Inhaltsverzeichnis

Wie man Beweise verifiziert, ohne sie zu lesen

Hans Jürgen Prömel und Angelika Steger

Ein Traum eines jeden Studenten, Doktoranden und Professors: Die Richtigkeit eines komplizierten Beweises zu verifizieren, ohne ihn mühselig Zeile für Zeile zu lesen und zu verstehen. Glaubt man einem Artikel der New York Times, so ist die Wissenschaft diesem Ziel schon sehr nahe gekommen:

> The New York Times — April 7, 1992
>
> New Shortcut Found For Long Math Proofs
>
> In a discovery that overturns centuries of mathematical tradition, a group of graduate students and young researchers has discovered a way to check even the longest and most complicated proof by scrutinizing it in just a few spots...

Doch was auf den ersten Blick aussieht wie ein immenser Fortschritt, hat auch dramatische Nachteile. In besagtem Artikel heißt es weiter:

> Using this new result, the researchers have already made a landmark discovery in computer science. They showed that it is impossible to compute even approximate solutions for a large group of practical problems that have long foiled researchers...

Das Ergebnis, auf das sich der erste Teil dieses Zitates aus der New York Times bezieht, läßt sich abkürzend schreiben als

$$\mathcal{NP} = \mathcal{PCP}(\log n, 1).$$

Dieses Resultat wird gemeinhin als eines der spektakulärsten Ergebnisse der Theoretischen Informatik innerhalb der letzten Jahre angesehen. Im ersten Teil dieses Artikels wollen wir darstellen, was dieses Ergebnis genau besagt. Anschließend werden einige überraschende und weitreichende Konsequenzen dieses Ergebnisses für die Approximierbarkeit kombinatorischer Optimierungsprobleme vorgestellt. Konsequenzen, auf die im zweiten Teil des Zitats Bezug genommen wird.

1.1 Probabilistisch verifizierbare Beweise

Was besagt das Resultat $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$ eigentlich? — Die Buchstabenfolge $\mathcal{PCP}$ steht für *probabilistically checkable proofs*. Ganz grob gesprochen geht es also um eine neue Charakterisierung der Klasse $\mathcal{NP}$ bei der die "Beweise" unter Zuhilfenahme von "wahrscheinlichkeitstheoretischen Ansätzen" überprüft werden.[1]

Im ersten Teil dieses Artikels werden wir versuchen, Schritt für Schritt die Aussage und die Bedeutung der Gleichung $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$ zu erläutern. Dabei wird nicht an jeder Stelle der formalen Korrektheit Genüge getan, sondern bisweilen an die Intuition appelliert werden. Für die formalen Definitionen der grundlegenden Begriffe verweisen wir auf Standardreferenzen der Theoretischen Informatik, beispielsweise auf Wegener (1993). Im Sinne einer geschlossenen Darstellung beginnen wir mit der Definition der bekannten Klassen $\mathcal{P}$, $\mathcal{NP}$ und $\mathcal{RP}$. Die von uns gewählte Form ist allerdings teilweise unkonventionell, wird sich im folgenden aber als sehr zweckmäßig erweisen.

In der Komplexitätstheorie untersucht man üblicherweise *Entscheidungs*probleme – im Unterschied zu *Optimierungs*problemen, die uns im zweiten Abschnitt dieses Artikels beschäftigen werden. Optimierungsprobleme sind dadurch gekennzeichnet, daß es zu jeder Eingabe I (aus einer Menge von Objekten $\mathcal{I}$) eine Menge von Lösungen gibt, unter denen eine zu bestimmen ist, die bezüglich einer vorgegebenen Bewertungsfunktion minimal beziehungsweise maximal ist. Bei Entscheidungsproblemen hingegen fragt man nur danach, ob eine Lösung mit einer gewissen Eigenschaft überhaupt existiert.

Sei $\mathcal{I}$ eine Menge von Objekten, von der wir im folgenden immer annehmen werden, daß von einem gegebenen Objekt I in polynomialer Zeit getestet werden kann, ob $I \in \mathcal{I}$. Zu jedem $I \in \mathcal{I}$ sei nun $\mathcal{L}(I)$ eine (möglicherweise leere) Menge. Weiter bezeichnen wir mit $\mathcal{I}_0 = \{I \in \mathcal{I} \mid \mathcal{L}(I) \neq \emptyset\}$ die Menge derjenigen Objekte, die mindestens eine "Lösung" besitzen. Das zugehörige Entscheidungsproblem ist dann also: *Gegeben ein $I \in \mathcal{I}$, gilt $I \in \mathcal{I}_0$?*

Schauen wir uns zwei Beispiele an, die auch im folgenden von Bedeutung sein werden:

k-SATISFIABILITY (kSAT):

Eingabe: *Eine Boolesche Formel in konjunktiver Normalform, in der jede Klausel die Disjunktion von genau k Literalen ist.*
Frage: *Gibt es eine Belegung der Variablen, so daß alle Klauseln erfüllt sind?*

und

k-COLORING (kCOL):

Eingabe: *Ein endlicher Graph $G = (V, E)$.*
Frage: *Gibt es eine Färbung $c : V \to \{1, \ldots, k\}$ der Knoten des Graphen mit k Farben, so daß je zwei benachbarte Knoten verschieden gefärbt sind, das heißt, daß für alle Kanten $\{x, y\} \in E$ gilt $c(x) \neq c(y)$?*

[1] Tatsächlich werden diese wahrscheinlichkeitstheoretischen Ansätze nichts weiter sein, als das wiederholte Werfen einer Münze!

Zu diesen Entscheidungsproblemen gehörige Optimierungsprobleme werden wir im zweiten Teil dieses Artikels betrachten.

Man sagt, ein Entscheidungsproblem $\mathcal{I}$ liegt in $\mathcal{P}$, falls es eine Turingmaschine gibt, die zu einem gegebenen $I \in \mathcal{I}$ in polynomial vielen Schritten (in der Länge $|I|$ der Eingabe) entscheidet, ob $I \in \mathcal{I}_0$, das heißt, die ACCEPT ausgibt, falls $I \in \mathcal{I}_0$, und REJECT sonst.

Bekannte Beispiele für Probleme in $\mathcal{P}$ sind die Spezialfälle 2SAT und 2COL der oben angeführten Entscheidungsprobleme. Für die Probleme 3SAT und 3COL konnte man andererseits bis heute nicht zeigen, daß sie zu $\mathcal{P}$ gehören. Sie liegen jedoch zumindest in der Klasse $\mathcal{NP}$.

Um die Klasse $\mathcal{NP}$ zu definieren, machen wir die Turingmaschine etwas "mächtiger", indem wir ihr erlauben, zusätzlich zur Eingabe I auch einen (potentiellen) Beweis B zu lesen. Ein Entscheidungsproblem liegt dann in $\mathcal{NP}$, falls es eine solche, wie man sagt, *nichtdeterministische* Turingmaschine M gibt, die zu einem gegebenen $I \in \mathcal{I}$ unter Zuhilfenahme eines entsprechenden Beweises B in polynomial (in $|I|$) vielen Schritten entscheidet, ob $I \in \mathcal{I}_0$. Aus Gründen, die etwas später klar werden, nennen wir solch eine nichtdeterministische Turingmaschine im folgenden auch einen *Verifier*. Graphisch kann man sich diesen wie folgt veranschaulichen:

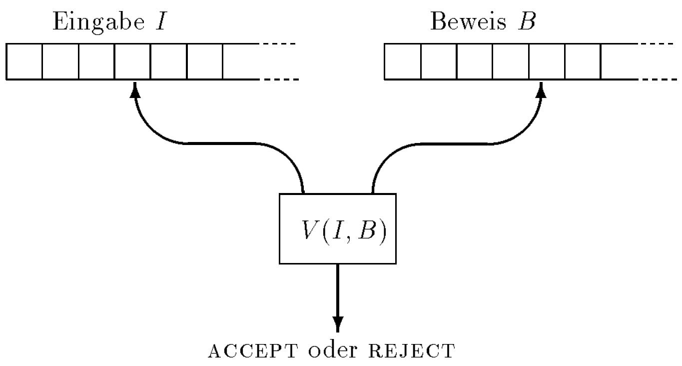

Abb. 1.1. Die Klasse $\mathcal{NP}$ (nondeterministic polynomial time).

Die Klasse $\mathcal{NP}$ besteht dann also aus allen Problemen $\mathcal{I}$, für die es einen polynomial zeitbeschränkten Verifier V gibt, so daß

$$\begin{array}{llcll} \forall I \in \mathcal{I}_0 & \exists B_0 : & V(I, B_0) & = & \text{ACCEPT}, \quad \text{und} \\ \forall I \notin \mathcal{I}_0 & \forall B : & V(I, B) & = & \text{REJECT}. \end{array}$$

Offensichtlich ist $\mathcal{P} \subseteq \mathcal{NP}$. Für Probleme aus $\mathcal{P}$ kann der Verifier die Frage "$I \in I_0$?" ja sogar entscheiden, ohne den Beweis überhaupt zu lesen. Die Probleme 3SAT und 3COL gehören offenbar ebenfalls zur Klasse $\mathcal{NP}$. Hier besteht der "Beweis" B_0 einfach aus einer erfüllenden Wahrheitsbelegung der

Variablen beziehungsweise aus einer zulässigen 3-Färbung $c : V \to \{1,2,3\}$ der Knoten von G. In beiden Fällen muß der Verifier nur überprüfen, ob für die vorliegende Wahrheitsbelegung beziehungsweise Färbung alle Klauseln erfüllt sind beziehungsweise die Endpunkte aller Kanten verschieden gefärbt sind. Und dies ist ganz offensichtlich in polynomialer Zeit möglich.

Allgemein nimmt man an, daß $\mathcal{P} \neq \mathcal{NP}$ gilt. Vereinfacht ausgedrückt, man vermutet, daß es kombinatorische Probleme gibt, für die es schwerer ist, eine Aussage zu beweisen, als einen gegebenen Beweis zu verifizieren.

