der Kanten zwischen A_t und D_t bezeichne.

Analog kann man die anderen beiden Fälle analysieren und erhält schliesslich:

$$E[\,|D_{t+1}| \mid |D_t|\,] \leq \left(1 - \frac{q - 2\Delta}{(q - \Delta)n}\right) \cdot |D_t|.$$

Hieraus folgt direkt $Pr[\,|D_t \neq 0\,] \leq n(1-a)^t \leq n \cdot e^{-at}$ für $a = \frac{q-2\Delta}{(q-\Delta)n}$, somit $Pr[D_t \neq \emptyset] \leq \epsilon$ für $t \geq a^{-1} \cdot \ln(n\epsilon^{-1})$. ■

Anhang A

Appendix

A.1 Berechnungsprobleme

1. $3SAT$ ($kSAT$)

   $I.: KNF\ f$ mit der Länge der Klauseln gleich 3 (k).

   $O.: \exists x \in \{0,1\}^n[f(x) = 1]$

2. $3DNF$ ($kDNF$)

   $I.: DNF\ f$ mit der Länge der Klauseln gleich 3 (k).

   $O.: \exists x \in \{0,1\}^n[f(x) = 1]$

3. MAX-SAT

   $I.: KNF\ f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$

   $O.: s \in \{0,1\}^n, s.d. |\{i|c_i(s) = 1\}| = \max\{|\{i|c_i(x) = 1\}| x \in \{0,1\}^n\}$

4. l-Occ-$kSAT$

   $I.: KNF\ f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ mit der Länge der Klauseln gleich k und jede Variable taucht höchstens l mal in f auf.

   $O.: s \in \{0,1\}^n, s.d. |\{i|c_i(s) = 1\}| = \max\{|\{i|c_i(x) = 1\}| x \in \{0,1\}^n\}$

5. MAX-$3LIN(MAX$-$kLIN)$

   $I.$: System von linearen Gleichungen mit n Variablem mod 2 und genau 3 (k) Variablem pro Gleichung.

   $O.$: $s \in \{0,1\}^n$, so daß die Anzahl der erfüllten Gleichungen maximal ist.

6. TSP (mit Dreiecksungleichung)

   $I.$: $U = \{U_1, \ldots, U_n\}, d : U \times U \mapsto I\!R^+$, s.d. $d(U_1, U_j) + d(U_j, U_k) \geq d(U_i, U_k)$ für alle $1 \leq i, j \leq n$

   $O.$: Eine Permutation σ über $\{1, \ldots, n\}$, s.d.

$$d(U_{\sigma(n)}, U_{\sigma(1)}) + \sum_{i=1}^{n-1} d(U_{\sigma(i)}, U_{\sigma(i+1)})$$

   ist minimal.

7. TSP-Metric

   $d : U \times U \mapsto I\!R^+$, s.d.

   - $d(u,v) = 0$, wenn $u = v$
   - $d(u,v) = d(v,u)$ (Symmetrie)
   - $d(u,z) + d(z,v) \geq d(u,v)$ (Dreiecksungleichung)

   $(\forall u, v \in U)$

8. TSP (A-B Metric)

   für $1 \in A \subseteq I\!N, B \in I\!N$, $d : U \times U \mapsto I\!R^+$ ist eine Metric, s.d.

   - $d(u,v) \in A$ für alle $u, v \in U, u \neq v$
   - $|\{v | d(u,v) = 1\}| \leq B$ für alle $u \in U$

9. TSP (Euclidean) (alte Bezeichnung: Geometric)

   l_2-Norm:

$$\begin{aligned} U &\subseteq I\!R^2 \\ u_i &= (x_i, y_i) \end{aligned}$$

$$\begin{aligned} u_j &= (x_j, y_j) \\ d(u_i, u_j) &= \sqrt{(x_i - x_j)^2 - (y_i - y_j)^2} \end{aligned}$$

$ETSP := TSP$ (Euklidean)

10. EPM

    (Deciding Existence of a Perfect Matching)

    $I.$: Ein Graph $G = (V, E), E \subseteq P_2(V) := \{A | A \subseteq V, |A| = 2\}$

    $O.$: Gibt es ein perfektes Matching (PM) $M \subseteq E$ von G?