Als Hilfsmittel zur Strukturierung der Klasse $\mathcal{NP}$ hat man den Begriff der polynomialen (Turing-)Reduzierbarkeit eingeführt, der Aussagen der Form gestattet "*Wenn dieses Problem in $\mathcal{P}$ liegt, dann liegt auch jenes Problem in $\mathcal{P}$*". Ein Problem $\mathcal{I} \in \mathcal{NP}$ heißt $\mathcal{NP}$-vollständig, falls aus $\mathcal{I} \in \mathcal{P}$ bereits $\mathcal{NP} \subseteq \mathcal{P}$ folgt. $\mathcal{NP}$-vollständige Probleme sind also in gewissem Sinne die schwierigsten Probleme in $\mathcal{NP}$. 3Sat und 3Col sind $\mathcal{NP}$-vollständig. Ein Problem $\mathcal{I}$ mit der Eigenschaft $\mathcal{I} \in \mathcal{P} \Longrightarrow \mathcal{NP} \subseteq \mathcal{P}$ nennt man $\mathcal{NP}$-schwer.

Eine andere Möglichkeit, eine (Standard-)Turingmaschine "mächtiger" zu machen, ist es, ihr – idealisiert ausgedrückt – das Werfen einer Münze zu gestatten. Erlaubt man einer solchen sogenannten *randomisierten* Turingmaschine zudem einen einseitigen Fehler zu machen, erhält man die Klasse $\mathcal{RP}$. Zur genauen Definition verwenden wie wieder unsere obige Sprechweise eines *Verifiers* (statt einer Turingmaschine), dem wir diesmal jedoch statt dem Zugriff auf einen "Beweis" den Zugriff auf eine Folge τ von zufälligen Bits gewähren (die sozusagen die Ergebnisse der sukzessiven "Münzwurfe" repräsentieren).

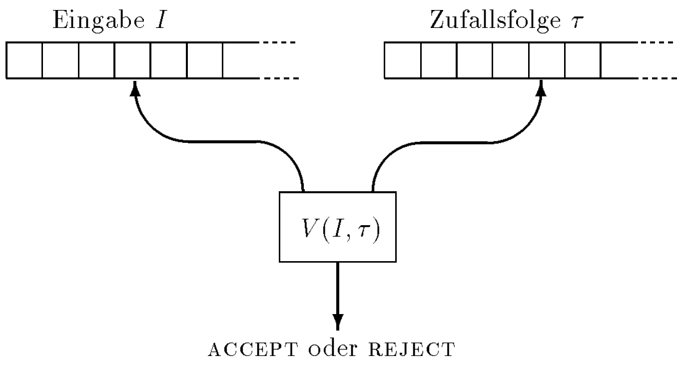

Abb. 1.2. Die Klasse $\mathcal{RP}$ (randomized polynomial time).

Die Klasse $\mathcal{RP}$ besteht aus allen Problemen $\mathcal{I}$, für die es einen polynomial zeitbeschränkten Verifier V gibt, so daß

$$\begin{array}{ll} \forall I \in \mathcal{I}_0: & \mathrm{Prob}_\tau\left[V(I,\tau)=\textsc{accept}\right] \;\geq\; \frac{1}{2}, \quad \text{und} \\ \forall I \notin \mathcal{I}_0: & \mathrm{Prob}_\tau\left[V(I,\tau)=\textsc{reject}\right] \;=\; 1. \end{array}$$

Ist also $I \notin \mathcal{I}_0$, dann *muß* der Verifier V die Eingabe I verwerfen, unabhängig von der Zufallsfolge τ, die V liest. Ist jedoch $I \in \mathcal{I}_0$, so muß V die Eingabe lediglich für mindestens die Hälfte aller Zufallsfolgen akzeptieren, für die übrigen darf er sie auch verwerfen. – Ein Wort noch zur Notation: Hier und im folgenden bedeutet $\mathrm{Prob}_\tau[\ldots]$ einfach, daß die Wahrscheinlichkeit bezüglich einer zufällig (gleichverteilt) gewählten Folge τ gemeint ist. Da der Verifier nur polynomial viel Zeit verwenden darf, kann er insbesondere auch nur polynomial viele Bits von τ lesen. Wir dürfen also ohne Einschränkung annehmen, daß τ nur aus polynomial vielen Bits besteht.

Offensichtlich ist $\mathcal{RP} \subseteq \mathcal{NP}$. Als "Beweis" verlangt der $\mathcal{NP}$-Verifier einfach eine Folge τ bei dem der $\mathcal{RP}$-Verifier akzeptiert. Ein Problem, welches in $\mathcal{RP}$ liegt, von dem jedoch weder bekannt ist, ob es $\mathcal{NP}$-vollständig ist, noch ob es in $\mathcal{P}$ liegt, ist

<u>Composite:</u>

`Eingabe:` *Eine natürliche Zahl n.*
`Frage:` *Gibt es natürliche Zahlen $2 \leq p \leq q < n$, so daß $n = p \cdot q$?*

Aus dem Primzahltest von Solovay und Strassen folgt, daß Composite in $\mathcal{RP}$ liegt.

Der Vollständigkeit halber fügen wir noch die Definitionen einiger weiterer Komplexitätsklassen an, auf die wir im folgenden Bezug nehmen werden. Mit DTIME$(t(n))$ bezeichnet man die Klasse aller Entscheidungsprobleme, die von einer Turingmaschine entschieden werden können, die für jede Eingabe der Länge n höchstens $t(n)$ Schritte benötigt. Insbesondere gilt also $\mathcal{P} = \mathrm{DTIME}(poly(n))$.

Analog bezeichnet man mit NTIME$(t(n))$ die Klasse aller Entscheidungsprobleme, die von einer nichtdeterministischen Turingmaschine entschieden werden können, die für jede Eingabe der Länge n höchstens $t(n)$ Schritte benötigt. Insbesondere gilt also $\mathcal{NP} = \mathrm{NTIME}(poly(n))$. Man setzt $\mathcal{NEXP} := \mathrm{NTIME}(2^{poly(n)})$.

Die Definitionen der Klassen $\mathcal{NP}$ und $\mathcal{RP}$ sind offenbar unsymmetrisch. Vertauscht man hier die Rolle der Mengen $\mathcal{I}_0$ und $\mathcal{I} \setminus \mathcal{I}_0$, so erhält man die Klassen *co*$\mathcal{NP}$ beziehungsweise *co*$\mathcal{RP}$. Beispielsweise besteht letztere also aus allen Problemen $\mathcal{I}$, für die es einen polynomial zeitbeschränkten Verifier V gibt, so daß

$$\begin{array}{ll} \forall I \in \mathcal{I}_0: & \mathrm{Prob}_\tau\left[V(I,\tau)=\textsc{accept}\right] \;=\; 1, \quad \text{und} \\ \forall I \notin \mathcal{I}_0: & \mathrm{Prob}_\tau\left[V(I,\tau)=\textsc{reject}\right] \;\geq\; \frac{1}{2}. \end{array}$$

1.2 Das Resultat $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$

Warum sind die Klassen $\mathcal{RP}$ und $\mathcal{NP}$ allem Anschein nach größer als die Klasse $\mathcal{P}$? In beiden Fällen hat der Verifier, wie zum Erkennen der Klasse

$\mathcal{P}$, nur polynomial beschränkte Zeit zur Verfügung. Einmal hat er jedoch zusätzlich Zugriff auf eine Folge τ zufälliger Bits und braucht eine Eingabe, die eine zulässige Lösung hat, nur mit Wahrscheinlichkeit mindestens 1/2 zu akzeptieren. Das andere Mal hat er zusätzlich Zugriff auf einen Kandidaten für einen Beweis und braucht nur zu entscheiden, ob der Kandidat wirklich das leisten kann, was er vorgibt zu leisten. Was passiert nun, wenn wir einem polynomial zeitbeschränkten Verifier gleichzeitig beide Hilfsmittel erlauben?

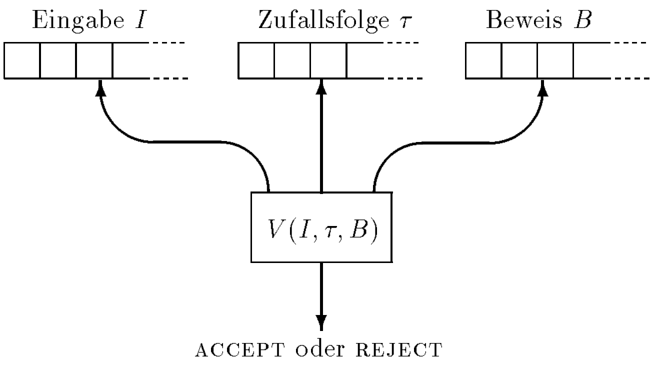

Abb. 1.3. Die Klassen $\mathcal{PCP}(\cdot,\cdot)$ (probabilistically checkable proofs).

Die Klasse $\mathcal{PCP}(poly(n), poly(n))$ besteht aus allen Problemen $\mathcal{I}$, für die es einen polynomial zeitbeschränkten Verifier V gibt, so daß

$$\begin{array}{llll} \forall I \in \mathcal{I}_0 \quad \exists B_0: & \mathrm{Prob}_\tau\,[V(I,\tau,B_0) = \text{ACCEPT}] & = & 1, \quad \text{und} \\ \forall I \notin \mathcal{I}_0 \quad \forall B\ : & \mathrm{Prob}_\tau\,[V(I,\tau,B) = \text{REJECT}] & \geq & \frac{1}{2}. \end{array}$$

Wie mächtig ist nun die Klasse $\mathcal{PCP}(poly(n), poly(n))$? Sehr mächtig! Babai, Fortnow und Lund (1991) bewiesen:

Theorem 1.1.

$$\mathcal{PCP}(poly(n), poly(n)) \;=\; \mathcal{NEXP}.$$

Die grundlegende Idee ist es nun, die Macht dieses Verifiers einzuschränken, indem man ihm nicht mehr beliebige (also polynomial) viele zufällige Bits zur Verfügung stellt, sondern nur noch eine beschränkte Anzahl, und ihm auch nur noch eine beschränkte Anzahl von Anfragen an den (polynomial langen) potentiellen Beweis erlaubt.