11. KPM (CPM)

    (Constructing a Perfect Matching)

    $I.$: Ein Graph $G = (V, E)$ mit PM

    $O.$: $M \subseteq E$, M ist ein perfektes Matching von G

12. $EBPM$

    (Deciding Existence of a Perfect Matching in Bipartite Graphs)

    $I.$: Ein bipartiter Graph $G = (V, E)$, $V = V_1 \cup V_2, |V_1| = |V_2|$

    $O.$: Gibt es ein perfektes Matching (PM) $M \subseteq E$ von G?

13. $CBPM$

    (Constructing a Perfect Matching in Bipartite Graphs)

    $I.$: Ein bipartiter Graph $G = (V, E)$ mit PM

    $O.$: $M \subseteq E$, M ist ein perfektes Matching von G

14. $MIN\text{-}W\text{-}KPM$

    $I.$: Ein Graph $G = (V, E)$ mit PM und eine Gewichtsfunktion $W : E \to I\!N$ auf den Kanten, so daß die Binärdarstellung von $W_{i,j} = W(\{i, j\})$ polynomiell in $|V|$ ist.

    $O.$: Bestimme ein gewichtsminimales prefektes Matching von G.

15. $MAX\text{-}W\text{-}KPM$

    $I.$: Ein Graph $G = (V, E)$ mit PM und eine Gewichtsfunktion $W : E \to I\!N$ auf den Kanten, so daß die Binärdarstellung von $W_{i,j} = W(\{i, j\})$ polynomiell in $|V|$ ist.

    $O.$: Bestimme ein gewichtsmaximales perfektes Matching von G.

16. *CHINESE-POSTMAN*

    $I.$: Ein Graph $G = (V, E)$ und ein Startknoten $v_0 \in V$.

    $O.$: Eine Rundreise, die in v_0 startet und endet, jede Kante durchläuft und eine minimale Anzahl von Kanten benutzt.

17. *CYCLE-COVER*

    $I.$: Ein Graph $G = (V, E)$ und eine Gewichtsfunktion $W : E \to I\!N$ auf den Kanten, so daß die Binärdarstellung von $W_{i,j} = W(\{i, j\})$ polynomiell in $|V|$ ist.

    $O.$: Gewichtsminimale Überdeckung der Knoten mit disjunkten Cyclen.

18. *SHORTEST-SUPERSTRING*

    $I.$: Endliche Menge von Strings $S \subseteq \Sigma^*$ über einem endlichen Alphabet Σ

    $O.$: Kürzeste String $q \in \Sigma^*$ sein, so daß jeder String $s \in S$ ein Unterstring von q (in Zeichen $s \sqsubseteq q$) ist.

19. $Max - FP$ (Max-Flow-Problem)

    $I.$: Ein Transportnetzwerk mit Kapazitäten $C = (V, E, s, t, c)$

    $O.$: Maximale Fluß f^* in C mit $\overline{V}(c, f^*) = \max_g\{\overline{V}(c, g)\}$ (siehe auch Kapitel 5.5)

20. $A - Max - FP$

    $I.$: Ein Transportnetzwerk mit Kapazitäten $C = (V, E, s, t, c)$ mit $c : E \mapsto A$.

    $O.$: Maximale Fluß f^* in C mit $\overline{V}(c, f^*) = \max_g\{\overline{V}(c, g)\}$

21. *LABEL COVER* (max)

    $I.$:

    (a) Ein regulärer[1] bipartiter Graph $G = (V_1, V_2, E)$.

    (b) Eine natürliche Zahl N, die die Menge $\{1, ..., N\}$ der *Labels* definiert.