Seien $r(n)$ und $q(n)$ zwei Funktionen. Ein $(r(n), q(n))$-beschränkter Verifier ist ein polynomial zeitbeschränkter Verifier mit Zugriff auf eine Eingabe I der Länge n und auf $\mathcal{O}(r(n))$ zufällige Bits τ, der höchstens $\mathcal{O}(q(n))$ Bits eines Beweises B erfragt. Der Verifier arbeitet nichtadaptiv, er liest also zunächst die Eingabe I und die $\mathcal{O}(r(n))$ zufälligen Bits τ. Auf Grund dieser Informationen berechnet er dann die $\mathcal{O}(q(n))$ Positionen der Bits des Beweises B, die er erfragen will. In Abhängigkeit der erhaltenen Antworten akzeptiert oder verwirft er daraufhin dann die Eingabe.

Die Klasse $\mathcal{PCP}(r(n), q(n))$ ist die Menge aller Probleme $\mathcal{I}$, für die es einen $(r(n), q(n))$-beschränkten Verifier V gibt, so daß

$$\begin{array}{llll} \forall I \in \mathcal{I}_0 & \exists B_0 : & \mathrm{Prob}_\tau\left[V(I,\tau,B_0) = \text{ACCEPT}\right] & = 1, \quad \text{und} \\ \forall I \notin \mathcal{I}_0 & \forall B : & \mathrm{Prob}_\tau\left[V(I,\tau,B) = \text{REJECT}\right] & \geq \frac{1}{2}. \end{array}$$

Für richtige Aussagen gibt es also einen Beweis der immer akzeptiert wird (completeness); jeder Versuch, eine falsche Aussage zu beweisen, scheitert andererseits mindestens mit Wahrscheinlichkeit 1/2 (soundness). Die folgenden drei Gleichungen folgen unmittelbar aus der Definition:

$$\begin{aligned} \mathcal{PCP}(0,0) &= \mathcal{P}, \\ \mathcal{PCP}(poly(n),0) &= co\mathcal{RP}, \\ \mathcal{PCP}(0,poly(n)) &= \mathcal{NP}. \end{aligned}$$

Zuweilen ist es vorteilhaft, die Wahrscheinlichkeit, mit der ein falscher Beweis akzeptiert wird, zu ändern. Durch eine wiederholte Ausführung des Verifiers ist dies auch einfach möglich: Führt man den Verifier mit Fehlerwahrscheinlichkeit 1/2 beispielsweise zweimal hintereinander aus (bezüglich verschiedener Zufallsfolgen τ) und akzeptiert nur dann, wenn er bei beiden Aufrufen akzeptiert hat, erhält man einen Verifier mit Fehlerwahrscheinlichkeit 1/4. Eine geänderte Fehlerwahrscheinlichkeit notieren wir, wo nötig, durch einen Index: Die Klasse $\mathcal{PCP}_\varepsilon(r(n), q(n))$ enthält also alle Probleme $\mathcal{I}$, für die es einen $(r(n), q(n))$-beschränkten Verifier V gibt, so daß

$$\begin{array}{llll} \forall I \in \mathcal{I}_0 & \exists B_0 : & \mathrm{Prob}_\tau\left[V(I,\tau,B_0) = \text{ACCEPT}\right] & = 1, \quad \text{und} \\ \forall I \notin \mathcal{I}_0 & \forall B : & \mathrm{Prob}_\tau\left[V(I,\tau,B) = \text{REJECT}\right] & \geq 1-\varepsilon. \end{array}$$

Angenommen, $\mathcal{I}$ ist ein Problem in $\mathcal{PCP}(r(n), q(n))$. Dann gibt es einen $(r(n), q(n))$-beschränkten Verifier V für $\mathcal{I}$. Aus diesem Verifier kann man dann wie folgt eine herkömmliche nichtdeterministische Turingmaschine M für $\mathcal{I}$ konstruieren. Sei $I \in \mathcal{I}$ eine Eingabe der Länge n und B ein Beweis. Für jede der $2^{\mathcal{O}(r(n))}$ vielen Zufallsfolgen der Länge $\mathcal{O}(r(n))$ simuliert M (in jeweils $poly(n)$ vielen Schritten) die Rechnung des Verifiers V und akzeptiert I genau dann, wenn V die Eingabe I für *jede* Zufallsfolge τ akzeptiert. M ist somit eine $\mathcal{O}(poly(n) \cdot 2^{\mathcal{O}(r(n))})$-zeitbeschränkte nichtdeterministische Turingmaschine, die I genau dann akzeptiert, wenn $I \in \mathcal{I}_0$. Wir haben also die folgende Inklusion gezeigt:

Proposition 1.1.

$$\mathcal{PCP}(r(n), q(n)) \subseteq NTIME(poly(n) \cdot 2^{\mathcal{O}(r(n))}).$$

Wählt man $r(n) = \log n$ und berücksichtigt, daß NTIME$(poly(n)) = \mathcal{NP}$ gilt, beziehungsweise wählt man $r(n) = \log n$ und $q(n) = 1$, so erhält man als Folgerung:

Korollar 1.1.

$$\mathcal{PCP}(\log n, poly(n)) \; = \; \mathcal{NP} \qquad \textit{und} \qquad \mathcal{PCP}(\log n, 1) \; \subseteq \; \mathcal{NP}.$$

Die Umkehrung der zweiten Inklusion ist nun das angekündigte Resultat.

Theorem 1.2 (Arora, Lund, Motwani, Sudan, Szegedy, 1992).

$$\mathcal{NP} \; = \; \mathcal{PCP}(\log n, 1).$$

Dieser Satz besagt also, daß es zu jedem Problem $\mathcal{I}$ in $\mathcal{NP}$ einen Verifier gibt, der zu jeder Eingabe nur konstant viele Stellen des Beweises lesen muß, die er unter Zuhilfenahme von $\mathcal{O}(\log n)$ vielen zufälligen Bits auswählt, um mit hoher Wahrscheinlichkeit richtig zu entscheiden, ob eine gegebene Eingabe "wahr" ist oder nicht.

Wieviele Anfragen an den Beweis sind nun mindestens nötig, um das $\mathcal{PCP}$-Resultat zu beweisen? Bellare, Goldreich und Sudan (1995) zeigten, daß

$$\begin{array}{lcll} \mathcal{NP} & \subseteq & \mathcal{PCP}_{\frac{1}{2}}(\log n, 19) & \text{und} \\ \mathcal{NP} & \subseteq & \mathcal{PCP}_{\varepsilon}(\log n, 3) & \text{für } \; \varepsilon = 0,902, \end{array}$$

wobei "19" und "3" tatsächlich Konstanten (und nicht asymptotisch konstante Funktionen) sind. Dies ist fast bestmöglich, da man leicht zeigen kann, daß

$$\mathcal{PCP}_{\varepsilon}(\log n, 2) \; \subseteq \; \mathcal{P} \qquad \text{für jedes} \quad \varepsilon < 1 \; .$$

1.3 Ideen zum Beweis

Wie beweist man nun Theorem 1.2? Nehmen wir einmal an, wir könnten zeigen, daß für irgendein $\varepsilon > 0$ das folgende Problem $\mathcal{NP}$-vollständig ist:

ε-ROBUST 3-SATISFIABILITY:

`Eingabe:` *Eine* 3SAT *Instanz I für die es entweder eine erfüllende Wahrheitsbelegung gibt oder bei der für jede Wahrheitsbelegung mindestens ein ε-Anteil aller Klauseln nicht erfüllt sind.*

`Frage:` *Gibt es eine erfüllende Wahrheitsbelegung für I?*

Dann wäre es in der Tat sehr einfach, Theorem 1.2 zu zeigen. Der Beweis B würde gerade aus einer Wahrheitsbelegung bestehen. Für deren Überprüfung würde der Verifier zufällig eine Klausel wählen (für deren Wahl genügen $\mathcal{O}(\log m)$ viele zufällige Bits, wobei m die Anzahl Klauseln ist), die Wahrheitsbelegung der darin vorkommenden Variablen erfragen (3 Bits) und überprüfen, daß die ausgewählte Klausel für diese Belegung erfüllt ist. Falls die Wahrheitsbelegung des Beweises B die 3SAT Instanz I erfüllt, ist – unabhängig von der Wahl der Klausel – jeder Test erfolgreich. Erfüllt andererseits die Belegung aus B die Instanz nicht, so ist – gemäß der Voraussetzung an I – die Wahrscheinlichkeit, daß der Verifier eine erfüllte Klausel wählt höchstens $1-\varepsilon$. Wiederholt er diesen Test nun $k = \lceil \log_{1-\varepsilon} \frac{1}{2} \rceil$ mal, verwirft er mit Wahrscheinlichkeit mindestens $1-(1-\varepsilon)^k \geq \frac{1}{2}$ in mindestens einem dieser Tests.

Wie wir später sehen werden, ist das Problem ε-ROBUST 3-SATISFIABILITY für hinreichend kleines $\varepsilon > 0$ tatsächlich $\mathcal{NP}$-vollständig. Nur, der einzig bisher bekannte Beweis hierfür basiert auf Theorem 1.2! Dieses Resultat müssen wir also anders beweisen.