    (c) Für jede Kante $e \in E$ eine partielle Funktion $\Pi_e : \{1, ..., N\} \to \{1, ..., N\}$.

    Ein *Labelling* ist durch eine nicht-leere Menge von Labels für jeden Knoten von G definiert. Wir sagen ein Labelling *überdeckt* eine Kante $e = \{u, v\}$ (mit $u \in V_1$ und

[1] Ein bipartiter Graph wird regulär genannt, wenn zwei Konstanten d_1, d_2 existieren, so daß der Grad der Knoten auf der linken Seite d_1 und auf der rechten Seite d_2 ist.

$v \in V_2$), wenn für jedes Label a_2 von v ein Label a_1 von u existiert, so daß

$$\Pi_e(a_1) = a_2.$$

$O.$: Finde Labelling, das ein Label pro Knoten vergibt und den Anteil der überdeckten Kanten maximiert.

22. $LABEL\ COVER$ (min)

    $I.$: Wie bei der Maximierungsversion

    $O.$: Ein Labelling, das alle Kanten überdeckt und dabei

    $$\sum_{v \in V_1} (\text{Anzahl der Labels, die } v \text{ zugeordnet wurden})$$

    minimiert.

23. CP (Clique Problem)

    $I.$: Ein Graph $G = (V, E)$ und $\in I\!N$

    $O.$: $\exists V'$ eine Clique von G mit $|V'| = k$? ($V' \subseteq V$, s.d. $\forall u, v \in V'[\{u, v\} \in E]$)

24. $USAT$ (Unique Sat)

    $I.$: $KNF\ f = c_1 \wedge c_2 \wedge \ldots \wedge c_m$ mit $|\{s \in \{0,1\}^n | f(s) = 1\}| \leq 1$

    $O.$: $\exists s \in \{0,1\}^n [f(s) = 1]$?

25. UCP (Unique Clique Problem)

    $I.$: Ein Graph $G = (V, E)$ und $k \in I\!N$ mit # der Cliquen der Größe k $\#_c^k(G) \leq 1$

    $O.$: $\exists V'$, eine Clique der Größe k von G?

26. $MAX\text{-}CLIQUE$

    $I.$: Ein Graph $G = (V, E)$

    $O.$: Max-Clique(G) $= \max\{|V'| | V' \subseteq V, V'$ ist eine Clique von $G\}$

27. $MAX\text{-}CUT$

    $I.$: Ein Graph $G = (V, E)$

    $O.$: Eine Partionierung von V in $V = V_1 \cup V_2$, ($V_1 \cap V_2 = \emptyset$), s.d. $|E(V_1, V_2)|$ ist Maximum.

28. MAX-$FUNCTIONAL$-k-SAT

    $I.$: m boolesche Funktionen $F_1, \ldots, F_m : \mathbb{B}^n \mapsto \mathbb{B}$, die von höchstens k Variablen abhängen

    $O.$: Konstruiere $s \in \mathbb{B}^n$, s.d.

$$|\{i|f_i(s) = TRUE\}| = \max_{s' \in \mathbb{B}^n} |\{i|f_i(s') = TRUE\}|$$

29. $\#SAT$ ($\#kSAT$)

    $I.$: f eine KNF (k-KNF)

    $O.$: $|\{s \in \{0,1\}^n | f(s) = 1\}|$

30. $\#DNF$ ($\#kDNF$)

    $I.$: f eine DNF (k-DNF)

    $O.$: $|\{s \in \{0,1\}^n | f(s) = 1\}|$

31. $\#XOR$ ($\#kXOR$)

    $I.$: $f \in GF[2][x_1, \ldots, x_n]$, (Deg $f \leq k$)