Auf den folgenden Seiten wollen wir versuchen, dem Leser einige der grundlegenden Ideen des Beweises von Theorem 1.2 nahezubringen. Eine ausführlichere Darstellung auch nur der verwendeten Ideen und Methoden würde den Rahmen dieses Artikels allerdings bei weitem sprengen. Hierfür müssen wir den interessierten Leser auf die Originalarbeit verweisen. Für eine vollständige Darstellung des Beweises empfehlen wir die Dissertation von Arora (1994) oder auch die Arbeit von Hougardy, Prömel und Steger (1994).

Die entscheidende Idee ist die Beobachtung, daß es im Grunde gar nicht wichtig ist, den "eigentlichen" Beweis (die Wahrheitsbelegung) zu lesen. Genauer gesagt, ob beispielsweise die Variable x_{10} einer 3SAT Formel den Wert 0 oder 1 hat ist vollkommen uninteressant. Wichtig ist nur, welche Klauseln sie erfüllt!

Was aber hilft uns diese banale Erkenntnis? Der Trick ist, im Beweis für den Verifier den "eigentlichen" Beweis (also beispielsweise die Wahrheitsbelegung) so zu "kodieren", daß er möglichst viele Zusatzinformationen enthält, die dem Verifier das Überprüfen des Beweises erleichtern.

Betrachten wir ein Beispiel: Unser Ziel ist es, einen 0-1 Vektor y der Länge n so zu kodieren, daß man durch das Lesen von nur *konstant* vielen Bits mit Fehlerwahrscheinlichkeit 1/2 prüfen kann, ob y identisch mit einem vorgegebenen Vektor $a \in \mathbb{F}_2^n$ ist?[2] Was auf den ersten Blick eigentlich unmöglich erscheint, ist in der Tat recht einfach. Betrachten wir das Skalarprodukt $y^T r$ (in $\mathbb{F}_2$) von y mit einem beliebigen Vektor $r \in \mathbb{F}_2^n$ und vergleichen es mit $a^T r$. Offensichtlich sind für das Ergebnis nur die Koordinaten von r erheblich, an denen y und a verschiedene Werte haben. Und $y^T r$ ist genau dann gleich $a^T r$, wenn r an einer geraden Anzahl dieser Stellen eine 1 enthält. Insbesondere ist $y^T r = a^T r$ für alle Vektoren r, falls y und a identisch sind. Ansonsten ist $y^T r$ aus Symmetriegründen für genau die Hälfte aller möglichen r-Vektoren gleich $a^T r$. Mit anderen Worten, wir haben

$$\mathrm{Prob}_{r \in \mathbb{F}_2^n}[y^T r = a^T r] = \begin{cases} 1, & \text{falls } y = a \text{ und} \\ \frac{1}{2}, & \text{falls } y \neq a. \end{cases} \tag{1.1}$$

[2] $\mathbb{F}_2$ bezeichnet den Körper, der aus den Elementen 0 und 1 besteht.

Als Kodierung y' unseres eigentlichen "Beweises" y wählen wir daher einfach eine Tabellierung aller Skalarprodukte $y^T r$ für $r \in \mathbb{F}_2^n$. Wie wird dann der Verifier vorgehen? Nun, zunächst muß er sicherstellen, daß der vorliegende Beweis y' die verlangte Form hat, das heißt, die Tabellierung einer homogenen linearen Funktion $y^T r$ ist, und, falls dies der Fall ist, wird er in einem zweiten Schritt die Eigenschaft (1.1) verwenden, um zu verifizieren, daß y und a identisch sind. Schauen wir uns den ersten Schritt etwas genauer an. Wie kann man überprüfen, daß y' die Tabellierung einer homogenen linearen Funktion ist? — Indem wir prüfen, daß die Linearitätsbedingung an zwei zufällig gewählten Stellen erfüllt ist. Dazu interpretieren wir y' als Funktion $y' : \mathbb{F}_2^n \to \mathbb{F}_2$ und testen, ob

$$y'(r) + y'(s) = y'(r+s) \qquad \text{für zufällig gewählte } r, s \in \mathbb{F}_2^n.$$

Das folgende Lemma, das nicht schwer zu beweisen ist, garantiert nun, daß diese Vorgehensweise das gewünschte Ergebnis liefert:

Lemma 1.1. *Ist $\delta < \frac{1}{6}$ eine Konstante und $\tilde{g} : \mathbb{F}_2^n \to \mathbb{F}_2$ eine Funktion mit $Prob_{r,s}\,[\tilde{g}(r) + \tilde{g}(s) \neq \tilde{g}(r+s)] \leq \delta$, dann gibt es ein $y \in \mathbb{F}_2^n$, so daß für die beiden Funktionen $g(r) = y^T r$ und $\tilde{g}(r)$ gilt: $Prob_r\,[g(r) \neq \tilde{g}(r)] \leq 2\,\delta$.*

Unser Algorithmus sieht damit folgendermaßen aus:

<u>EQUALITY TESTER</u>

`Eingabe:` Vektor $a \in \mathbb{F}_2^n$ und Zugriff auf eine Funktion $y' : \mathbb{F}_2^n \to \mathbb{F}_2$.
`Ausgabe:` ACCEPT oder REJECT.
Wiederhole 6 mal:
  Wähle $r, s \in \mathbb{F}_2^n$ zufällig,
  if $y'(r) + y'(s) \neq y'(r+s)$ **then Return**(REJECT).
Wiederhole 3 mal:
  Wähle $r \in \mathbb{F}_2^n$ zufällig,
  if $y'(r) \neq a^T r$ **then Return**(REJECT).
Return(ACCEPT).

Was können wir über diesen Algorithmus aussagen? Falls y' die "richtige" Kodierung des "richtigen" Beweis ist (das heißt, falls $y'(r) = a^T r$ für alle $r \in \mathbb{F}_2^n$), so stoppt der Algorithmus *immer* mit ACCEPT. Ist andererseits y' von der richtigen Kodierung "weit entfernt", das heißt, gilt $\text{Prob}_r\,[y'(r) \neq a^T r] \geq 1/4$, so gibt der Algorithmus ACCEPT höchstens mit Wahrscheinlichkeit $1/2$ aus. Um dies zu zeigen, unterscheiden wir zwei Fälle. Nehmen wir zunächst an, daß es *keinen* Vektor $y \in \mathbb{F}_2^n$ mit $\text{Prob}_r\,[y'(r) \neq y^T r] \leq 1/4$ gibt. Dann folgt aus Lemma 1.1, daß $\text{Prob}_{r,s}\,[y'(r) + y'(s) \neq y'(r+s)] > 1/8$. Die Wahrscheinlichkeit, daß bei einer 6-maligen Wiederholung des Linearitätstests kein Fehler entdeckt wird, ist daher höchstens $(1 - \frac{1}{8})^6 < 1/2$. Gibt es andererseits ein $y \in \mathbb{F}_2^n$ mit $\text{Prob}_r\,[y'(r) \neq y^T r] \leq 1/4$, dann folgt aus (1.1), daß

$$\begin{aligned}\mathrm{Prob}_r\,[y'(r) = a^T r] \;&\leq\; \mathrm{Prob}_r\,[y'(r) = a^T r \wedge y'(r) = y^T r] \\ &\quad + \mathrm{Prob}_r\,[y'(r) = a^T r \wedge y'(r) \neq y^T r] \\ &\leq\; \mathrm{Prob}_r\,[a^T r = y^T r] + \mathrm{Prob}_r\,[y'(r) \neq y^T r] \;\leq\; \tfrac{1}{2} + \tfrac{1}{4}.\end{aligned}$$

Die Wahrscheinlichkeit, daß bei einer 3-maligen Wiederholung dieses Tests kein Fehler entdeckt wird, ist daher höchstens $(3/4)^3 < 1/2$.

1.4 Approximationsprobleme

Wir wenden uns nun einem scheinbar angewandteren Problem zu, als der Verifikation von Beweisen, und kommen zu den angekündigten Anwendungen der oben vorgestellten Ergebnisse auf Fragen der effizienten Approximierbarkeit von Lösungen $\mathcal{NP}$-schwerer diskreter Optimierungsprobleme.

Für die diskrete Optimierung führte das Konzept der $\mathcal{NP}$-Vollständigkeit zunächst nur zu niederschmetternden Ergebnissen: Fast jede interessante Frage, eingeschlossen das Rucksackproblem, das Set Cover Problem und das Cliquenproblem, stellte sich als $\mathcal{NP}$-schwer und damit als vermutlich nicht in polynomialer Zeit lösbar dar. Vom Standpunkt des Praktikers ist jedoch nur selten die Kenntnis der optimalen Lösung wirklich notwendig, eine gute Näherung reicht in der Regel aus. Dies hat zu einem großen Interesse an Approximationsfragen geführt und zunächst auch zu etlichen intelligenten Approximationsalgorithmen für diskrete Optimierungsprobleme. Meist offen blieb jedoch die Frage, warum man nicht besser approximieren kann als die teilweise mit großer Mühe gefundenen Approximationsalgorithmen es zeigen. Zumindest teilweise soll diese Frage hier beantwortet werden. Bevor wir dies tun, soll zunächst kurz daran erinnert werden, was man allgemein unter einem Optimierungsproblem versteht.

Ein Optimierungsproblem Π ist ein Tripel $\Pi = (\mathcal{I}, \mathcal{L}, w)$, wobei $\mathcal{I}$ die Menge der Eingaben (Instanzen) für Π ist, $\mathcal{L}(I)$ zu einer gegebenen Eingabe I die Menge der zulässigen Lösungen bezeichnet (wobei wir im folgenden immer annehmen wollen, daß $\mathcal{L}(I)$ nicht leer ist), und schließlich $w(I, L)$ zu einer Eingabe I und einer Lösung $L \in \mathcal{L}(I)$ den Wert der Lösung L angibt. Optimierungsprobleme sind entweder Minimierungsprobleme oder Maximierungsprobleme. Mit $\mathrm{Opt}(I)$ bezeichnen wir den Wert einer optimalen Lösung für eine Eingabe I, das heißt, $\mathrm{Opt}(I) = \min_{L \in \mathcal{L}(I)} w(I, L)$, falls Π ein Minimierungsproblem ist, $\mathrm{Opt}(I) = \max_{L \in \mathcal{L}(I)} w(I, L)$, falls es sich bei Π um ein Maximierungsproblem handelt. Die (relative) Güte einer Lösung $L \in \mathcal{L}(I)$ bezüglich der optimalen Lösung $\mathrm{Opt}(I)$ zu einer Eingabe wird durch den Quotienten

$$R(I, L) \;:=\; \frac{w(I, L)}{\mathrm{Opt}(I)},$$

falls Π ein Minimierungproblem ist und

$$R(I, L) \;:=\; \frac{\mathrm{Opt}(I)}{w(I, L)},$$

falls Π ein Maximierungsproblem ist, gemessen.

Betrachten wir als ein erstes Beispiel für ein Optimierungsproblem eine Maximierungsversion des 3-COLORING Problems, das sogenannte MAXIMUM 3-COLORABLE SUBGRAPH Problem, abgekürzt MAX-3COL. Gegeben ist ein endlicher Graph $G = (V, E)$, gesucht ist eine (kardinalitäts-)maximale Teilmenge $E' \subseteq E$ der Kantenmenge von G für die die Knoten in V so mit 3 Farben gefärbt werden können, daß in dem von E' aufgespannten Teilgraph $G' = (V, E')$ benachbarte Knoten jeweils verschieden gefärbt sind.

Die Eingaben $\mathcal{I}$ bei diesem Maximierungsproblem sind alle endlichen Graphen. Zu einem Graphen $G = (V, E)$ gibt $\mathcal{L}(G) \subseteq \mathcal{P}(E)$ die Menge der Kantenmengen von G an, die einen 3-färbbaren Teilgraphen aufspannen. Die Zielfunktion $w(G, E')$ hat gerade die Größe des von E' aufgespannten 3-färbbaren Teilgraphen als Wert, das heißt $w(G, E') = |E'|$, und $\mathrm{Opt}(G) = \max_{E' \in \mathcal{L}(G)} |E'|$ ist die Größe des maximalen 3-färbbaren Teilgraphen von G.

Sei A ein Approximationsalgorithmus für ein Optimierungproblem Π und bezeichne $A(I) \in \mathcal{L}(I)$ für eine gegebene Instanz I die Lösung, die A ausgibt. Die Güte von A bezüglich der Eingabe I wird dann gemessen durch

$$R_A(I) \ := \ R(I, A(I)).$$

falls Π ein Maximierungsproblem ist. Je näher der Wert $R_A(I)$ bei 1 liegt, desto besser ist offensichtlich das Ergebnis, das A liefert. Ist $R_A(I) = 1$, so ist das von A ausgegebene Ergebnis bereits optimal. Sei $f : \mathbb{N} \to \mathbb{N}$ eine beliebige Funktion. A ist dann ein $f(n)$-Approximationsalgorithmus für ein Optimierungsproblem Π, falls für jede Eingabe $I \in \mathcal{I}$ der Länge $|I|$ gilt:

$$R_A(I) \ \leq \ f(|I|).$$

Wie kann nun ein Approximationsalgorithmus für MAX-3COL aussehen? Zu einem gegebenen Graphen $G = (V, E)$ und einer 3-Färbung c der Knoten von G bezeichnen wir mit $a(v, i)$ die Anzahl der Nachbarn des Knoten $v \in V$ in der Farbe i, das heißt

$$a(v, i) := |\{w \in V : \{v, w\} \in E \text{ und } c(w) = i\}|.$$

Gilt für einen Knoten $v \in V$, daß $a(v, c(v)) > \min_i a(v, i)$, so kann man die Anzahl "falsch" gefärbter Kanten in G dadurch reduzieren, daß man den Knoten v mit einer Farbe j_0 mit $a(v, j_0) = \min_i a(v, i)$ neu färbt. Auf dieser simplen Idee basiert nun der Approximationsalgorithmus C:

Man beginne mit einer beliebigen 3-Färbung des Graphen $G = (V, E)$ und färbe solange sukzessive Knoten um, bis für die resultierende Färbung $c : V \to \{1, 2, 3\}$ an jedem Knoten $v \in V$ gilt: $a(v, c(v)) = \min_i a(v, i)$.

Da $\min_i a(v, i) \leq \frac{1}{3} |\{w \in V : \{v, w\} \in E\}|$ für jedes $v \in V$, ist die Menge der einfarbigen Kanten in G, wenn der Algorithmus C terminiert, höchstens

$\frac{1}{3}|E|$. Mit anderen Worten $R_C(G) \leq \frac{3}{2}$ für jeden Graphen G. Dieser $\frac{3}{2}$-Approximationsalgorithmus ist der beste derzeit bekannte Approximationsalgorithmus für Max-3Col.

Schauen wir uns nun ein $\mathcal{NP}$-schweres Optimierungsproblem an, dessen optimale Lösung sich noch besser approximieren läßt. Das Rucksackproblem ist das folgende Maximierungsproblem: Gegeben n Gegenstände, sagen wir $1, \ldots, n$. Jeder Gegenstand i hat eine Größe s_i und einen Wert w_i, beides positive ganze Zahlen. Ferner ist eine positive ganze Zahl S gegeben, die Gesamtgröße des Rucksacks. Gesucht ist nun eine wertmaximale Ausnutzung des Rucksacks, das heißt, eine Teilmenge $U \subseteq \{1, \ldots, n\}$, so daß $\sum_{i \in U} s_i \leq S$ und $\sum_{i \in U} w_i$ maximal unter dieser Nebenbedingung ist. Das Rucksackproblem ist $\mathcal{NP}$-schwer. Es ist aber schon seit langem bekannt, daß es einen Algorithmus A gibt, der zu jeder Eingabe I und jedem $\varepsilon > 0$ in polynomialer Zeit in der Länge der Eingabe $|I|$ eine Lösung $A(I, \varepsilon)$ für das Rucksackproblem berechnet, dessen Approximationsgüte durch $1 + \varepsilon$ beschränkt ist.

Die Menge aller diskreten Optimierungsprobleme für die es ein solches polynomiales Approximationsschema gibt, wird mit $\mathcal{PAS}$ bezeichnet. Man beachte, daß die Zeitkomplexität eines solchen Approximationsschemas beispielsweise von der Form $|I|^{1/\varepsilon}$ sein kann. Für jedes feste $\varepsilon > 0$ ist die Laufzeit des Verfahrens dann zwar polynomial, für Werte von ε nahe 0 werden die Laufzeiten jedoch völlig unpraktikabel. Für das Rucksackproblem kann man diese Problematik umgehen. Hierfür kennt man sogar einen Algorithmus, der zu jeder Eingabe I und zu jedem $\varepsilon > 0$ in einer Anzahl von Schritten, die durch ein Polynom in der Länge der Eingabe $|I|$ und in ε^{-1} beschränkt ist, eine Lösung zum Rucksackproblem generiert und dessen Approximationsgüte durch $1+\varepsilon$ beschränkt ist. Die Menge aller diskreten Optimierungsprobleme, für die es solch ein Approximationsschema gibt, das durch ein Polynom in der Länge der Eingabe und in ε^{-1} beschränkt ist, wird mit $\mathcal{FPAS}$ bezeichnet. Für Probleme, die zu $\mathcal{PAS}$ oder gar $\mathcal{FPAS}$ gehören, ist, zumindest aus der Sicht eines Theoretikers, die $\mathcal{NP}$-Schwere eines Problems daher kein Hindernis, effizient eine Lösung zu finden, die den meisten Anforderungen Genüge tut.

1.5 $\mathcal{APX}$-schwere Probleme

Mitte der 80er Jahre wurde es mehr und mehr offensichtlich, daß, von wenigen Ausnahmen abgesehen, die meisten interessanten diskreten Optimierungsprobleme schwer zu approximieren sind, also nicht zu $\mathcal{PAS}$ oder $\mathcal{FPAS}$ gehören. Diese Einsicht wurde von Papadimitriou und Yannakakis (1991) durch die Einführung des Begriffes der $\mathcal{MAXSNP}$-schweren Probleme formalisiert. Die Definition war motiviert durch Fagins syntaktische Definition der Klasse $\mathcal{NP}$ mit Hilfe von endlichen Modellen der Logik zweiter Stufe. Auf eine Wiedergabe der Definition soll hier verzichtet werden.

Wir betrachten statt dessen die Klasse $\mathcal{APX}$ aller Probleme, die bis auf einen konstanten Faktor approximiert werden können. Wie Khanna, Motwani, Sudan und Vazirani (1994) zeigten, ist die Klasse $\mathcal{MAXSNP}$ (schließt man sie geeignet unter sogenannten L-Reduktionen ab) gleich der Klasse $\mathcal{APX}$.

Wie wir bereits gesehen haben, besitzt das Max-3Col Problem einen $\frac{3}{2}$-Approximationsalgorithmus, liegt also in der Klasse $\mathcal{APX}$. Wir wollen nun ein weiteres Problem in $\mathcal{APX}$ kennenlernen, das Max-3Sat Problem. Max-3Sat ist gerade die Maximierungsversion des üblichen 3Sat Problems. Gegeben ist eine 3Sat Formel, gesucht eine Belegung der Variablen, die möglichst viele Klauseln wahr werden läßt. Es ist leicht, eine Belegung anzugeben, die mindestens die Hälfte der Klauseln erfüllt:

Man belege sukzessive die Variablen $x_1, \ldots, x_n$ so mit "wahr" oder "falsch", daß im $i$ten Schritt mindestens die Hälfte der Klauseln, in denen x_i aber keine der Variablen $x_1, \ldots, x_{i-1}$ vorkommt, erfüllt werden.