    $O.$: $|\{s|f(s) = 1\}|$

32. $\#IS$

    $I.$: Ein Graph $G = (V, E)$

    $O.$: $\#IS(G) = |\{I \subseteq V | I$ ist unabhängig in $G.\}|$

33. $\#kIS$

    $I.$: Ein Graph $G = (V, E)$ mit maximalem Knotengrad $\leq k$

    $O.$: $\#IS(G) = |\{I \subseteq V | I$ ist unabhängig in $G.\}|$

34. $\#CLIQUE$

    $I.$: Ein Graph $G = (V, E)$

    $O.$: $\#CLIQUE(G) = |\{C \subseteq V | C$ ist eine Clique von $G.\}|$

Literaturverzeichnis

[A95] Alizadeh, F., *Combinatorial optimization with intererior point methods and semidefinite matrices*, SIAM Journal on Optimization **5** (1995), pp. 13–51.

[AFWZ95] Alon, N., Feige, U., Wigderson, A., Zuckerman, D., *Derandomizing Graph Products*, Computational Complexity **5** (1995), pp. 60–75.

[AS92a] Alon, N., Spencer, J., *The Probabilistic Method*, Wiley Interscience, 1992.

[A96] Arora, S., *Polynomial Time Approximation Schemes for Euclidean TSP and other Geometric Problems*, Proc. 37^{th} IEEE FOCS (1996), pp. 2–11.

[AKK95] Arora, S., Karger, D., Karpinski, M., *Polynomial Time Approximation Schemes for Dense Instances of NP-Hard Problems*, Proc. 27^{th} ACM STOC (1995), pp. 284–293.

[ALM+92] Arora, S., Lund, C., Motwani, R., Sudan, M., Szegedy, M., *Proof Verification and Hardness of Approximation Problems*, Proc. 33^{rd} IEEE FOCS (1992), pp. 13–22.

[AS92b] Arora, S., Safra, S., *Probabilistic Checking of Proofs: A New Characterization of NP*, Proc. 33^{rd} IEEE FOCS (1992), pp. 2–13.

[BFLS91] Babai, L., Fortnow, L., Levin, L., Szegedy, M., *Checking Computations in Polylogarithmic Time*, Proc. 23^{rd} IEEE FOCS (1991), pp. 21–31.

[BGKW88] Ben-Or, M., Goldwasser, S., Kilian, J., Wigderson, A., *Multi-Prover Interactive Proofs: How to Remove Intractibility Assumptions*, Proc. 20^{th} ACM STOC (1988), pp. 113–131.

[BK98] Berman, P., Karpinski, M., *On Some Tighter Inapproximability Results, Further Improvements*, Research Report TR98-065, 1998, appeared also in Proc. 26th ICALP(1999), LNCS Vol. 1644, Springer, 1999, pp. 200–209.

[BJL$^+$91] Blum, A., Jiang, T., Li, M., Tromp, J., Yannakakis, M., *Linear approximation of shortest superstrings*, Proc. 23^{rd} ACM STOC (1991), pp. 328–336.

[B67] Blum, M., *A Machine-Independent Theory of the Complexity of Recursive Functions*, Journal of the ACM **14**(2) (1967), pp. 322–336.

[BLR90] Blum, M., Luby, M., Rubinfeld, R., *Self-Testing/Correcting with Applications to Numerical Problems*, ACM STOCS **22** (1990), pp. 73–83.

[BDK05] Bordewich, M., Dyer, M., Karpinski, M., *Path Coupling Using Stopping Times and Counting Independent Sets and Colourings in Hypergraphs*, Proc. 15th FCT (2005), LNCS Vol. 3623, Springer (2005), pp. 19–31; journal version to appear in Random Structures and Algorithms, 2007

[BDK06] Bordewich, M., Dyer, M., Karpinski, M., *Stopping Times, Metrics and Approximate Counting*, Proc. 33rd ICALP (2006), pp. 108–119.

[C74] Chung, K., *Elementary Probability Theory with Sochastic Processes*, Springer-Verlag, 1974.