Dieser lineare Approximationsalgorithmus hat eine Approximationsgüte von 2. Der beste bekannte polynomiale Approximationsalgorithmus für das Max-3Sat Problem, hat eine Güte von $\frac{8}{7}$. Kurz und gut, das Max-3Sat Problem liegt auch in der Klasse $\mathcal{APX}$.

Um nun die schwierigsten Probleme in $\mathcal{APX}$ herauszufiltern, und dies war gerade die Intention der Begriffsbildung von Papadimitriou und Yannakakis, bedarf es eines geeigneten Reduktionsbegriffs. Ein Optimierungsproblem $\Pi = (\mathcal{I}, \mathcal{L}, w)$ heißt $\mathcal{APX}$-reduzierbar auf ein Optimierungsproblem $\Pi^* = (\mathcal{I}^*, \mathcal{L}^*, w^*)$, falls es in polynomialer Zeit berechenbare Funktionen f und g und eine Konstante $\rho > 0$ gibt, wobei $f : \mathcal{I} \longrightarrow \mathcal{I}^*$ die Eingaben von Π auf die Eingaben von Π^* und $g : \mathcal{L}(f(I)) \longrightarrow \mathcal{L}(I)$ für jedes $I \in \mathcal{I}$ die Lösungen von $f(I)$ auf die Lösungen von I abbildet, so daß für alle $L^* \in \mathcal{L}(f(I))$ gilt:

$$R(f(I), L^*) \;\leq\; 1 + \rho\varepsilon \quad \Longrightarrow \quad R(I, g(L^*)) \;\leq 1 + \varepsilon.$$

Läßt sich also Π^* in polynomialer Zeit bis auf einen Faktor $1+\rho\varepsilon$ approximieren, so läßt sich Π in polynomialer Zeit bis auf einen Faktor $1+\varepsilon$ approximieren. Ein Optimierungsproblem Π^* heißt nun $\mathcal{APX}$-schwer, falls jedes Problem Π in $\mathcal{APX}$, auf Π^* $\mathcal{APX}$-reduzierbar ist. Π^* heißt $\mathcal{APX}$-vollständig, falls Π^* $\mathcal{APX}$-schwer ist und $\Pi^* \in \mathcal{APX}$. Mit anderen Worten, kann man von einem $\mathcal{APX}$-vollständigen Problem zeigen, daß es in $\mathcal{PAS}$ liegt, so muß bereits jedes $\mathcal{APX}$-vollständige Problem ein polynomiales Approximationsschema haben. Und umgekehrt, kann von einem $\mathcal{APX}$-vollständigen Problem gesagt werden, daß es nicht in $\mathcal{PAS}$ liegt, so hat man die Klassen $\mathcal{PAS}$ und $\mathcal{APX}$ voneinander getrennt und die Hoffnung auf ein polynomiales Approximationsschema für viele diskrete Optimierungsprobleme zunichte gemacht.

Welche Probleme sind nun $\mathcal{APX}$-vollständig? Zunächst einmal die beiden Probleme, die wir schon kennen:

Theorem 1.3. Max-3Sat *und* Max-3Col *sind* $\mathcal{APX}$*-vollständig.*

Darüberhinaus sind jedoch auch viele andere interessante Optimierungsprobleme, auf die wir hier nicht weiter eingehen wollen, $\mathcal{APX}$-vollständig. Beispielsweise sei für den graphentheoretisch interessierten Leser vermerkt, daß das Steinerbaum Problem in Netzwerken, das sogenannte Max Cut Problem und das metrische Traveling Salesperson Problem $\mathcal{APX}$-vollständig sind.

Als eine erste Anwendung des Resultats $\mathcal{NP} = \mathcal{PCP}(\log n, 1)$ zeigen wir nun, daß Max-3Sat nicht in $\mathcal{PAS}$ liegt und damit also insbesondere kein $\mathcal{APX}$-schweres Problem ein polynomiales Approximationsschema besitzen kann, außer, es gilt $\mathcal{P} = \mathcal{NP}$.

Sei $0 < \varepsilon < 1$. Eine Abbildung ϕ, die jede 3Sat Formel auf eine 3Sat Formel abbildet, heißt ε-robust, falls für jede Eingabe F gilt: Ist F erfüllbar, dann ist auch $\phi(F)$ erfüllbar und ist F nicht erfüllbar, dann ist für jede Belegung der Variablen von $\phi(F)$ mindestens ein ε-Anteil aller Klauseln nicht erfüllt.

Offenbar bildet solch eine ε-robuste Abbildung ϕ beliebige 3Sat Instanzen auf Instanzen von ε-Robust 3-Satisfiability ab. Aus der Existenz einer polynomial berechenbaren ε-robusten Abbildung folgt damit also insbesondere, daß letzteres $\mathcal{NP}$-vollständig ist.

Theorem 1.4. *Es gibt ein* $\varepsilon > 0$*, so daß eine polynomiale* ε*-robuste Reduktion* ϕ *für* 3Sat *existiert.*

Der Beweis stützt sich ganz maßgeblich auf Theorem 1.2. Danach wissen wir, daß es für 3Sat einen $(\log n, 1)$-beschränkter Verifier V gibt. Diesen werden wir jetzt verwenden, um aus einer gegebenen 3Sat Formel F in polynomialer Zeit eine neue, ε-robuste 3Sat Formel $\phi(F)$ zu konstruieren.

Dazu interpretieren wir jede Position eines gegebenen Beweises B als eine Variable, das heißt,

$$B = (x_1, x_2, x_3, \ldots).$$

Für eine Zufallsfolge τ der Länge $\mathcal{O}(\log n)$ erfragt V nun $q = \mathcal{O}(1)$ Bits des Beweises B, sagen wir $x_{\tau_1}, \ldots, x_{\tau_q}$.

Unter Verwendung des Verifiers V konstruieren wir zu jeder Zufallsfolge τ eine 3Sat Formel F_τ mit Variablen $x_{\tau_1}, \ldots, x_{\tau_q}$ so, daß $V(F, B, \tau)$ die Antworten $x_{\tau_1}, \ldots, x_{\tau_q}$ genau dann akzeptiert, wenn $x_{\tau_1}, \ldots, x_{\tau_q}$ eine Wahrheitsbelegung für F_τ ist. Dies ist offensichtlich leicht möglich: Man beginnt mit einer disjunktiven Normalform in der jede Klausel einer erfüllenden Belegung entspricht und transformiert diese dann in eine 3Sat Formel. Insbesondere gibt es daher auch eine Konstante $k = \mathcal{O}(1)$, so daß jedes F_τ höchstens k Klauseln enthält.

Wenn F erfüllbar ist, gibt es offenbar eine Wahrheitsbelegung, die *alle* F_τ erfüllt. Ist F hingegen nicht erfüllbar, so sind für jede Belegung der x_i mindestens die Hälfte der F_τ nicht erfüllbar. In der Formel

$$\phi(F) = \bigwedge_{\tau} F_\tau$$

ist dann also mindestens ein $\frac{1}{2k}$-Anteil aller Klauseln nicht erfüllbar. Damit haben wir Theorem 1.4 für $\varepsilon = \frac{1}{2k}$ bewiesen. Hieraus folgt insbesondere:

Korollar 1.2. *Falls $\mathcal{P} \neq \mathcal{NP}$, so hat* Max-3Sat *kein polynomiales Approximationsschema.*

Mit anderen Worten, es gibt eine Konstante $C > 0$, so daß Max-3Sat nicht bis auf den Faktor $1 + C$ in polynomialer Zeit approximiert werden kann, es sei denn, es gilt $\mathcal{P} = \mathcal{NP}$.

Die beste derzeit bekannte Konstante ist $C = 0,027$ (Bellare, Goldreich und Sudan, 1995), das heißt, es ist nicht möglich, Max-3Sat bis auf 2,7% genau in polynomialer Zeit zu approximieren, während man es, wie erwähnt, bis auf 14,3% genau in polynomialer Zeit approximieren kann.

Korollar 1.3. *Falls $P \neq \mathcal{NP}$, so gibt es zu jedem $\mathcal{APX}$-schweren Problem eine Konstante $C > 0$, so daß das Problem nicht bis auf den Faktor $1 + C$ in polynomialer Zeit approximiert werden kann.*

1.6 Jenseits von $\mathcal{APX}$

Eine andere Qualität von Nichtapproximierbarkeitsresultaten, die auch mit Hilfe des $\mathcal{PCP}$-Resultats erzielt werden können, werden uns in diesem und im nächsten Absatz begegnen.

Bisher haben wir Probleme kennengelernt, deren optimale Lösungen sich in polynomialer Zeit "gut" approximieren lassen, das heißt, entweder bis auf einen Faktor $1 + \varepsilon$ für *jedes* $\varepsilon > 0$, also Probleme, die zu $\mathcal{PAS}$ gehören, oder bis auf einen Faktor $1 + \varepsilon_0$ für *ein festes* $\varepsilon_0 > 0$ approximieren lassen, also Probleme, die in $\mathcal{APX}$ liegen. Etliche grundlegende und ausgiebig untersuchte $\mathcal{NP}$-schwere Probleme der diskreten Optimierung haben wir jedoch bisher nicht betrachtet.

Der Grund dafür ist, daß sich diese Probleme nur "schlecht" approximieren lassen. Natürlich kann man versuchen zu jedem Optimierungsproblem eine möglichst langsam wachsende Funktion $f(n)$ anzugeben, von der man zeigen kann, daß es einen polynomialen Algorithmus gibt, der eine optimale Lösung des Problems bis auf einen Faktor $f(n)$, wobei n die Länge der Eingabe ist, approximiert. Wählen wir $f(n) = n$, so ist ein solcher Approximationsalgorithmus meist trivial. Wir werden nun sehen, daß man für etliche Probleme untere Schranken für die Approximationsgüte beweisen kann, die mit n wachsen, das heißt, daß sich eine optimale Lösung dieser Probleme in polynomialer Zeit nicht bis auf eine Konstante genau approximieren läßt.

Von besonderer Bedeutung in diesem Zusammenhang ist das Set Cover Problem. Zum einen, weil sich eine Lösung dieses Problems noch "relativ gut" approximieren läßt, zum anderen, weil es eines der wenigen Probleme ist, für das man den Schwellenwert für die Approximationsgüte kennt.

<u>SET COVER:</u>

Eingabe: *Eine endliche Menge S mit $|S| = n$ und eine Familie $\mathcal{F} = \{S_1, S_2, \dots, S_m\}$ von Teilmengen von S.*
Problem: *Finde eine minimale Teilfamilie $\mathcal{F}' \subseteq \mathcal{F}$, so daß jedes Element in S in mindestens einer der Mengen von $\mathcal{F}'$ enthalten ist.*

Betrachten wir den folgenden einfachen Greedyalgorithmus. Beginnend mit $\mathcal{F}' = \emptyset$ fügt man so lange Mengen zu $\mathcal{F}'$ hinzu, bis ganz S überdeckt ist. Dabei wählt man im $j$ten Schritt gerade die Menge S_{i_j}, die die meisten derjenigen Knoten überdeckt, die bisher noch nicht überdeckt sind. Man kann zeigen, daß dies ein polynomialer $H(n)$-Approximationsalgorithmus ist, wobei $H(n) = \ln n + \mathcal{O}(1)$ die Harmonische Funktion ist.

Feige (1996) bewies nun, daß man die Existenz eines besseren polynomialen Approximationsalgorithmus auch nicht erwarten kann. Die Annahme "$\mathcal{P} \neq \mathcal{NP}$", unter der wir die Nichtapproximierbarkeit von $\mathcal{APX}$-schweren Problemen in polynomialer Zeit bewiesen haben, läßt sich als "$\mathcal{NP} \not\subseteq \mathrm{DTIME}(poly(n))$" reformulieren. Unter der etwas stärkeren Annahme, daß sogar $\mathcal{NP} \not\subseteq \mathrm{DTIME}(n^{\mathcal{O}(\log\log n)})$, zeigte Feige, daß es kein $\varepsilon > 0$ gibt, so daß man SET COVER bis auf einen Faktor $(1-\varepsilon)\ln n$ in polynomialer Zeit approximieren kann. Unter dieser Annahme ist $\ln n$ also der Schwellenwert für die Approximationsgüte des SET COVER Problems. Es ist jedoch ein offenes Problem, ob es $\mathcal{NP}$-schwer ist, SET COVER bis auf einen Faktor $\Omega(\log n)$ zu approximieren.

Andere Beispiele mit einem Schwellenwert $\ln n$ sind natürlich die zum SET COVER Problem äquivalenten Probleme, wie beispielsweise das Dominating Set Problem. Darüber hinaus sind bisher nur sehr wenige Probleme bekannt, die einen nicht trivialen (das heißt, von 1 verschiedenen) Schwellenwert für die Approximationsgüte haben. Ein solches Beispiel ist das sogenannte Minimum p-Center Problem, zu dessen Lösung Hochbaum und Shmoys einen einfachen polynomialen 2-Approximationsalgorithmus angegeben haben, von dem Hsu und Nemhauser aber andererseits gezeigt haben, daß es $\mathcal{NP}$-schwer ist, eine Approximationsgüte von kleiner als 2 zu erzielen.

1.7 Das Cliquen Problem

Man erinnere sich, daß eine Clique in einem Graphen eine Menge von Knoten ist, die paarweise durch Kanten verbunden sind.

Die Größe $\omega(G)$ einer größten Clique eines Graphen G ist einer der fundamentalen Parameter in der Graphentheorie mit Anwendungen in so scheinbar weit entfernten Gebieten wie der kommutativen Algebra oder der Statistik. Seit Karps grundlegender Arbeit "Reducibility among combinatorial problems" ist bekannt, daß das Problem, $\omega(G)$ zu bestimmen, $\mathcal{NP}$-schwer ist. Wegen der Wichtigkeit des Problems haben sich viele Informatiker und Mathematiker daran versucht, Approximationsalgorithmen für die Bestimmung einer größten Clique zu entwerfen – mit vergleichsweise sehr geringem

Erfolg. Der beste derzeit bekannte polynomiale Approximationsalgorithmus stammt von Boppana und Halldórsson und hat eine Approximationsgüte von $n/(\log n)^2$. Mit anderen Worten, der Algorithmus liefert, angesetzt auf einen Graphen G auf n Knoten, unter Umständen eine Clique in G, deren Größe lediglich $\omega(G)\cdot(\log n)^2/n$ beträgt, obwohl es auch eine der Größe $\omega(G)$ gibt — für wahr ein bescheidenes Ergebnis! Der Grund für diese schlechte Ausbeute wurde durch eine erstaunliche Verbindung zwischen dem Cliquenproblem und den probabilistisch verifizierbaren Beweisen klar, die von Feige, Goldwasser, Lovász, Safra und Szegedy (1991) entdeckt wurde.

Als ersten Schritt hierzu zeigen wir, daß das Problem CLIQUE (das heißt, gegeben ein endlicher Graph $G = (V, E)$, bestimme eine größte Clique in G) kein polynomiales Approximationsschema besitzt. Wegen Korollar 1.3 ist hierfür nur zu zeigen:

Proposition 1.2. *Das Problem* CLIQUE *ist* $\mathcal{APX}$*-schwer.*

Zum Beweis reduzieren wir das MAX-3SAT Problem, von dem wir bereits gesehen haben, daß es $\mathcal{APX}$-schwer ist, auf das Problem CLIQUE. Wir benutzen dazu eine der bekannten Reduktionen von 3SAT auf die Entscheidungsversion von CLIQUE. Dabei beobachten wir, daß dies bereits eine $\mathcal{APX}$-Reduktion ist.

Sei F also eine 3SAT Formel mit m Klauseln und n Variablen. Wir ordnen F einen Graphen $G(F)$ auf $3\,m$ Knoten wie folgt zu. Zunächst wählen wir für jede Klausel 3 Knoten, für jedes Literal in der Klausel einen. Wir verbinden nun zwei Knoten genau dann miteinander, wenn sie weder zu derselben Klausel gehören noch der eine Knoten die Negation des anderen ist.

Jede Clique in $G(F)$ kann also aus jeder Klausel nur höchstens ein Literal enthalten und niemals ein Literal und seine Negation. Mit anderen Worten, jede Clique entspricht einer partiellen Wahrheitsbelegung, die mindestens soviele Klauseln erfüllt, wie die Kardinalität der Clique ist. Und umgekehrt zeigt auch jede Wahrheitsbelegung, die $c\,m$ Klauseln von F erfüllt, daß es in $G(F)$ eine Clique der Größe $c\,m$ gibt.

Aus der Tatsache, daß das Problem CLIQUE $\mathcal{APX}$-schwer ist, folgt aber nun a priori nur, daß es ein – unter Umständen verschwindend kleines – $\varepsilon > 0$ gibt, so daß, falls $\mathcal{P} \neq \mathcal{NP}$, kein Algorithmus existiert, der in jedem Graphen G in polynomialer Zeit eine Clique der Größe mindestens $\omega(G)/(1+\varepsilon)$ bestimmt. In der Tat, kann man aber leicht zeigen, daß man sogar $\varepsilon = 1$ wählen darf, daß man also in polynomialer Zeit nicht einmal eine Clique der Größe $\frac{1}{2}\omega(G)$ bestimmen kann. Um dies einzusehen, verbinden wir die Konstruktionen aus dem Beweis von Theorem 1.4 und Proposition 1.2.

Sei V ein $(\log n, 1)$-beschränkter Verifier für 3SAT. Wie im Beweis von Theorem 1.4 interpretieren wir wieder jede Position eines gegebenen Beweises B als eine Variable, also $B = (x_1, x_2, x_3, \ldots)$. Für eine Zufallsfolge τ der Länge $\mathcal{O}(\log n)$ erfragt der Verifier $q = \mathcal{O}(1)$ Bits des Beweises, sagen wir $x_{\tau_1}, \ldots, x_{\tau_q}$.

Zu einer gegebenen 3SAT Formel F konstruieren wir jetzt zu jeder Zufallsfolge τ eine stabile Menge (das heißt, eine Menge von Knoten, die paarweise nicht durch Kanten verbunden sind) wie folgt: Für jede Belegung der Variablen $x_{\tau_1}, \ldots, x_{\tau_q}$, für die der Verifier akzeptieren würde (bezüglich F und τ), fügen wir einen Knoten ein. Knoten in verschiedenen stabilen Mengen verbinden wir genau dann miteinander, wenn sie sich nicht widersprechen — wenn also entweder die verwendeten Variablen alle verschieden sind oder die in beiden Knoten vorkommenden Variablen auch alle die gleiche Belegung haben.

Der Leser führe sich vor Augen, daß diese Konstruktion genau die Idee aus dem Beweis von Proposition 1.2 widerspiegelt: Dort haben wir für jede Klausel eine stabile Menge eingeführt und die Knoten in verschiedenen stabilen Mengen dann genau so untereinander verbunden, daß eine Clique einer Wahrheitsbelegung entspricht, wobei die Größe der Clique der Anzahl der bei dieser Belegung erfüllten Klauseln entspricht. Jetzt haben wir jeder Zufallsfolge eine stabile Menge zugeordnet und die Knoten in verschiedenen stabilen Mengen genau so untereinander verbunden, daß jede Clique einem Beweis B entspricht, wobei wiederum gilt, daß die Größe der Clique der Anzahl der Zufallsfolgen entspricht, für die der Verifier die 3SAT Formel F bei Vorlage des Beweises B akzeptiert.