[CKST99] Crescenzi, P., Kann, V., Silvestri, R., Trevisan, L. *Structure in approximation classes* SIAM J. Computing 28:1759-1782, 1999. COCOON 95, 539-548, LNCS 959, 1995.

[C71] Cook, S., *The complexity of theorem-proving procedures*, Proc. 3^{rd} ACM STOC (1971), pp. 151–158.

[DHK93] Dahlhaus, E., Hajnal, P., Karpinski, M., *On the Parallel Complexity of Hamiltonian Cycle and Matching Problem on Dense Graphs*, Journal of Algorithms **15** (1993), pp. 367–384.

[DFJ98] Dyer, M., Frieze, A., Jerrum, M., *On Counting independent sets in sparse graphs*, (Preprint), University of Leeds, 1998.

[F74] Fagin, R., *Generalized first-order spectra and polynomial-time recognizable sets*, Complexity of Computations, pp. 151–158, R. Karp, 1974.

[FKL00] Feige, U., Karpinski, M., Langberg, M., *Improved Approximation of MAX-CUT on Graphs of Bounded Degree*, Research Report ECCC TR00-021, 2000.

[FK98] Fernandez de la Vega, W., Karpinski, M., *On Approximation Hardness of Dense TSP and other Path Problems*, Research Report ECCC TR 98-024, 1998, also in Information Processing Letters 70(1999), pp. 53–55.

[FRM88] Fortnow, L., Rompel, J., M.Sipser, *On the power of multi-prover interactive protocols*, Proc. 3^{rd} ACM STOC (1988), pp. 156–161.

[GG81] Gabber, O., Galil, Z., *Explicit Construction of Linear Sized Superconcentrators*, Journal of Computing Systems Science **22** (1981), pp. 407–420.

[GJ79] Garey, M. R., Johnson, D. S., *Computers and Intractability. A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

[G77] Gill, J., *Computational Complexity of Probabilistic Turing Machines*, SIAM Journal on Computing **6**(4) (1977), pp. 675–695.

[GW94] Goemans, M., Williamson, D., *0,878-Approximation Algorithms for MAXCUT and MAX 2SAT*, Proc. 26^{th} ACM STOC (1994), pp. 422–431.

[GK91] Grigoriev, D. Y., Karpinski, M., *An Approximation Algorithm for the Number of Zeros of Arbitrary Polynomials over GF[q]*, Proc. 32^{nd} IEEE FOCS (1991), pp. 662–669.

[GLS88] Grötschel, M., Lovász, L., Schrijver, A., *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag, 1988.

[H97a] Håstad, J., *Some Optimal Inapproximability Results*, Proc. 28^{th} ACM STOC (1997), pp. 1–10.

[H97b] Hochbaum, D., ed, *Approximation Algorithms for NP-Hard Problems*, PWS Publ. Co., 1997.

[K72] Karp, R., *Reducibility Among Combinatorial Problems*, Miller, R., Thatcher, J., ed, Complexity of Computer Computations, pp. 85–103, 1972.

[KLM89] Karp, R. M., Luby, M., Madras, N., *Monte-Carlo Approximation Algorithms for Enumeration Problems*, Journal of Algorithms **10** (1989), pp. 429–448.

[K91] Karpinski, M., *Effiziente Algorithmen und Komplexitätstheorie (Algebraische Interpolations- und Zählalgorithmen) (ausgearbeitet von Kai Werther)*, Vorlesung an der Universität Bonn, 1991.

[K97] Karpinski, M., *Polynomial Time Approximation Schemes for Some Dense Instances of NP-Hard Optimization Problems*, Proc. 1^{st} Symposium on Randomization and Approximation Techniques in Computer Science, RANDOM'97, Lecture Notes in Computer Science Vol.1269, Springer Verlag (1997), pp. 1–14.

[KL91] Karpinski, M., Lhotzky, B., *An (ϵ, δ)-Approximation Algorithm of the Number of Zeros for a Multilinear Polynomial over GF[q]*, Technical Report TR-91-022, International Computer Science Institute Berkeley, 1991.