Wir haben also gezeigt: Gibt es für F eine erfüllende Wahrheitsbelegung, dann besitzt der konstruierte Graph eine Clique, die *jede* stabile Menge trifft. Gibt es andererseits keine solche erfüllende Wahrheitsbelegung, dann kann es keine Clique geben, die mehr als die Hälfte der stabilen Mengen trifft.

Geht es noch besser? Ja! Und wir wissen auch schon wie: In Abschnitt 1.2 hatten wir gesehen, daß die Konstante $1/2$ in der Definition der Klasse $\mathcal{PCP}(r(n), q(n))$ beliebig war. Wir hätten als Fehlerwahrscheinlichkeit auch 10^{-10} wählen können. Dann führt die obige Konstruktion aber zu Graphen, in dem sich die maximalen Cliquen um einen Faktor 10^{10} unterscheiden, in Abhängigkeit davon, ob die 3SAT Formel erfüllbar ist oder nicht. Mit anderen Worten, das Problem CLIQUE läßt sich in polynomialer Zeit nicht bis auf *irgendeine* Konstante genau approximieren!

Geht es noch besser? Nun, erst einmal ist diese Konstruktion an ihre Grenzen gelangt. Um eine Fehlerwahrscheinlichkeit zu erreichen, die $o(1)$ ist, beispielsweise $n^{-\delta}$ für ein $\delta > 0$, müßten wir den Standardverifier (mit Fehlerwahrscheinlichkeit $1/2$) mindestens $\delta \cdot \log_2 n$ oft aufrufen. Dazu würden wir dann insgesamt $\mathcal{O}((\log n)^2)$ viele Zufallsbits benötigen, was wiederum dazu führen würde, daß der oben konstruierte Graph aus $2^{\mathcal{O}((\log n)^2)}$ vielen stabilen Mengen bestehen würde – und daher nicht mehr in polynomialer Zeit konstruiert werden kann.

Doch es gibt einen Ausweg: Wir "recyclen" die zufälligen Bits! Was heißt das? Sei V ein $(\log n, 1)$-Verifier mit Fehlerwahrscheinlichkeit, sagen wir $1/16$. Rufen wir ihn $k = \delta \cdot \log n$ oft mit der *gleichen* Zufallsfolge τ auf, so benötigen wir dafür natürlich nur $\mathcal{O}(\log n)$ viele zufällige Bits – aber die Fehlerwahrscheinlichkeit reduziert sich auch nicht. Verwenden wir andererseits jedesmal eine komplett neue Zufallsfolge, so reduziert sich, wie wir oben gesehen haben, die Fehlerwahrscheinlichkeit zwar auf den gewünschten Wert $n^{-\delta}$, aber wir benötigen $\mathcal{O}((\log n)^2)$ viele zufällige Bits. Die Idee ist nun, ausgehend von einer ersten Zufallsfolge τ, die weiteren sukzessive durch Verwendung von jeweils nur *konstant* vielen zusätzlichen zufälligen Bits so zu erzeugen, daß sie sich jeweils "in etwa" wie komplett neue Zufallsfolgen verhalten.

Realisieren läßt sich dieses Recyclen der Zufallsbits durch eine Irrfahrt in einem geeignet gewählten Expandergraphen. Eine Irrfahrt in einem Graphen ist nichts anderes als eine Folge von Knoten $(v_1, v_2, v_3, \ldots)$, die dadurch bestimmt ist, daß man ausgehend von einem zufällig gewählten Startknoten v_1 rekursiv den Knoten v_i *zufällig* aus der Nachbarschaft von v_{i-1} auswählt. Ein Expandergraph andererseits hat genau die Eigenschaft, daß man mit diesem Verfahren mit "hoher Wahrscheinlichkeit" den ganzen Graphen "gleichmäßig" besucht. Genauer kann man zeigen (wobei die Konstanten nicht bestmöglich sind):

Lemma 1.2. *Zu jeder natürlichen Zahl n existiert ein 8-regulärer Expandergraph $G = (V, E)$ auf n Knoten mit der Eigenschaft, daß für jede Menge $W \subseteq V$ mit $|W| \leq \frac{1}{10}|V|$ die Wahrscheinlichkeit, daß eine Irrfahrt der Länge $10 \cdot k$ in jedem Schritt einen Knoten aus W trifft höchstens 2^{-k} ist.*

Die Vorgehensweise ist nun klar. Um Zufallsfolgen der Länge $\tilde{r}(n) = \mathcal{O}(r(n))$ zu erzeugen, wählen wir den Expandergraphen aus Lemma 1.2 bezüglich $n = 2^{\tilde{r}(n)}$ und identifizieren die Knoten mit den $2^{\tilde{r}(n)}$ verschiedenen Zufallsfolgen der Länge $\tilde{r}(n)$. Dann erzeugen wir die Knoten einer Irrfahrt der Länge $10 \cdot \delta \cdot \log n$. Dazu benötigen wir $\tilde{r}(n)$ Zufallsbits für die Wahl des Startknotens und $10 \cdot \delta \cdot \log n$ oft 3 Bits für die Wahl des nächsten Knotens. Danach rufen wir den Verifier V für jede Zufallsfolge der (Knoten der) Irrfahrt auf. Nach Lemma 1.2 ist die Wahrscheinlichkeit, daß der Verifier in jedem Aufruf akzeptiert, obwohl er eigentlich verwerfen sollte, höchstens $n^{-\delta}$.

Die Frage nach dem größten Faktor, für das sich ein solches Nichtapproximierbarkeitsergebnis beweisen läßt, hat in den letzten Jahren die Gemüter stark bewegt und dazu geführt, daß man neben der "query complexity", also der minimalen Anzahl von Anfragen, die an einen potentiellen Beweis B gestellt werden müssen, die sogenannte "free bit complexity" der $\mathcal{PCP}$-Verifier untersucht hat. Grob gesprochen ist die "free bit complexity" der Logarithmus der Anzahl der Zeichenfolgen, die der Verifier V (bei Eingabe I mit $|I| = n$ und $\tilde{r}(n)$ zufälligen Bits, die ihm zur Verfügung stehen) akzeptiert. Stellt der Verifier V beispielsweise drei Anfragen an den potentiellen Beweis B und akzeptiert genau dann, wenn die Anzahl der gelesenen Einsen ungerade ist, so gibt es vier Zeichenfolgen, die V akzeptiert. Die "free bit complexity" wäre also 2. Unter Ausnutzung von Resultaten über die "free bit complexity" von $\mathcal{PCP}$-Verifiern konnten Bellare, Goldreich und Sudan (1995) zeigen, daß es unter der Annahme $\mathcal{P} \neq \mathcal{NP}$ keinen polynomialen

Approximationsalgorithmus für die Cliquenzahl eines Graphen auf n Knoten mit einer Approximationsgüte von $n^{\frac{1}{4}}$ geben kann. Unter der Annahme $co\mathcal{RP} \neq \mathcal{NP}$ zeigten sie sogar, daß es keinen polynomialen Approximationsalgorithmus mit einer Approximationsgüte von $n^{\frac{1}{3}}$ geben kann. Aufbauend auf diesen Ergebnissen bewies Håstad (1996, 1996a), daß es unter der Annahme $co\mathcal{RP} \neq \mathcal{NP}$ sogar unmöglich ist, die Cliquenzahl bis auf einen Faktor $n^{1-\varepsilon}$ für beliebiges $\varepsilon > 0$ in polynomialer Zeit zu approximieren. Damit ist (unter der Annahme $co\mathcal{RP} \neq \mathcal{NP}$) zumindest der Schwellenwert im Logarithmus für die Approximationsgüte des Cliquenproblems bestimmt - und es ist die schlechtest mögliche Approximationsgüte!

Schriftenverzeichnis

1. S. Arora (1994). Probabilistic checking of proofs and hardness of approximation problems. Dissertation, Department of Computer Science, Princeton University.
2. S. Arora und C. Lund (1996). Hardness of approximations. Erscheint in: Approximation Algorithms for NP-Hard Problems (ed. Dorit Hochbaum).
3. S. Arora, C. Lund, R. Motwani, M. Sudan und M. Szegedy (1992). Proof verification and intractability of approximation problems. Proc. 33rd IEEE Symp. on Foundations of Computer Science, 13-22.
4. L. Babai, L. Fortnow und C. Lund (1991). Non-deterministic exponential time has two-prover interactive protocols. Computational Complexity 1, 3-40.
5. M. Bellare, O. Goldreich und M. Sudan (1995). Free bits, PCPs and nonapproximability - towards tight results. Proc. 36th IEEE Symp. on Foundations of Computer Science, 422-431.
6. U. Feige (1996). A threshold of $\ln n$ for approximating set cover. Erscheint in: Proc. 28th ACM Symp. on Theory of Computing.
7. U. Feige, S. Goldwasser, L. Lovász, S. Safra und M. Szegedy (1991). Approximating clique is almost NP-complete. Proc. 32nd IEEE Symp. on Foundations of Computer Science, 2-12.
8. J. Håstad (1996). Testing of the long code and hardness for clique. Erscheint in: Proc. 28th ACM Symp. on Theory of Computing.
9. J. Håstad (1996a). Clique is hard to approximate within $n^{1-\varepsilon}$ (Preliminary version). Royal Institute of Technology, Schweden.
10. S. Hougardy, H. J. Prömel und A. Steger (1994). Probabilistically checkable proofs and their consequences for approximation algorithms. Discrete Mathematics 136, 175-223.
11. S. Khanna, R. Motwani, M. Sudan und U. Vazirani (1994). On syntactic versus computational views of approximability. Proc. 35th Symp. on Foundation of Computer Science, 819-830.
12. C. Papadimitriou und M. Yannakakis (1991). Optimization, approximation and complexity classes. Journal of Computer and System Sciences 43, 425-440.
13. I. Wegener (1993). Theoretische Informatik. B. G. Teubner Stuttgart.