[KL93] Karpinski, M., Luby, M., *Approximating the Number of Solutions of a GF[2] Polynomial*, Journal of Algorithms **14** (1993), pp. 280–287.

[KR98] Karpinski, M., Rytter, W., *Fast Parallel Algorithms for Graph Matching Problems*, Oxford University Press, 1998.

[KV85] Karpinski, M., Verbeek, R., *There is No Polynomial Deterministic Space Simulation of Probabilistic Space with a Two-Way Random-Tape Generator*, Information and Control **67**(1–3) (1985), pp. 158–162.

[KWZ97] Karpinski, M., Wirtgen, J., Zelikovsky, A., *An Approximation Algorithm for the Bandwidth Problem on Dense Graphs*, Proc. 1^{st} RALCOM (1997), pp. 1–14.

[KZ97a] Karpinski, M., Zelikovsky, A., *Approximating Dense Cases of Covering Problems*, Technical Report TR-97-004, ECCC, 1997.

[KZ97b] Karpinski, M., Zelikovsky, A., *New Approximation Algorithms for the Steiner Tree Problems*, Journal of Combinatorial Optimization (1997), pp. 47–65.

[KZ98] Karpinski, M., Zelikovsky, A., *Approximating Dense Cases of Covering Problems*, Proc. DIMACS Workshop on Network Design: Connectivity and Facilites Location, DIMACS Series Volume **40** (1998), pp. 169–178.

[LPS86] Lubotzky, A., Phillips, R., Sarnak, P., *Explicit Expanders and the Ramanujan Conjectures*, Proc. 18^{th} ACM STOC (1986), pp. 240–246.

[LPS88] Lubotzky, A., Phillips, R., Sarnak, P., *Ramanujan Graphs*, Combinatorica **8** (1988), pp. 261–277.

[LV97] Luby, M., Vigoda, E., *Approximately Counting up to Four (Extended Abstract)*, Proc. 29^{th} ACM STOC (1997), pp. 682–687.

[M73] Margulis, G., *Explicit Construction of Concentrators*, Problemy Peredachi Informatsii **9**(4) (1973), pp. 71–80.

[MR95] Motwani, R., Raghavan, P., *Randomized Algorithms*, Cambridge University Press, 1995.

[MVV87] Mulmuley, K., Vazirani, U. V., Vazirani, V. V., *Matching is as Easy as Matrix Inversion*, Proc. 19^{th} ACM STOC (1987), pp. 345–354.

[PY91] Papadimitriou, C., Yannakakis, M., *Optimization, Approximation, and Complexity Classes*, Journal of Computing Systems Science **43** (1991), pp. 425–440.

[PY93] Papadimitriou, C., Yannakakis, M., *The Traveling Salesman Problem with Distances One and Two*, Math. of Operations Research **18** (1993), pp. 1–12.

[R88] Raghavan, P., *Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs*, JCSS **37** (1988), pp. 130–143.

[S80a] Schönhage, A., *Storage Modification Machines*, SIAM Journal on Computing **9**(3) (1980), pp. 490–507.

[S86] Schrijver, A., *Theory of Linear and Integer Programming*, Wiley & Sons, 1986.

[S80b] Schwartz, J., *Fast Probabilistic Algorithms for Verification of Polynomial Identities*, Journal of the ACM **27**(4) (1980), pp. 701–717.

[V79a] Valiant, L. G., *The Complexity of Computing the Permanent*, Theoretical Computer Science **8** (1979), pp. 189–201.

[V79b] Valiant, L. G., *The Complexity of Enumeration and Reliability Problems*, SIAM Journal on Computing **8**(3) (1979), pp. 410–421.

[VB94] Vandenberghe, L., Boyd, S., *Semidefinite Programming*, SIAM Review **38**(1) (1996), pp. 49–95